particular implementation of the invention, one or more of the aspects may be used together in various permutations and/or combinations, with the understanding that different permutations and/or combinations may be better suited for particular applications or have more or less benefits or advantages than others.

The underlying scenario common to all these basic examples is that there is a hierarchical arrangement to the possible choices that can be illustrated in a form of "tree" structure.

FIG. 1 is an example graph 100 representing a possible hierarchically arranged transaction processing or decisional system suitable for use with the invention. The individual boxes 102 - 120 are referred to as "nodes" and each represents a specific choice or option in the hierarchy. For purposes described in more detail below, each node is arbitrarily uniquely identified in some manner. In the example of FIG. 1, the individual nodes 102 - 120 are numbered 1 through 10 starting from the top node 102 in the hierarchy.

Each "node" is associated with exactly one verbal description, for example in the case of an airline system, a verbal description relating to some aspect of the reservation process. Each such description contains "key" words that are deemed to be of importance and other words that can be disregarded. For example, one node may have the associated verbal description "Would you like to make a reservation?" In this description, there is only one "key" word – "reservation" deemed important, so all of the other words in the description can be ignored.

A level in the hierarchy below that one may be used to obtain further narrowing information, for example, using the verbal description "Is the reservation for a domestic or international flight?" In this description, the terms "domestic" and "international" are "key" words. Similarly, the word "flight" could be a "key" word, for example, for a system that

214 PETITIONERS - EXHIBIT 1002 - Part 2 of 3
IPR2022-00217

involves not only airline travel but also rail and/or cruise travel or it could be an "ignored" or stop word for a purely airline related system because it has minimal meaning in that context. Again, the other words can be ignored as well.

The unique identification of each node allows the creation of a list of all the key words and their associated nodes so that, if a key word is duplicated in two or more nodes, it need only be listed once. For example, a hierarchical tree related to "pens" might have nodes for ball-point pens, fine point pens, medium point pens, fountain pens, felt-tip pens, quill pens, erasable pens, etc. By using this approach, one could list the keyword "point" once, but associate it with each of the nodes where that keyword appears by using the unique identifier for each node where the term appears.

In this manner the keywords are obtained from the collection of available descriptions found in the particular application in which the invention will be used. In addition, each particular node where the keyword appears is associated with the keyword. Thus, with respect to the pen application above, the keyword "point" might appear in nodes 2, 3, 6, 7, 13 and 15. Similarly, the keyword "erasable" might appear in nodes 3, 4, 5, 6 and 22. An index, as described more fully below, associating these keywords with the nodes containing them is then created, for example:

point: 2, 3, 6, 7, 13,15
erasable: 3, 4, 5, 22

By making use of these associations the "tree" can be negotiated by allowing presentation of relevant verbal descriptions for the nodes associated with a term, irrespective of where in the

hierarchy they are, thereby causing a "jump" to a particular node without necessarily traversing the tree in the rigid hierarchical manner.

Various examples will now be presented to illustrate certain concepts related to the invention. It should be understood that while these examples are presented in the context of things and likely experiences of ordinary people, the same approach can be applied to other forms of transaction processing including navigating through hierarchically nested data files in a computer system, pattern analysis or image processing, etc. the term "transaction" as used herein relating to traversal through a hierarchy to a goal, not mathematical calculation per se.

Moreover, the specific formats used and presented in these examples are purely for illustration purposes. It should be understood that that other techniques for interrelating data, such as hash tables, direct or indirect indexing, etc. can be substituted in a straightforward manner. Thus, for example, the relationship between the word and a node could be configured such that the location of the word in a list as the "n-th" item could be used as an index into another list containing the nodes correlated to the list. A similar approach could be used for the thesaurus, the important aspect relative to the invention being the relationship among certain words and the node(s) in which they occur and, where applicable, the relationship between certain words and "synonyms" for those words, not the data structure or its form or format whereby that information is kept or maintained.

Example 1

Example 1 illustrates, in simplified form, how an index is used to jump among nodes with reference to FIG. 2. In this example, the hierarchical tree 200 represents a portion of a more

complex tree specifically involving possible decision relating to fruit and a decision between two specific types of fruits, an apple and an orange.

In prior art hierarchical trees, navigation of this graph 200 would necessarily involve going through the "fruit" node 202 in order to reach the "apple" 204 or "orange" 206 nodes. As a result, assuming this simple tree was part of a larger tree for an on-line supermarket that prompted the user for what they wanted to purchase, the exchange would be both rigid and time consuming. For example, in response to a prompt "What do you want to purchase?" if the response was anything other than "fruit" traversal to the "fruit" node 202 could not occur. At the point in the tree that would lead to the "fruit" node 202, neither apple nor orange would be an acceptable response.

In accordance with the invention, assuming the only relevant keywords for that portion of the tree were "fruit", "apple" and "orange", an inverted index would be created that includes an association of "Fruit" with the top node 202, "Apple" with the bottom left node 204, and "Orange" with the bottom right node 206. As shown above, that association can be created using node identifiers, in this example, the node identifiers 1A01, 1A02 and 1A03 are arbitrarily assigned and used. Thus, the information can be stored in a file, for example, as follows:

Fruit, 1A01
Apple, 1A02
Orange, 1A03

Accordingly, to navigate the system 200, when a response to a verbal description is provided by a user, possible keywords are identified in the response and used to search the index and identify any node to which the response may be directed, irrespective of the hierarchy. Thus, a user response of "an orange" to a verbal description located above the "fruit" node 202 in

the hierarchy, for example, "What would you like to buy today?" would cause the system to identify "orange" as a key word from the response, search the index, and directly identify node 1A03 (206) as the node whose verbal description should be presented next, thereby avoiding the need to traverse intervening nodes, for example, through the "fruit" node (202) 1A01, at all. This illustrates an example of a simple jump according to the invention.

Example 2

Having illustrated a simple "node jump" a more complex (and likely) scenario can be shown. In this example, the Example 1 graph of FIG. 2 applies, but relevant portion of the index is as follows:

Fruit, 1A01
Apple, 1A02, 2F09
Orange, 1A03

As a result, there are two nodes relevant to the keyword "apple" one being the node 204 in the portion of the graph shown in FIG. 2 and one in the node uniquely identified as 2F09 located somewhere else in the hierarchy (not shown).

In this example, a user response containing the keyword "apple" would identify nodes with identifiers 1A02 and 2F09. In this case, and unlike the prior art, the verbal descriptions from both nodes would be presented to the user, likely in alternative fashion. Thus, if the user did not want an apple, they wanted apple cider, node 2F09 might be more appropriate because it is part of the "drinks" portion of the overall hierarchy.

Thus, presenting the user with the verbal description from both nodes would likely result in a jump to the portion of the graph nearer to node 2F09 since it is closer to the user's goal thereby speeding up the process and avoiding potentially confusing or frustrating the user.

Example 3

While the verbal descriptions associated with various nodes will generally be chosen to accurately represent the node, in accordance with certain variants of the invention, it is possible to create a situation where a user response takes them away from their ultimate desired goal. Nevertheless, by using the teachings of the present invention, the user can often still be brought to their goal quicker than possible with the prior art because the user need not rigidly trace through the hierarchy. This is accomplished by virtue of the "grouping" aspect inherent in some implementations of the invention.

This example illustrates the "grouping" aspect using a simplified graph 300 representing a portion of an airline reservation system as shown in FIG. 3.

In particular, the graph of FIG.3 can be thought of as part of a very simple interactive voice response ("IVR") system.

As described above, each node is uniquely identified, for example, by the numbers 1 through 7 and the identified terms "Reservation", "Domestic", "International", "Business Class", "Economy Class" are deemed the relevant keywords. Note, there is no requirement for a the "keyword" to be a single word, in some implementations, keywords could be single words, phrases of two or more words, or even some other form of information like a specific data pattern.

Again, an inverted index is created as described above associating those keywords with the nodes, in this case:

Reservation, 1
Domestic, 2
International, 3

Business Class, 4, 6
Economy Class, 5, 7

Assuming that the top node is assigned the number 1, its two child nodes (Domestic and International) are assigned the numbers 2 and 3, and the grandchild nodes (i.e. at the lowest level in the hierarchy) have been assigned numbers 4, 5, 6, and 7 taken from left to right each node can be uniquely located. Note that the last two entries in the inverted index are each associated with two nodes, 4 and 6 in the first case, and 5 and 7 in the second.

Using the above, the concept of grouping of nodes from different parts of the graph (i.e. nodes that are not siblings or nodes that do not have a common parent) can be explained.

Presume that the response to a verbal description presented as an initial query of "What do you want to do?" was "Make a business class reservation." In this case there are two keywords present, "reservation" and "business class".

Depending upon the particular implementation, as noted previously, the verbal descriptions associated with each identified node could be presented together or in sequence. Alternatively, and as is the case here, a set of rules can be established, for example, such that if an identified node is a sub-node of another identified node, only the verbal description of the sub-node(s) is provided because of inherent redundancy. Thus, since both "business class" nodes 310, 314 are sub-nodes of the "reservation" node 302, the verbal description associated with the "reservations" node can be suppressed if it can be determined that business class necessarily implies reservations.

In this example, a search of the inverted index would identify nodes 4 and 6 (310, 314) from different parts of the tree are associated with the keywords in the query, and thus the

system, in presenting the verbal descriptions from each, in effect, alters the tree structure and groups these nodes in the result. Thus, the combination of result nodes presented depends upon the user query or response, not that predetermined by the graph structure itself.

Of course, the goal would still not be reached because of the ambiguity caused by "Business Class" being under both "Domestic" and "International". However, that ambiguity can be handled by suitable wording of the following verbal descriptions and whether they are combined or provided sequentially or by other nodes.

Example 4

A persistent and further drawback present in the prior art is the inability to operate if any term other than the specific allowed terms are provided. Thus, in an IVR of the prior art, providing anything other than the recognized term(s) will likely result in meaningless repeat of the same inquiry by the IVR or an error.

Advantageously, the teachings of the present invention allow for construction of a more flexible system than available in the prior art. Specifically, we can incorporate a thesaurus to accommodate synonyms for the keywords.

Example 4 illustrates the addition of a simple thesaurus as an aspect of a system so that a synonym of a keyword may also be used by the system to jump to the desired nodes in the graph. Example 4 is discussed with reference to a portion 400 of an interactive television program listing system as shown in FIG. 4.

Such a system implementing the invention will allow a user to speak to or interact with a device to look for programs of his choice by time slot, genre, favorite actor or actress, etc.

This example, as with the other examples above, use an inverted index, in this case one where each node 402, 404, 406 is uniquely identified by a string of six characters, the portion of which corresponding to FIG. 4 is shown as follows.

Programs; acgyct
Sitcoms; ifgnxh
Films; vnymos

Since a common synonym for "Films" is "Movies" a thesaurus can be created associating the two. Depending upon the particular implementation, thesaurus terms to be equated to the keywords can be taken from a standard thesaurus or can be custom created for the particular application. In addition, the equating of terms can be done in any of a myriad of different ways, the exact implementation details of which however re irrelevant to the invention, but a few representative examples of which however are contained herein for purposes of illustration.

In one example case, the equating can be done on a purely word basis. For example, a file can be constructed such that one or more single word synonyms are directly associated with an index word, for example as follows:

Movies, Flicks – Films

Alternatively, the synonyms can be equated with the node identifier(s) corresponding to the index term, for example as follows:

Movies, Flicks – vnymos

In the former case, the system would still have to search the index after the thesaurus has provided the proper index term(s). In the latter case, the thesaurus provides a direct link to the respective node(s) so that re-searching is not required.

In the system of Example 4, a user who provides the input "Movies" would cause the processing to occur as follows.

The system would search the inverted index of keywords and fail to locate "Movies" as a keyword. As a result, it would search the thesaurus and find that the word "Movies" is a synonym that can be correlated with a keyword. At this point, depending upon the particular thesaurus, it would either return to the inverted index and search using the synonym keyword "Films" and return the result as the node 406 identified by "vnymos", or go directly to the node 406 identified by "vnymos" based upon the thesaurus entry.

Of course, it is possible (and likely) that in actual usage a synonym will be associated with more than one keyword. For example, "Comedies" may be associated with both the keywords "Sitcoms" and "Films", resulting in, for example, the following entry in a thesaurus:

Comedies – Sitcoms, Films

In this case, a search for "Comedies" would result in the system identifying that the synonym was associated with nodes 404, 406 for both "Sitcoms" and "Films", and it would return both terms or node identifiers corresponding to the two keywords as the result.

Example 5

Advantageously, the thesaurus concept can be extended further so that an initially unknown word (i.e. a word that is neither a keyword nor a thesaurus word) can be learned by the system and added to a thesaurus for future use.

This example is described with reference to FIG. 5 which is a portion 500 of a larger system graph as part of a very simple "geographic information system" found in some automobiles, kiosks and elsewhere today. Such a system enables a user to, among other things,

identify and get information about different locations in an environment. For example, information about particular types of restaurants in an area.

In this example, the inverted index for the portion 500 shown in FIG. 5 could look as follows:

Restaurants, 1
Pizza, 2
Burgers, 3
Chinese, 4

A user issues the following query to the system "fast food" in order to find a quick meal.

The system's search of both the index and thesaurus would result in the "term", in this case a phrase, not being found in either. In this case, it is an unknown phrase, and the system has to learn the "meaning" of the term.

To do this, the system first offers the verbal description from the top level node(s) 502 to the user – in this example, just "Restaurants". The user presumably provides a positive response. (Of course, in a real system, it is possible and likely there are more top level nodes than just one. In that case, the user would be offered two or more of these nodes, and would have to select "Restaurants" to match his intended request.)

Continuing on, once the user has responded affirmatively, the system moves down the tree and offers the verbal description from each of the child nodes: "Pizza" (504), "Burgers" (506), and "Chinese" (508). Presuming that the user picks "Pizza", the transaction interaction would look something like this:

User: Fast food

System: Restaurants?

User: Yes

System: Pizza, Burgers, or Chinese?

User: Pizza

At this point, the system has "learned" for the time being that it can equate "fast food" with "pizza" and can add "fast food" as a synonym to "pizza" in the thesaurus.

This user, who first used the unknown term "fast food", had to trace a path down the tree. However, now the system is able to associate "pizza" with "fast food" and create or add a thesaurus entry to reflect this association, for example as follows:

Fast food – Pizza

Thus, the system has learned a meaning of the initially unknown term "fast food" and has added it to the thesaurus for future use.

As a result, a subsequent uses of the same term "fast food" will enable the system to jump directly to the "pizza" node 504.

Example 6

This example illustrates how additional meanings for an existing thesaurus term or phrase can be learned by the system for future use, whether the existing thesaurus term or phrase was an original thesaurus term or one previously learned with continuing reference to FIG. 5.

At this point, the inverted index is unchanged as:

Restaurants, 1
Pizza, 2
Burgers, 3
Chinese, 4

Additionally, presume the following entry now exists in the thesaurus.

Fast food – Pizza

Suppose a new user now issues the query "fast food" as above, but with "Burgers" rather than "Pizza" in mind.

Based upon the thesaurus, the system would go directly to the "Pizza" node. However, the user will reject "Pizza", having "burgers" in mind. By rejecting the "Pizza" node 504 description, the user indicates that the "Pizza" node 504 is not of interest. The system is therefore configured with a further set of rules, in this case one in which the system goes up in the hierarchy to a higher node, the top node 502 in this portion of the example, and provides the verbal descriptions for the other nodes 502, 504, 506, 508 so as to cause a tracing down the tree. This can be illustrated by the following "dialog":

User: Fast food

System: Pizza?

User: No

System: Restaurants?

User: Yes

System: Pizza, Burgers, or Chinese?

User: Burgers

This time, although this user has had to trace through at least a portion of the path from a higher-level node 502 of the tree 500, the system has learned yet another meaning for "fast food". It now adds this meaning to the earlier entry in the thesaurus, for example as:

Fast food – Pizza, Burgers

It has now learned two meanings for future use. If a user were now to issue the query "Fast food", the system would respond with the verbal descriptions from the nodes 504, 506 corresponding to both Pizza and Burgers.

Thus, the system can keep learning new meanings of terms based on the intended meanings of users "deduced" from the interactions between users and the system.

Of course, the nature and extent to which the system will incorporate synonyms and/or keywords in a continual learning process will not only depend upon its construction and rules, but also on the quality of the original thesaurus and the quality of the initial inverted index. In addition, where in the tree the system jumps if the user rejects the initial meaning(s) offered by the system can be handled different ways in different implementations.

For example, the system can always jump to fixed ancestor(s) (either the top node or a parent or some ancestor(s) at an intermediate point) or a fixed level (e.g. halfway from the top). This approach has the advantage of being simple to implement, but it has the problem of inflexibility because it may be relatively efficient for certain graphs and associated verbal descriptions, but not for all. For example, if two or more nodes' verbal descriptions are offered and rejected, the relevant node selected would have to be common ancestor(s) of the offered nodes. In other words, with reference to Example 6 which is part of a larger tree, going up to the "Restaurants" node 502 would mean going to the parent of the "Pizza" node 504 rather than all the way to the top in the larger tree containing the portion 500 shown.

A more flexible alternative uses the information recorded in the thesaurus to find every synonym for "pizza" in the thesaurus and collect all the other keywords associated with those synonyms. Then the system would search the inverted index to identify all the nodes associated

with these other associated keywords and identify the most common ancestor of all of those nodes and go to it. By using the information in the thesaurus in this way the system makes use of known properties of the one meaning of "fast food", which is "Pizza", to construct an intelligent hypothesis about where the other meanings of "fast food" might lie in the graph. This allows the user to reach another meaning with the least effort and allows the system thereby to learn what the new meaning of "fast food" is more efficiently.

Example 7

Of course, just as it may be desirable to create implementations to add meanings to the thesaurus, it may be equally or more desirable to cause an existing meaning for a thesaurus word to be dropped, for example, due to relative lack of use. This process is described with continuing reference to FIG. 5 and the associated inverted index, particularly with respect to the thesaurus entry resulting from the most recent example.

Fast food – Pizza, Burgers

In this example, presume that there have been several uses of the query "fast food" and that the user(s) issuing these queries have almost always selected "Burgers" and almost never "Pizza".

In accordance with another implementation of the invention, the system is constructed to track the frequency of use of a particular term in the thesaurus. Depending upon the particular implementation, the tracking can be done for all entries in the thesaurus, for only those added as part of the "learning" process, or for some specified combination thereof.

In addition, some specified criterion is used to determine when, and which terms, if any, should be removed from the thesaurus. Depending upon the particular implementation the

criterion can be based upon usage relative to time, usage of a particular term relative to some other term(s), term usage relative to overall thesaurus usage, or simply elimination of all added terms not used since the last purge.

Thus, presuming that the system has kept track of the frequency of use of different meanings of "fast food", and that "Pizza" does not meet the criterion for a sufficiently high frequency, the meaning "Pizza" can be dropped as a synonym for "Fast food" and the entry (after purging) would look as follows:

Fast food – Burgers

Thus, a further enhanced implementation can be constructed so the system is dynamically updating the thesaurus, either adding meanings or dropping meanings for existing and/or initially unknown words.

Example 8

A further advantage to the invention is that, in some implementations, it can be configured so that, when there are multiple relevant nodes to be presented, an associated ranking can be used to determine the type, method or order of presentation. For example, the ranking can be based upon the frequency of use of particular nodes, which is tracked in these implementations, so that the most frequently selected or used nodes are presented first, more prominently, or in a particular manner.

For example, this can be illustrated by continuing from Example 7, where the thesaurus entry was as follows:

Fast food – Pizza, Burgers

Under the assumption that the system has been tracking the frequency of usage of the "Pizza" node and the "Burgers" node and each has been accessed an identical number of times. When a user enters the query "Fast food", as above, the system presents the user with both the "Pizza" node 504 and the "Burgers" node 506, but because it tracks usage and the usage is the same, it presents them in the order they are listed, i.e. "Pizza" and then "Burgers". However, at this point, the user's selection will cause one entry to have a greater frequency of usage relative to the other entry, for example a selection of "Burgers" will make it have a higher frequency of usage and, accordingly, a higher ranking for the next instance of use.

Thus, the next time the system will be presenting both the "Pizza" and "Burgers" nodes to a user, the "Burgers" node 506 will have the higher frequency of usage and, accordingly, will be presented first, or more prominently, or in some other specified manner because of its ranking. If the frequency reverses with use so that the "Pizza" node 504 outranks "Burgers" node 506, then the "Pizza" node 504 will supplant the "Burgers" node 506.

Example 9

A further variant of Example 8 allows the node rankings to be used to prune the nodes themselves. In this variant, a criterion can be specified, typically zero usage over a long specified period of time, that is used to remove an entire node. This is advantageously made possible because of the system's ability to "jump" among nodes. Thus, it may occur that a node within the tree is never accessed, but a child node of that node is. In some variants therefore, when this state exists for a sufficiently long period of time, the system is constructed to delete that node. It should be understood that, if handled properly, this process will not even affect the "learning" process because, even if no user action ever directly causes the node to be presented,

if the learning process causes the node to be presented the node's access frequency will be non-zero and it will not be "pruned".

In addition, by tracking access frequency on a node basis, a qualitative evaluation of the hierarchical system can be made and visualized. This makes it possible to review the overall hierarchy after some period of time and periodically optimize it based upon the result instead of relying purely upon the dynamic optimization that inherently and naturally flows from use of the teachings of the invention.

Having now described various component aspects of different variants implementing the invention, by way of the above examples, it should be understood that the "jumps" can occur from any node to any node, i.e. vertically and/or laterally and to another node that is higher, lower or on the same "level" as the node from which the jump is made. All manner of vertical and lateral jumps from multiple nodes to multiple nodes are possible.

In addition, it should be understood that in some applications (like document retrieval systems) the verbal description from the identified node may be the one issued whereas, in others (like an IVR system), the verbal descriptions for the children of the identified nodes may be what is presented. Nevertheless, in both cases, the process as described above by way of example will be the same or directly analogous.

Having described the various aspects individually a more commercially suitable example, employing a combination of the above examples, can now be presented with reference to FIG. 6 which illustrates a simplified example of an "interactive voice response unit" (IVR) hierarchy 600 that might be used in the airline industry. Of course, a real menu tree used in an IVR may have any number of nodes from several, up to a thousand, or more. For example, a tree with 4

branches from each node and which has 5 levels uniformly would have 1365 nodes. As shown in FIG. 6, the tree 600 is a hierarchical tree and consists of the following nodes and branches:

Initial start (node a0) 602

domestic flight arrival information (node a1) 604

domestic reservations (node a2) 606

international flight arrival information (node a3) 608

international reservations (node a4) 610

The node 604 identified by a1 is a service node with pre-recorded information.

The node 606 has two child node a 2, first/business class (node a5) and economy (node a6).

The node 608 identified by a3 is service node with pre-recorded information.

The node identified as a4 has three child nodes identified as first class (node a7), business class (node a8), and economy (node a9).

The nodes 612, 614, 616, 618, 620 identified as a5, a6, a7, a8, a9 are all service nodes (i.e. terminal nodes) where a respective customer service representative will interact with the caller.

Of course, a real system may also have a choice at the top level or at each level for a live operator and may even have a choice to go back to the previous menu.

Even for such a simple example, in a traditional interactive voice response system, the caller would have to listen to several choices and then traverse a path down to a service node. Someone interested in business class reservations on a domestic flight would have to traverse the path (a0, a2, a5) for example. This involves listening to multiple choices at each level of the tree (e.g. first a prompt at a0, then four prompts offering a1, a2, a3, and a4 at the next level, at which the caller would choose a2, and finally two prompts offering a5 and a6, at which level the caller

would choose a5 and then wait for the operator) and then making a choice by pressing an appropriate number on the telephone dial pad or alternatively saying the appropriate number. In certain cases, he may make a mistake: he may choose international reservations when he is interested in domestic reservations or something similar (simply by pressing the wrong number on his touch-tone telephone or saying the wrong number). If he does, then he has no choice but to disconnect the phone and redial the number (or if the system has a backtracking option, then he can backtrack, but even here he has wasted valuable time).

In contrast, in accordance with a system implementing the invention, the caller would be able to say what he was looking for (e.g. "I want to make a domestic business class reservation") and the system would identify and respond with the appropriate node 612 (e.g. a5 in this case or the relevant customer service representative directly). In other words, it would enable the caller to skip to the correct node(s) without having to trace through the entire path. If the user makes a mistake, he could ask for something different wherever he finds himself in the tree, and skip laterally or vertically to his preferred choice.

The system implementing the invention can further include an option that the entire transaction (e.g. the making of the reservation) would be carried out through natural language interactions with the system without the intervention of a human customer service representative. In other words, all the details of his domestic reservation are obtained by the system and the system updates a database accordingly and issues whatever commands are required (e.g. the mailing of a ticket) to be carried out by some human representative later.

While it is true that some more advanced interactive voice response systems available today allow for natural language interactions, they are highly constrained natural language

interactions with relatively little or no intervention by a human operator. However, unlike with systems using the invention, those systems still require direct path traversal through the hierarchy (i.e. jumping to non-connected nodes is not contemplated or possible, let alone allowed). Moreover, such systems still typically use a limited list of keywords, which the caller is required to use to correctly traverse to the next connected node.

In contrast, variants of a system implemented in accordance with the invention can incorporate an automatically generated or updated thesaurus, which greatly expands the range of words or terms a caller can use. In these variants, the user is not restricted to parroting the highly constrained script as required by other interactive voice response systems, nor is the user limited to traversal to a connected node. In these more complex implementations of the invention, a system can be constructed that is able to learn new words or terms that it may not have understood the first time. For example, if a user asks for "coach class" and the system does not have the word "coach" or the phrase "coach class" in its keyword list or in its current thesaurus, then on this first occasion, it offers the user a traditional path down the conventional tree. But it tracks what the user did, what node of the tree the user went to, and on this basis, it learns a new response to "coach class". The next time a caller (either the same person or a different person) uses the words "coach class" the system does not offer the traditional path as it did the first time, but instead it offers a new set of nodes based on what it learned the first time. Thus, in such implementations, the thesaurus is a dynamically changing entity, continually updating itself by learning new words and terms and learning new "meanings" (i.e. new actions or responses) for existing terms.

Implementations according to the invention can also allow novel groupings of nodes to be presented to the caller based on his query. If he asks for "economy class" without specifying whether he wants an international or domestic reservation, then the system would offer him the nodes a6 and a9 (appropriately phrased in natural language), and allow him to further choose whether he wants international or domestic reservations, something current systems do not offer. In other words, the system can pick out the relevant responses from different branches of the tree and pool them together and offer them to the caller.

This functioning of the system, by which it is able to skip around laterally or vertically in the tree, is enabled by the associating of natural language (i.e. human language) verbal descriptions with each node, and then using these as an initial basis for the navigation, augmented, in some variants, by a dynamically changing thesaurus that greatly expands its range and comprehension.

Thus, based upon a conceptual understanding of the above examples, further details of the process will now be presented.

The flowcharts of FIGS. 7 through 14 are illustrative of a functional example of the general method of a more complex variant the invention as would be implemented in software according to the flowcharts in this case for a newspaper subscription application. It should be understood that particular details are provided in the description below merely for completeness or because they are necessary or helpful for forming an understanding of the particular implementation. They are not to be considered essential for implementing the invention. Similarly, details unrelated to or unnecessary for understanding the invention have been omitted to avoid confusion.

An example implementation is described and contains two programs, a preparatory program, illustrated in FIGS. 7-10 and a transaction or query processing program, illustrated in FIGS. 11-14. In addition, a particular software implementation fairly corresponding to the flowcharts of FIGS. 7 - 14 appears in the Appendix A that follows. The program contained therein, is written in the "C" programming language for execution on any personal computer having a processor, memory , input-output, etc. capabilities to run the particular application in its intended environment.

Broadly, the first program process of FIGS. 7-10 constructs an inverted index and an application-specific thesaurus and the second program process of FIGS. 11-14 uses those constructs in a transaction processing system to interact with a user.

In the preparatory program of FIGS. 7A, 7B and 8-10, the shorthand names of files that the program uses and the contents of the corresponding files are as follows. Notably, both the process parts shown in FIG. 7A and 7B as well as the process part shown in FIG. 8 are indicated as start points. This is because they are each independent of each other in that any of the three could start before any other or two or more could be run concurrently. Thus, it should not be presumed that they are mutually exclusive or any one is per se required for the invention. Moreover, it should be understood that any one or more could have been undertaken at a different time, by a different entity, or for a different application. Whether one or more of the portions shown in FIG. 7A, FIG. 7B or FIG. 8 are the starting points, the starting point for actual operational processing will be the same.

The file named 'p' contains a list of prompts or verbal descriptions in a hierarchical relationship (i.e. they can be visualized or arranged in a tree-type graph).

The file named 'w' contains documents that are related to the prompts or verbal descriptions in 'p'. For example, 'w' could contain a training manual for customer service personnel or a website document that is likely to contain material that is related to the queries customers may have. This file is used to create a thesaurus.

The file named 'f' contains forms that are used to elicit relevant information from customers. They have fields like 'name', for example, which would be used by the system to ask and record a caller's name.

The file named 'x' contains an index associating the forms in 'f' with terminal prompts or descriptions in 'p'. Once a terminal prompt is reached in the process, the corresponding form from the file 'x' is activated, and the system proceeds to elicit information from the user.

The file named 's' contains a list of application-specific stop words, many of which are high-occurrence and/or generally uninformative words like 'a', 'an', 'the' or 'from' or words with a high-occurrence in for the particular application such that they have little meaning, for example, 'fly' in an airline reservation system, 'street' in a navigation system, 'file' in a computer search tool. These are eliminated from 'p' and 'w' and 'f' before processing, because they don't carry any useful information for the application.

The file 't.cfg' contains the thesaurus and inverted index that will be constructed by the program. Of course, in alternative variants, the thesaurus could be a separate file from the inverted index file and either or both could be made up of multiple files.

The file 'l.cfg' is a file that is used to store newly learned words. As with the t.cfg' file, the 'l.cfg' file need not be a separate file, it could be part of 't.cfg', or part of a separate thesaurus and/or inverted index file. Similarly, the 'l.cfg' file could be made up of several files.

With reference to FIGS. 7A, 7B and 8 through 10, the processes as carried out by the first program are as follows. It bears noting that, although the process and its components are presented by way of example in a particular order, unless a specific process component is expressly stated to necessarily have to occur at a particular time or after some other particular process component, or two process components must necessarily occur in sequence because one relies upon completion of the other before it can start, no order should be implied or considered required since the order in different implementations may be different and may vary based upon the particular programmer, programming language and/or computer involved.

The files p, w, f, x, and s are each read and processed as follows. It should be understood that the order of processing of file 'p' relative to file 'f' or their respective sub-processing components, although shown sequentially, could be done in a myriad of ways including doing each of the reading extracting and storing concurrently or as a common operation (i.e. reading for both is done before extracting for both, etc.).

Specifically, keywords are extracted from p ___ and from f ___. These are initially just all the words or terms contained in the prompts in p. The keywords are stored, for example, in a temporary file.

Similarly, thesaurus words are extracted from w. These are initially just all the words or terms in w. They are also stored, for example, in a temporary file.

Stop words are loaded from s (902) and stop words and duplicate words are eliminated from keywords and thesaurus words stored in the temporary files.

The thesaurus is constructed in accordance with FIGS. 9 and 10 described in overview as follows:

a. Increment the file of thesaurus words with keywords from p and f remaining after elimination of stop words.

b. Create a matrix of thesaurus words as row words (or words listed along the rows of the matrix) against keywords as column words (or words listed along the columns of the matrix).

c. Count the number of co-occurrences of each row word with each column word of the matrix in the documents contained in w and fill in that number in the corresponding matrix cell. (For example, a co-occurrence of a pair of words may be defined as that pair occurring in the same paragraph. If w is made up of a hundred paragraphs, then take each pair of row word and column word and count the number of times this pair occurs within the space of each of the hundred paragraphs in w. For each pair, the pair may co-occur zero or more times in a paragraph and add up the number of co-occurrences in all the paragraphs in w.)

This process yields a matrix filled with nonnegative integers in each cell. It is then possible to consider each row of numbers as a vector associated with the corresponding row word. When viewed geometrically, these vectors, one for each row word, form angles with each other in a multi-dimensional space. As a result, we can calculate the cosine of each such angle by computing scalar products for the angles. Thus, we compute the cosines of the angles formed by the vectors associated with each pair of row words.

The cosine values for all pairs of row words and column words are calculated and stored, for example, in a new matrix.

For each row word, the top 'n' cosine values are identified as are the corresponding keywords. For example, in an airline system context, if there are two row words 'coach' and 'economy', where 'economy' is also a keyword (originally from p and/or f), and if the cosine value of this pair or words is among the top 'n' cosines for the word 'coach', then 'economy' is identified as a synonymous keyword for coach.

A new file can then be created, formatted for example, by listing thesaurus words on the left (e.g. coach), and against each thesaurus word, its associated keywords (e.g. economy). This is referred to as an inverted index (i.e. the thesaurus) of row words and their keyword synonyms. Essentially, this file will now contain words like 'coach' coupled with its particular alternative meanings, one of which may be 'economy'. The user interactive transaction processing program, the second program, will later use this thesaurus file when a caller uses a word like 'coach' in his query to determine the relevant keywords (like 'economy'). This will enable the program to find the relevant prompt with which to respond to the user.

Optionally, to provide the system with a set of prompts or verbal descriptions with which to respond to a user, another inverted index is created using the files p, f, and x. This index will contain a list of keywords from p and/or f associated with the prompts in which they occur. Thus, when a user uses a synonym like 'coach' in a query, the second program will look up the thesaurus, find the keywords corresponding to it (e.g. 'economy'), and then look up the inverted index to find the prompts corresponding to 'economy' and other corresponding keywords.

Once both the inverted index and thesaurus files have been created, the file t.cfg can be created from them for use by the second program.

One example of the program flow for a fairly generic transaction processing program implementing one variant of the invention is illustrated in the flowcharts of FIGS. 11 through 14. This example is configured to incorporate a collection of several of previously described simple aspects. To demonstrate the functions of this program and how this program operates, for context we use an example interaction that a calling customer might have with this example system.

Following the example is the Appendix contains that program code essentially implementing a variant of the invention largely corresponding to that of FIGS. 7 through 14.

The particular example we use for purposes of illustration is for an automated telephone system for a newspaper, like the New York Times. For simplicity, every item in the flowchart is not traced through since, an understanding of the process with respect to one path will be sufficient to allow an understanding of the other paths.

The example begins with "I want to subscribe" uttered by the caller to the system. We will assume that the first three words of the query (i.e. "I", "want", and "to") are stop words and the last word (i.e. "subscribe") is neither a keyword nor a thesaurus word.

The process as carried out by the second program are as follows:

The files t.cfg, l.cfg, f, x, and s are read (1102).

The keywords, thesaurus words, prompts from t.cfg. are loaded (1104), as are the learned words from l.cfg. Initially, l.cfg will be empty as the program has not yet learned any new words. The forms and index of forms against prompts from f and x respectively are loaded, as are stop words from s.

The program opens the interaction with a greeting (1106) and an elicitation of the first query from the caller (1108). This might be: "Thank you for calling the New Herald. How may we help you?"

The caller then utters his or her statement: "I want to subscribe".

The stop words in the statement are first eliminated, leaving behind just the word "subscribe".

The statement is then processed in the following way:

The keywords and the thesaurus words remaining in the query are identified (1202, 1204) by comparing with the list in t.cfg and l.cfg. As we have assumed that "subscribe" is neither, we have none.

The prompts that best match the identified keywords and thesaurus words are selected (1206). As there are no such words identified, there are no prompts selected.

The program arrives at a decision in the flowchart: are any nodes selected? (1208). Since the answer is in the negative, the program will follow the branch and select the top level node (1218). (Note: These top level prompts are the ones at the top level of the menu tree.) This completes the prompt selection process. The process then proceeds to the second part of the query process.

The process proceeds with another decision: has a single leaf prompt been selected? (1210). Since the top level prompts are selected (of which there are more than one and also none is a leaf prompt), a negative answer is the result.

These prompts or verbal descriptions are issued to the user (caller) and elicit another response. Assume that the offered verbal descriptions are as follows:

System: Are you calling about subscriptions?

System: Is there a problem with your paper or delivery?

System: Would you like information about the New York Times website?

System: Are you calling about advertisements?

System: Are you calling about something else?

Assume further that the caller responds as follows:

User: I am calling about subscriptions.

As a result, the program returns to selecting verbal descriptions by identifying the keywords and the thesaurus words remaining in the query by comparing with the list in t.cfg and l.cfg (1202, 1204). "Subscriptions" is now synonymous with a keyword and it is identified.

The program will again select verbal description(s) that best match the identified keywords and thesaurus words (1206).

For example, assume these are:

System: Would you like to order a subscription?

System: Would you like to give a gift subscription?

System: Would you like to change your address or change any other information?

The program then arrives at a decision branch (1208) in the flowchart: are any nodes selected? Since the answer is affirmative, it follows that branch and exits the prompt selection process and returns to the query process.

This begins with another decision box: is a single leaf node selected? (1210). The answer is no, since three prompts have been selected.

Next, these verbal descriptions are issued to the caller and the system will await his response (1220). We assume the caller responds as follows:

User: I want to order a subscription

The program will again return through a loop to the prompt selection process (1202, 1204, 1206) where the program will identify the keywords and the thesaurus words remaining in the query by comparing with the list in t.cfg and l.cfg. "Order" and "subscription" are now identified.

Verbal descriptions are selected that best match the identified keywords and thesaurus words. Now assume this is just the prompt "Would you like to order a subscription?" from the three descriptions above.

The program will then arrive at a decision branch (1208) in the flowchart: are any nodes selected? Since the answer is affirmative, it follows that branch and exits the prompt selection process and returns to the query process to again arrive at a decision: has a single leaf node been selected? (1210). This time the answer is yes, a single prompt has been reached, which is also a leaf prompt, since it is at the bottom of the menu tree.

This is followed by another decision: any verbal description corresponding to the node? (1212). The program checks t.cfg and finds the answer is no.

The branch then leads to yet another decision (1214): is a form for verbal description available? The answer by checking the index x is the yes branch. This leads to the portion of the flowchart of FIG. 13.

The form is processed in the following way:

The first part is a decision: is it a response form? (1302). The answer is no.

The system then issues questions to the caller based on the form and accepts information back (1304). The questions are of the form "Please tell us your name", "Where do you live?", "Do you want an annual or half-yearly subscription?" etc. The caller provides the information to the system.

It repeats the information the caller has given the system and asks if the information is correct. Let us assume the user confirms that the information is correct.

The system then calls an external routine to store the information in a database. The routine returns another form to the system (1306) and returns in a loop to the question: is it a response form? (1302). Since the form contains questions about the payment, based on the type and period of subscription selected by the caller, the answer will be negative.

The system then issues these questions to the caller and the caller provides the required information (1304).

The system then repeats the information and gets a confirmation from the caller.

The information is passed to another routine (mentioned in the form) to update the database. This routine (1306) then returns a response form and again returns in the loop to the question: is it a response form? (1302). This time the answer is yes. The system then issues a response (1308) thanking the caller for the subscription, and exits this process returning to FIG. 11.

The system now exits the query process as well since the caller's query has been completely processed and the corresponding actions taken by the system. It now returns to the main part of the program.

The next process in the main part of the program is a question: is there an unknown word in the caller's query? (1112). The answer is yes, since the word "subscribe" in the initial query was not known to the system. This invokes the portion of the flowchart of FIG. 14.

The system therefore has to learn this previously unknown word:

The learning process begins with a decision: is the word already in l.cfg? (1402). The answer is no, since l.cfg is initially empty and the word has not been encountered before.

The corresponding "NO" branch is followed and the word is added to the list of learned words (initially empty) with keywords from the final single leaf prompt that was selected (1404).

The system then records these changes in l.cfg (1408) and returns to the main part of the program in FIG. 11. It has now learned the meaning of the initially unknown word "subscribe".

Next, the program asks the caller if he wishes to continue (1114) (i.e. are there any further queries). We assume the answer is no and the system thanks the user and exits.

Now, having described the example traversal of one path through the second program with reference to the flowchart, an example dialogue for the path traversal that has taken place is presented so the complete transaction can now be understood.

Dialogue:

Caller: I want to subscribe

System: Are you calling about subscriptions?

System: Is there a problem with your paper or delivery?

System: Would you like information about the New York Times website?

System: Are you calling about advertisements?

System: Are you calling about something else?

Caller: I am calling about subscriptions

System: Would you like to order a subscription?

System: Would you like to give a gift subscription?

System: Would you like to change your address or change any other information?

Caller: I want to order a subscription

System: Please tell us your name

Caller: Bertrand Russell

System: Where do you live?

The dialogue continues in this way with the system eliciting the required information from the caller.

Having demonstrated traversal in a system where the system was constructed to learn when an unknown word is used, what happens the second time a caller uses the same word "subscribe" in a query after it has been learned by the system can now be demonstrated. This demonstrates the power of including the optional feature of learning in the program.

In this case, the dialogue that occurs when a new caller uses the word "subscribe" following the above is now presented.

Dialogue:

Caller: I want to subscribe

System: Please tell us your name

Caller: J. L. Austin

System: Where do you live?

Thereafter, the process continues. Notably, the system has now learned the correct response to the query "I want to subscribe".

Other Variants

Having described several simple and more complex examples that make it possible to use the invention, other variants can now be presented. Examples of such optional functions that can be incorporated into other variants, individually or collectively, include:

a) creating the thesaurus by providing access to a collection of multiple documents and determining synonymy based on sufficient similarity of meaning with the keywords as measured by the frequency of co-occurrence of the keywords in the collection of documents;

b) identifying words in the user's response by recording the response for future learning;

c) parsing out of a response all non-stop word unknown terms or keywords;

d) identifying synonyms for all non-stop terms in the user's response;

e) cycling between user and system responses until the user reaches a vertex (i.e. verbal description) that enables him to carry out his goal and updating the thesaurus when the goal vertex is reached by associating the recorded previously unknown words in the user's response with the keywords that are associated with the verbal description reached by the user;

f) recording, when the goal vertex is reached, the pairs of synonyms in the user's responses and the keywords that are associated with the verbal description reached by the user;

g) removing associations between keywords and their synonyms from the thesaurus that have not been accessed more than a specified amount of times within a specified period (this can be based upon a parameter set in the system by the system's administrator or can occur as part of program maintenance or updates);

h) selecting the verbal descriptions that best meet the user's goal as indicated by the keywords and synonyms in the user's response by identifying the keywords in the user's response and/or the keywords corresponding to synonyms in the user's response and computing a degree of match between each verbal description and the identified keywords, in accordance with conventional linguistic processing techniques;

i) computing the degree of match between verbal descriptions and identified keywords by utilizing the pairs of synonyms in user's response and the keywords associated with the verbal descriptions reached by users as previously recorded;

j) responding to the user on the basis of verbal descriptions selected by presenting the user with verbal descriptions that best match the user's previous response;

k) for "best match" variants, in the event that even the best matches have a low degree of match, the best "n" verbal descriptions are presented to the user ("n" being a number representing a predetermined system parameter);

l) for "best match" variants, in the event that the best matches have a low degree of match, the user is automatically connected to a human operator, when or if a human operator is available;

m) for "best match" variants in the event that the best matches have a low degree of match, the best "n" verbal descriptions are presented to the user, along with an option of being connected to a human operator when or if a human operator is available;

n) presenting the user with those verbal descriptions that best match the user's previous response in order to elicit any information from the user that may be required to accomplish the user's goal;

o) recording information elicited from a user in a database for future use;

p) selecting multiple verticies in the graph structure that are not connected to a previously selected vertex, based upon parameters associated with nodes correlated to keywords and synonyms in a user's response;

q) selecting a vertex in the graph structure in the same row as the previously selected vertex based upon the keywords and synonyms in the user's response; and/or

r) updating the thesaurus by adding words from a user's response that are not in the thesaurus.

Finally, it is to be understood that various variants of the invention including representative embodiments have been presented to assist in understanding the invention. It should be understood that they are not to be considered limitations on the invention as defined by the claims, or limitations on equivalents to the claims. For instance, some of these variants are mutually contradictory, in that they cannot be simultaneously present in a single embodiment. Similarly, some advantages are applicable to one aspect of the invention, and inapplicable to others. Thus, no particular features or advantages should be considered dispositive in determining equivalence.

It should therefore be understood that the above description is only representative of illustrative embodiments. For the convenience of the reader, the above description has focused on a representative sample of all possible embodiments, a sample that teaches the principles of the invention. The description has not attempted to exhaustively enumerate all possible combinations or variations, for example, those arising out of the use of particular hardware or software, or the vast number of different types of applications in which the invention can be used. That alternate embodiments may not have been presented for a specific portion of the invention, or that further undescribed alternate embodiments may be available for a portion of the invention, is not to be considered a disclaimer of those alternate embodiments. One of ordinary skill will appreciate that many of those undescribed embodiments incorporate the minimum essential aspects of the invention and others incorporate one or more equivalent aspects.

APPENDIX A

FILE IDENTIFICATION

Main Source Files
main.c, process.c, arraylib.c, stemlib.c, dialog.c, interactive.c, formlib.c
Header Files
globalvar.h, process.h, arraylib.h, forms.h
Make Files
Makefile
Parameter Files
t.ini, d.ini
Data Files
p, w, s, f, x, a
Configuration Files
t.cfg, l.cfg
Shell Script Files
acct_info, add_acct, chg_acct, get_pymt, updt_pymt, susp_deli, updt_acct, prefer

MAIN SOURCE CODE (in C)
main.c: Main Program to process p and w to create the thesaurus

```
/*****************************************************************

   SOURCE CODE DOCUMENTATION

*****************************************************************/
#include <stdio.h>
#include <string.h>
#include "process.h"
#include "arraylib.h"
#include "forms.h"

int numColumn = 0,  numRow = 0 , numIndex = 0, numMenu;
int topValues = 5 ;
char **rowTerms, **columnTerms, **prompts, **stopWords;
double **matrix, **cosine;
float phoneThreshold = 0.02, webThreshold = 0.0006;
int **indexList, **menuList, **thesaurus, **promptKeys;
int numStopWord = 0;
int numForms, numPF;
struct form **formlist;
char ***Fprompts, *wdoc, *pdoc, *sdoc, *fdoc, *xdoc, *cfg;

main(int argc, char *argv[]) {

int i, j;

/*****************************************************************

   PRINT THESAURUS PROGRAM INFO

*****************************************************************/

 /* if (argc != 5) {
   printf("Usage Instructions: t p w f x\n");
   printf("Parses w for matrix row terms and p for matrix column terms.\n");
   printf("*** Exiting, goodbye.\n");
   exit(1);
 } */

 if (argc != 2) {
   printf("Usage Instructions: t <ini-file>\n");
```

```
    printf("*** Exiting, goodbye.\n");
    exit(1);
  }
/*********************************************************************

   OPEN INPUT FILES

*********************************************************************/
  readini(argv[1]);
  loadStopWords(sdoc);
  numPF = loadFormsList(xdoc);
  numForms = loadForms(fdoc);

/*********************************************************************

   PREPARATION FOR PHONEDOC PARSING

*********************************************************************/
  printf("\nReading files ....\n");
  numColumn = processFile(pdoc, &columnTerms, phoneThreshold);
  // printf("The document contains %d relevant terms.\n\n", numColumn);

  // This routine will add the keywords from the Forms into ColumnTerms.
  numColumn = addFormKeys(&columnTerms, numColumn);

  // printf("The document contains %d relevant terms.\n\n", numColumn);

/*********************************************************************

   PREPARATION FOR WEBDOC PARSING

*********************************************************************/

  numRow = processFile(wdoc, &rowTerms, webThreshold);

/*********************************************************************

   MERGE COLUMNTERMS & FINALTERMS INTO ROWTERMS

*********************************************************************/

  numRow = mergeArray(&rowTerms, columnTerms, numRow, numColumn);
  sortArray(rowTerms, numRow);
```

```
  // printf("The document contains %d relevant terms.\n\n", numRow);
/*****************************************************************

   MATRIX CONSTRUCTIONS

*****************************************************************/

  printf("loading prompts ...\n", numIndex);
  numIndex = loadPrompts(pdoc);
  printf("processing words ...\n", numIndex);
  createMatrix(wdoc);
  numRow = eraseZeroes();
  calcCosine();
  fillIndex();
  // This function will add leaf prompts to the index keywords from Forms.
  // appendIndex(argv[3]);
  createThesaurus();
  // printf("created thesaurus .\n\n", numIndex);
  printf("saving data ...\n");
  saveData(cfg);
  printf("done.\n");
}


readini(char * filenm)
{
        FILE * fp;
        char buf[80], key[80], value[80], comment[80];
        int cnt;
        if ((fp=fopen(filenm,"r"))==NULL)
        {
                perror(filenm);
                exit(1);
        }
        while (fgets(buf,79,fp)!=NULL)
        {
        sscanf(buf,"%s %s %s",key,value, comment);
        if (!strcmp(key, "pdoc"))
                        pdoc=strdup(value);
        if (!strcmp(key, "wdoc"))
                        wdoc=strdup(value);
        if (!strcmp(key, "sdoc"))
                        sdoc=strdup(value);
```

```
        if (!strcmp(key, "fdoc"))
                fdoc=strdup(value);
        if (!strcmp(key, "xdoc"))
                xdoc=strdup(value);
        if (!strcmp(key, "cfg"))
                cfg=strdup(value);
        if (!strcmp(key, "pt"))
                sscanf(buf,"%s %f %s",key,&phoneThreshold,value);
                //phoneThreshold=(float)atof(value);
        if (!strcmp(key, "wt"))
                sscanf(buf,"%s %f %s",key,&webThreshold,value);
        if (!strcmp(key, "tv"))
                topValues=atoi(value);

        }
}
```

process.c: This program contains various functions called from Main

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "globalvar.h"
#include "arraylib.h"
#include "forms.h"
#define min(x, y)  (x < y) ? x : y

int *rowcount, *colcount;

int processFile(char *filename, char ***cArray, float threshold) {
FILE * fp;
char tmpWord[50], paraFlag;
int i, numWords = 0, wordLen = 0, totWords = 0;
float *freqArray;

fp = fileOpen(filename,"r");
*cArray = NULL;
freqArray = NULL;

while((wordLen = fetchWord(fp, tmpWord)) != 0) {
        totWords++;
        if (! inArray(stopWords, tmpWord, numStopWord)) { // ignore stopwords
```

```
                stemWord(tmpWord);
                if (i = inArray(*cArray, tmpWord, numWords)) {
                        freqArray[i - 1]++;
                        }
                else {
                        addWord(cArray, tmpWord, ++numWords);
                        freqArray = (float *) realloc(freqArray, numWords * sizeof(float));
                        freqArray[numWords - 1] = 1;
                        }
                }
        }
 // printf("Totwords = %d, numWords = %d\n",totWords, numWords);
for ( i = 0; i < numWords; i++)
        {
       if ((float)(freqArray[i] / totWords) >= threshold)
          (*cArray)[i] = NULL;
        }

numWords = removeNulls((*cArray), numWords);
sortArray((*cArray), numWords);

return numWords;
}

/*****************************************************************************
Prompts Processing
*****************************************************************************/

int loadPrompts(char *filename)
{
 char buffer[256];
 int i = 0, j, len, nc;
 int level[10], tabs, m = 0;
 FILE * fp;

 for (i = 0; i < 10; i++)
        level[i] = 0;

 fp = fileOpen(filename,"r");
 prompts = NULL;
 menuList = NULL;
 j = i = 0;
 while (fgets(buffer, 256, fp) != NULL)
```

```
        {
        tabs = allTrim(buffer);
        if ((len = strlen(buffer)) == 0)
                continue;
        if ((j = inArray(prompts, buffer, i)) == 0)
                        j = i;
        else
                j--;
        level[tabs + 1] = j + 1;
        menuList = (int **)realloc(menuList, ++m * sizeof(int *));
        menuList[m - 1] = (int *)malloc(3 * sizeof(int));
        menuList[m - 1][0] = level[tabs];
        menuList[m - 1][1] = level[tabs + 1];
        menuList[m - 1][2] = 0;
        if (j == i)
                addWord(&prompts, buffer, ++i);
        }
        numMenu = m;
for (j = 0; j < numMenu; j++)
        {
        for(m = 0; m < numMenu; m++)
                if (menuList[j][1] == menuList[m][0])
                        break;
        if (m != numMenu)  /* Leaf Node */
                continue;
        nc = 0;
        for (m = 0; m < numMenu; m++)
                if (menuList[m][0] == menuList[j][0])
                        nc++;
        if (nc != 1)
                {
                menuList[j][2] = 99;
                continue;
                }
        len = strlen(prompts[menuList[j][1] - 1]);
        if (prompts[menuList[j][1] - 1][len - 1] == '?')
                menuList[j][2] = 99;
        else
                {
                for (m = 0; m < numMenu; m++)
                        if (menuList[m][1] == menuList[j][0])
                                menuList[m][2] = menuList[j][1];
                menuList[j][2] = 100;
```


[redacted]

```
                }
        }
return i;
}


void fillIndex()
{
int i, j, k;

indexList = (int **)malloc(numColumn * sizeof(int *));
for (i = 0; i < numColumn; i++) {
        indexList[i] = (int *)malloc(numIndex * sizeof(int));
        for (j = 0; j < numIndex; j++)
                indexList[i][j] = 0;
        }

for (i = 0; i < numIndex; i++)
                updateThesaurus(prompts[i], i + 1);
updateFrmForms();

}

updateThesaurus( char *str, int pmpt)
{
char tmpstr[256];
char *sarray[50];
int i, j, k, wrds;
int iflag = 0, dflag = 0;

strcpy(tmpstr, str);
wrds = readValues(tmpstr, sarray);
stemArray(sarray, wrds);
for (i = 0; i < wrds; i++)
        {
        for (j = 0; j < numColumn; j++)
                {
                if (!strcmp(columnTerms[j], sarray[i]))
                        {
                        iflag = 1;
                        dflag = 0;
                        for (k = 0; indexList[j][k] && k < numIndex; k++)
                                if (indexList[j][k] == pmpt)
```

```
                                        dflag = 1;
                        if (k < numIndex && dflag == 0)
                                indexList[j][k] = pmpt;
                        break;
                        }
                }
        }
if (iflag == 0)
        {
        printf("** warning the following prompt does not contain index word\n");
        printf("\t%s\n", str);
        }
}


/*****************************************************************
 Create Matrix here
*****************************************************************/
void createMatrix(char * filename) {
 int i, j, nwp = 0, k = 0;
 int minv;
 FILE * fp;

 /* allocate memory for matrix */
 matrix = (double **)malloc(numRow * sizeof(double *));
 for (i = 0; i < numRow; ++i) {
        matrix[i] = (double *)malloc(numColumn * sizeof(double));
        for (j = 0; j < numColumn; j++)
                matrix[i][j] = 0;
        }

 /* allocate memory for rowcount and column count */
 rowcount = (int *)malloc(numRow * sizeof(int));
 colcount = (int *)malloc(numColumn * sizeof(int));

 /* Go to start of document */

 fp = fileOpen(filename,"r");

 while (!feof(fp))
        {
        /* initialize rowcount array */
        for (j = 0; j < numRow; j++)
```

```
            rowcount[j] = 0;
        /* initialize columncount array */
        for (j = 0; j < numColumn; j++)
            colcount[j] = 0;

        nwp = readPara(fp);
        if (feof(fp))
            break;
        if (nwp == 0)
            continue;

        /* add co-occurance of rowword & colword to the matrix */
        for (j = 0; j < numRow; j++)
            for (k = 0; k < numColumn; k++) {
                minv = min(rowcount[j], colcount[k]);
                matrix[j][k] += minv;
                }
        }
}


int readPara(FILE *fp )
{
int i, j, k, wcount = 0, m = 0;
int nextpara, currpara, wordLen;
char tmpword[50];

currpara = ftell(fp);
wcount = wordsInPara(fp);
if (feof(fp)) {
    if (wcount == 0)
        return 0;
    }

nextpara =  ftell(fp);
fseek(fp, currpara, 0);
for (i = 0; i < wcount; ++i) {
    wordLen = fetchWord(fp, tmpword);
    if (inArray(stopWords,tmpword,numStopWord))
        continue;
    stemWord(tmpword);

        /* count the occurance of each word from the row in para */
```

```
                for (j = 0; j < numRow; j++)
                        if (!strcmp(rowTerms[j], tmpword)) {
                                rowcount[j]++;
                                break;
                                }
                /* count the occurance of each word from the column in para */
                for (j = 0; j < numColumn; j++)
                        if (!strcmp(columnTerms[j], tmpword)) {
                                colcount[j]++;
                                break;
                                }

 }
 fseek(fp, nextpara, 0);
 return wcount;
}

int wordsInPara (FILE *fp)
{
 int c, count = 0;
 int state;
 const int out = 0, in = 1;

 state = out;
 while ((c = getc(fp)) != EOF) {
        if (!isalpha(c)) {
                if (c == '\n' || c == EOF)
                        break;
                state = out;
        }
        else
        if (state == out) {
                state = in;
                count++;
        }
 }
 return count;
}

/*************************************************************
Calculate Cosine Function
*************************************************************/
void calcCosine()
```

```
{
int i, j, k, sum;

/* memory allocation for the cosine matrix */

cosine = (double **)malloc(numRow * sizeof(double *));
for (i = 0; i < numRow; ++i) {
        cosine[i] = (double *)malloc(((numRow) * sizeof(double)));
        for (j = 0; j < numRow; j++)
                cosine[i][j] = 0;
        }

/*Normalization*/

for (i = 0; i < numRow; ++i)
        {
        sum = 0;
        for (k = 0; k < numColumn; ++k)
                sum += matrix[i][k] * matrix[i][k];
        if (sum != 0)
                {
                for (j = 0; j < numColumn; ++j)
                        matrix[i][j] = matrix[i][j] / sqrt(sum);
                }
        }

 /*Cosines*/

for (i = 0; i < numRow; ++i)
        {
        for (k = i + 1; k < numRow; ++k)
                {
                cosine[i][k] = 0;
                for (j = 0; j < numColumn; ++j)
                        cosine[i][k] += matrix[i][j] * matrix[k][j];
                }
        }
}

/*********************************************************************
eraseZeroes : removes the row with all zero column in the matrix
*********************************************************************/
int eraseZeroes() {
```

```
int j, k;
int cond;

/* Free and nullify the rowTerms and matrix row for all zeroes */
for (j = 0; j < numRow; ++j) {
  cond = 1;
  for (k = 0; k < numColumn; ++k) {
    if (matrix[j][k] != 0) {
        cond = 0;
        break;
    }
  }
  if (cond == 1) {
        rowTerms[j] = NULL;
        matrix[j] = NULL;
    }
  }

/* Push NULL rows at the end of arrays */
for (j = 0; j < numRow; j++)
  {
  if (rowTerms[j] == NULL)
        {
        for (k = j + 1; k < numRow; k++)
                if (rowTerms[k] != NULL)
                        break;
        if (k < numRow)
          {
          rowTerms[j] = rowTerms[k];
          matrix[j] = matrix[k];
          rowTerms[k] = NULL;
          matrix[k] = NULL;
          }
    }
  }

/* count new numRow */
for (j = 0; (rowTerms[j] != NULL) && j < numRow; j++);

return j;

}
```

```
/*************************************************************************
createThesaurus: Function to Create Thesaurus of rowTerms by taking the
index words matching the top 5 cosine values.
*************************************************************************/
void createThesaurus()
{
int i, j, k, l;
int m, numword;
double *tmpcos, prevcosine = 0;
int *colnum;

tmpcos = (double *)malloc(numRow * sizeof(double));
colnum = (int *)malloc(numRow * sizeof(int));

thesaurus = (int **)malloc(numRow * sizeof(int *));
for (i = 0; i < numRow; i++) {
        thesaurus[i] = (int *)malloc(numColumn * sizeof(int));
        for (j = 0; j < numColumn; j++)
                thesaurus[i][j] = 0;
        }

/* initialization of thesaurus */

for (i = 0; i < numRow; i++) {
        for ( j = 0; j < numRow; j++) {
                if (i > j)
                        tmpcos[j] = cosine[j][i];
                else
                        if (i < j)
                                tmpcos[j] = cosine[i][j];
                        else
                                tmpcos[j] = 0;
                colnum[j] = j;
                }
        floatSort(colnum, tmpcos, numRow);

        numword = prevcosine = 0;

        /* count top 'topValues' of cosine */
        for (m = 0; m < numColumn; m++) {
                if (prevcosine != tmpcos[m])
                        numword++;
```

```
                prevcosine = tmpcos[m];
                if (numword == topValues + 1)
                        break;
                }
        --m;
        /* m = total num of syn */

        for (j = k = 0; k <= m; k++) {
                if ((l = inArray(columnTerms,rowTerms[colnum[k]], numColumn)) != 0)
                        if (tmpcos[k] != 0) {
                                thesaurus[i][j] = l;
                                j++;
                                }
                }

        }
}

/*************************************************************************
floatSort : Sorts the array of cosine values and corresponding index of
index words in reverse order.
*************************************************************************/
floatSort(int *colnum, double *tmpcos, int numRow)
{
int i, j, k;
double f;
for (i = numRow - 1; i > 0; i--)
        for (j = 0; j < i; j++) {
                if (tmpcos[j] < tmpcos[j + 1]) {
                        f = tmpcos[j], k = colnum[j];
                        tmpcos[j] = tmpcos[j + 1], colnum[j] = colnum[j + 1];
                        tmpcos[j + 1] = f, colnum[j + 1] = k;
                        }
                }
}

void saveData(char *filenm)
{
int i, j, k, l;
FILE *fp;

fp = fileOpen(filenm, "w");
```

```
printArray(fp, "PROMPTS", prompts, NULL, numIndex, 0); // Write Prompts to the file

/********** Write Menu-Tree to the file **********/
// printArray(fp, "MENUTREE", NULL, menuList, numMenu, 2);
fprintf(fp, "[%s]\n", "MENUTREE");
for (i = 0; i < numMenu; i++)
        fprintf(fp, "%d,%d,%d\n", menuList[i][0], menuList[i][1],menuList[i][2]);
fprintf(fp, "\n");

printArray(fp, "INDEX", columnTerms, indexList, numColumn, numIndex); // Write Index to
the file
printArray(fp, "THESAURUS", rowTerms, thesaurus, numRow, numColumn); // Write
Thesaurus to the file

fclose(fp);
printf("Data saved in %s\n",filenm);
}

printArray(FILE *fp, char *head, char **cArray, int **iArray, int cNum, int iNum)
{
int i,j;
fprintf(fp, "[%s]\n", head);
for (i = 0; i < cNum; i++)
        {
        fprintf(fp, "%s ", cArray[i]);
        for (j = 0; j < iNum && iArray[i][j] != 0; j++)
                fprintf(fp, "%d,", iArray[i][j]);
        fprintf(fp, "\n");
        }
fprintf(fp, "\n");
}



int  addFormKeys(char ***cArray, int count)
{
char **wordList, *tmparray[20];
int i, j, k, words;
int l, tmpcount;

wordList = NULL;
words = 0;
for(i = 0; i < numForms; i++)
```

```
        for (j = 0; j < formlist[i]->numFields; j++)
                {
                if (!strcmp("MChoice", formlist[i]->fields[j]->Type))
                        for(k = 0; k < formlist[i]->fields[j]->numChoice; k++)
                                {
                                tmpcount = createArray(formlist[i]->fields[j]->Choice[k],
tmparray);
                                for(l = 0; l < tmpcount; l++)
                                        addWord(&wordList, tmparray[l], ++words);
                                }
                }

i = mergeArray(cArray, wordList, count, words);
sortArray((*cArray), i);
return i;

}



updateFrmForms()
{
int i, j, k, l;
int m, n, x, tmpcount;
int pmpt;
char *tmpstr, *tmparray[20];

for (i = 0; i < numPF; i++)
        {
        pmpt = inArray(prompts, Fprompts[i][1], numIndex);
        if (pmpt == 0)
                {
                printf("Unknown prompt encountered for form %s\n",Fprompts[i][0]);
                exit(1);
                }
        for(j = 0; j < numForms; j++)
                if (!strcmp(Fprompts[i][0], formlist[j]->name))
                        break;
        if (j == numForms)
                continue;
        for(k = 0; k < formlist[j]->numFields; k++)
                {
                if (strcmp(formlist[j]->fields[k]->Type,"MChoice"))
```

```
                continue;
            for(l = 0; l < formlist[j]->fields[k]->numChoice; l++)
                {
                tmpcount = createArray(formlist[j]->fields[k]->Choice[l], tmparray);
                for(m = 0; m < tmpcount; m++)
                    {
                    n = inArray(columnTerms, tmparray[m], numColumn);
                    n--;
                    for (x = 0; indexList[n][x] && x < numIndex; x++)
                        if (indexList[n][x] == pmpt)
                            break;
                    if (x < numIndex && indexList[n][x] == 0)
                        indexList[n][x] = pmpt;
                    }
                }
            }
        }
}
```

arraylib.c: This program contains general purpose functions

```
#include <stdio.h>
#include <string.h>
#include "globalvar.h"
#include "forms.h"

FILE * fileOpen(char *, char *);

int fetchWord(FILE *f, char * wrd) {
int i = 0, c;
wrd[0] = 0;
if (feof(f))
        return 0;
while(!isalpha(c = fgetc(f)))
        if (c == EOF)
                return 0;
do {
        wrd[i++] = tolower(c);
        } while(isalpha(c = fgetc(f)));
wrd[i] = 0;
return i;
}
```

```
int inArray(char **array, char *word, int length)
{
int i;

for (i = 0; i < length; i++)
        if (array[i] != NULL && !strcmp(array[i], word))
                        return i + 1;
return 0;
}

int removeNulls(char **strarray, int numWords)
{
int i, j;

for (i = 0; i < numWords; i++)
        {
        if (strarray[i] == NULL)
                {
                for (j = i + 1; j < numWords; j++)
                        if (strarray[j] != NULL)
                                {
                                strarray[i] = strarray[j];
                                strarray[j] = NULL;
                                break;
                                }
                }
        }

/* get count of filtered words */
for (j = 0; (strarray[j] != NULL) && (j < numWords); j++);
return j;
}

int mergeArray(char ***Array1, char **Array2, int numArray1, int numArray2) {
int i;

for (i = 0; i < numArray2; i++)
        if (! inArray((*Array1), Array2[i], numArray1))
                addWord(Array1, Array2[i], ++numArray1);

return numArray1;
}
```

```
int readValues(char *str, char **array)
{
int i , j = 0, c;
int state;
const int out = 0, in = 1;

state = out;
for (i = 0; (c = str[i]) != 0; i++)
        {
        if (!isalnum(c))   /* alfa-numeric to read numbers also  */
                {
                state = out;
                str[i] = 0;     /* word is over end it with null */
                }
        else
                {
                str[i] = tolower(c);
                if (state == out)
                        {
                        state = in;
                        array[j++] = str + i; /* word started, store the ptr.*/
                }
                }
        }
return j;
}

void sortArray(char *allwords[], int numwords) {
 int i = 0;
 int j = 0;
 char *tmp;

 for (i = 0; i < numwords; ++i)
  for (j = i + 1; j < numwords; ++j)
   if (strcmp(allwords[i], allwords[j]) > 0) {
       tmp = allwords[i];
       allwords[i] = allwords[j];
       allwords[j] = tmp;
   }
}

loadStopWords( char * filename) {
```

```
FILE * fp;
char tmpWord[50];
int wordLen = 0;

numStopWord = 0;
fp = fileOpen(filename,"r");
stopWords = NULL;
while((wordLen = fetchWord(fp, tmpWord)) != 0)
                addWord(&stopWords, tmpWord, ++numStopWord);
}

FILE * fileOpen(char *filename, char *mode)
{
FILE * fp;

if ((fp = fopen(filename, mode)) == NULL) {
                perror(filename);
                exit(1);
                }
return fp;
}

addWord(char ***cArray, char * word, int c)
{
*cArray = (char **) realloc(*cArray, c * sizeof(char *));
(*cArray)[c - 1] = strdup(word);
}

int removeZeros(int *intArray, int numInt)
{
int i, j;

for (i = 0; i < numInt; i++)
        {
        if (intArray[i] == 0)
                {
                for (j = i + 1; j < numInt; j++)
                        if (intArray[j] != 0)
                                {
                                intArray[i] = intArray[j];
                                intArray[j] = 0;
                                break;
                                }
```

```
                }
        }

/* get count of filtered integers */
for (j = 0; (intArray[j] != 0) && (j < numInt); j++);
return j;
}

/*************************************************************************88
Newly added functions ( for further reducing the code )
*************************************************************************/
int breakStr(char * str, char **strarray)
{
char c, *tmpstr;
int i, j = 0;
int state;
const int out = 0, in = 1;
/* Seperate the sentence into individual words */
tmpstr = strdup(str);
state = out;
for (i = 0; (c = tmpstr[i]) != 0; i++)
        {
        if (!isalpha(c))
                {
                state = out;
                tmpstr[i] = 0;
                }
        else
                {
                tmpstr[i] = tolower(c);
                if (state == out)
                        {
                        state = in;
                        strarray[j++] = tmpstr + i;
                }
                }
        }
return j;
}

/* remove stopWords */
filterStopWords(char ** strarray, int numWords)
{
```

```
int i;

for (i = 0; i < numWords; i++)
        if (inArray(stopWords, strarray[i], numStopWord))
                        strarray[i] = NULL;
}

/* remove duplicates */
filterDuplicates(char ** strarray, int numWords)
{
int i;
for (i = 0; i < numWords; i++)
        if (strarray[i] != NULL && inArray(strarray, strarray[i], i))
                        strarray[i] = NULL;
}

int loadFormsList( char *filename)
{

char buf[256];
FILE *fp;
int len, i;

fp = fileOpen(filename, "r");

Fprompts = NULL;
numPF = 0;
while (fgets(buf,255,fp) != NULL)
        {
        len = strlen(buf);
        for (i = 0; i < len; i++)
                if (buf[i] == ':')
                        {
                        buf[i] = 0;
                        break;
                        }
        if (i == len)
                {
                fprintf(stderr, "Error in Prompt list\n");
                exit(0);
                }
        allTrim(buf);
        allTrim(buf + i + 1);
```

```
        Fprompts = (char ***)realloc(Fprompts, (++numPF)*sizeof(char **));
        Fprompts[numPF-1] = (char **)malloc(2 * sizeof(char *));
        Fprompts[numPF-1][0] = strdup(buf);
        Fprompts[numPF-1][1] = strdup(buf + i + 1);
        }
fclose(fp);
return numPF;
}

int loadForms(char * filename)
{
int i, j, formcount = 0;
FILE *fp;
char buf[80], **namelist = NULL;
formlist = NULL;
numForms = 0;

fp = fileOpen(filename, "r");
while(fgets(buf,79,fp) != NULL)
        {
        if (buf[0] == '[')
                {
                for(i = 0; buf[i]; i++)
                        if (buf[i] == '[' || buf[i] == ']')
                                buf[i] = ' ';
                allTrim(buf);
                addWord(&namelist, buf, ++formcount);
                }
        }

for ( i = 0; i < formcount; i++)
        {
        formlist = (struct form **)realloc(formlist, (++numForms) * sizeof(struct form *));
        formlist[numForms - 1] = (struct form*)malloc(sizeof(struct form));
        loadForm(fp , formlist[numForms - 1], namelist[i]);
        }
fclose(fp);
return numForms;
}

int allTrim (char * str)
{
        int i, j, sf, tabs;
```

```
        for (i = tabs = 0; isspace(str[i]) && str[i]; i++)
                tabs += (str[i] == '\t')? 1: 0;
        for ( j = sf = 0; str[i]; i++, j++)
                str[j] = iscntrl(str[i])? ' ': str[i];
        for(str[j--] = 0; isspace(str[j]) && j > 0; str[j--] = 0);
        return tabs;
}

int createArray (char * str, char ** array)
{
int count;
count = breakStr(str, array);
return processArray(array, count, 1);
}

int processArray(char ** array, int count, int sflag)
{
if (sflag)
        filterStopWords(array, count);
stemArray(array, count);
filterDuplicates(array, count);
return removeNulls(array, count);
}
```

stemlib.c: This program contains functions related to stemming algorithim

```
/* This is the Porter stemming algorithm, coded up in ANSI C by the author.
It may be be regarded as cononical, in that it follows the algorithm presented
in Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14, no. 3,
pp 130-137, only differing from it at the points maked --DEPARTURE-- below.

  See also http://www.tartarus.org/~martin/PorterStemmer

  The algorithm as described in the paper could be exactly replicated
  by adjusting the points of DEPARTURE, but this is barely necessary,
  because (a) the points of DEPARTURE are definitely improvements, and
  (b) no encoding of the Porter stemmer I have seen is anything like
  as exact as this version, even with the points of DEPARTURE!

  You can compile it on Unix with 'gcc -O3 -o stem stem.c' after which
  'stem' takes a list of inputs and sends the stemmed equivalent to
```

stdout.

The algorithm as encoded here is particularly fast.

Release 1
*/

```
#include <string.h>  /* for memmove */

#define TRUE 1
#define FALSE 0

/* The main part of the stemming algorithm starts here. b is a buffer
   holding a word to be stemmed. The letters are in b[k0], b[k0+1] ...
   ending at b[k]. In fact k0 = 0 in this demo program. k is readjusted
   downwards as the stemming progresses. Zero termination is not in fact
   used in the algorithm.

   Note that only lower case sequences are stemmed. Forcing to lower case
   should be done before stem(...) is called.
*/

static char * b;       /* buffer for word to be stemmed */
static int k,k0,j;     /* j is a general offset into the string */

/* cons(i) is TRUE <=> b[i] is a consonant. */

int cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return FALSE;
      case 'y': return (i==k0) ? TRUE : !cons(i-1);
      default: return TRUE;
   }
}

/* m() measures the number of consonant sequences between k0 and j. if c is
   a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
   presence,

      <c><v>       gives 0
      <c>vc<v>     gives 1
      <c>vcvc<v>   gives 2
      <c>vcvcvc<v> gives 3
```

```
    ....
*/

int m()
{  int n = 0;
   int i = k0;
   while(TRUE)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(TRUE)
   {  while(TRUE)
      {  if (i > j) return n;
           if (cons(i)) break;
           i++;
      }
      i++;
      n++;
      while(TRUE)
      {  if (i > j) return n;
         if (! cons(i)) break;
         i++;
      }
      i++;
   }
}

/* vowelinstem() is TRUE <=> k0,...j contains a vowel */

int vowelinstem()
{  int i; for (i = k0; i <= j; i++) if (! cons(i)) return TRUE;
   return FALSE;
}

/* doublec(j) is TRUE <=> j,(j-1) contain a double consonant. */

int doublec(int j)
{  if (j < k0+1) return FALSE;
   if (b[j] != b[j-1]) return FALSE;
   return cons(j);
}
```

```
/* cvc(i) is TRUE <=> i-2,i-1,i has the form consonant - vowel - consonant
   and also if the second c is not w,x or y. this is used when trying to
   restore an e at the end of a short word. e.g.

      cav(e), lov(e), hop(e), crim(e), but
      snow, box, tray.

*/

int cvc(int i)
{  if (i < k0+2 || !cons(i) || cons(i-1) || !cons(i-2)) return FALSE;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return FALSE;
   }
   return TRUE;
}

/* ends(s) is TRUE <=> k0,...k ends with the string s. */

int ends(char * s)
{  int length = s[0];
   if (s[length] != b[k]) return FALSE; /* tiny speed-up */
   if (length > k-k0+1) return FALSE;
   if (memcmp(b+k-length+1,s+1,length) != 0) return FALSE;
   j = k-length;
   return TRUE;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

void setto(char * s)
{  int length = s[0];
   memmove(b+j+1,s+1,length);
   k = j+length;
}

/* r(s) is used further down. */

void r(char * s) {  if (m() > 0) setto(s); }

/* step1ab() gets rid of plurals and -ed or -ing. e.g.
```

```
      caresses -> caress
      ponies   -> poni
      ties     -> ti
      caress   -> caress
      cats     -> cat

      feed     -> feed
      agreed   -> agree
      disabled -> disable

      matting  -> mat
      mating   -> mate
      meeting  -> meet
      milling  -> mill
      messing  -> mess

      meetings -> meet

*/

void step1ab()
{  if (b[k] == 's')
   {  if (ends("\04" "sses")) k -= 2; else
      if (ends("\03" "ies")) setto("\01" "i"); else
      if (b[k-1] != 's') k--;
   }
   if (ends("\03" "eed")) { if (m() > 0) k--; } else
   if ((ends("\02" "ed") || ends("\03" "ing")) && vowelinstem())
   {  k = j;
      if (ends("\02" "at")) setto("\03" "ate"); else
      if (ends("\02" "bl")) setto("\03" "ble"); else
      if (ends("\02" "iz")) setto("\03" "ize"); else
      if (doublec(k))
      {  k--;
         {  int ch = b[k];
            if (ch == 'l' || ch == 's' || ch == 'z') k++;
         }
      }
      else if (m() == 1 && cvc(k)) setto("\01" "e");
   }
}

/* step1c() turns terminal y to i when there is another vowel in the stem. */
```

```
void step1c() { if (ends("\01" "y") && vowelinstem()) b[k] = 'i'; }


/* step2() maps double suffices to single ones. so -ization ( = -ize plus
  -ation) maps to -ize etc. note that the string before the suffix must give
  m() > 0. */

void step2() { switch (b[k-1])
{
  case 'a': if (ends("\07" "ational")) { r("\03" "ate"); break; }
          if (ends("\06" "tional")) { r("\04" "tion"); break; }
          break;
  case 'c': if (ends("\04" "enci")) { r("\04" "ence"); break; }
          if (ends("\04" "anci")) { r("\04" "ance"); break; }
          break;
  case 'e': if (ends("\04" "izer")) { r("\03" "ize"); break; }
          break;
  case 'l': if (ends("\03" "bli")) { r("\03" "ble"); break; } /*-DEPARTURE-*/

/* To match the published algorithm, replace this line with
  case 'l': if (ends("\04" "abli")) { r("\04" "able"); break; } */

          if (ends("\04" "alli")) { r("\02" "al"); break; }
          if (ends("\05" "entli")) { r("\03" "ent"); break; }
          if (ends("\03" "eli")) { r("\01" "e"); break; }
          if (ends("\05" "ousli")) { r("\03" "ous"); break; }
          break;
  case 'o': if (ends("\07" "ization")) { r("\03" "ize"); break; }
          if (ends("\05" "ation")) { r("\03" "ate"); break; }
          if (ends("\04" "ator")) { r("\03" "ate"); break; }
          break;
  case 's': if (ends("\05" "alism")) { r("\02" "al"); break; }
          if (ends("\07" "iveness")) { r("\03" "ive"); break; }
          if (ends("\07" "fulness")) { r("\03" "ful"); break; }
          if (ends("\07" "ousness")) { r("\03" "ous"); break; }
          break;
  case 't': if (ends("\05" "aliti")) { r("\02" "al"); break; }
          if (ends("\05" "iviti")) { r("\03" "ive"); break; }
          if (ends("\06" "biliti")) { r("\03" "ble"); break; }
          break;
  case 'g': if (ends("\04" "logi")) { r("\03" "log"); break; } /*-DEPARTURE-*/
```

```
 /* To match the published algorithm, delete this line */

} }

/* step3() deals with -ic-, -full, -ness etc. similar strategy to step2. */

void step3() { switch (b[k])
{
   case 'e': if (ends("\05" "icate")) { r("\02" "ic"); break; }
             if (ends("\05" "ative")) { r("\00" ""); break; }
             if (ends("\05" "alize")) { r("\02" "al"); break; }
             break;
   case 'i': if (ends("\05" "iciti")) { r("\02" "ic"); break; }
             break;
   case 'l': if (ends("\04" "ical")) { r("\02" "ic"); break; }
             if (ends("\03" "ful")) { r("\00" ""); break; }
             break;
   case 's': if (ends("\04" "ness")) { r("\00" ""); break; }
             break;
} }

/* step4() takes off -ant, -ence etc., in context <c>vcvc<v>. */

void step4()
{  switch (b[k-1])
   {  case 'a': if (ends("\02" "al")) break; return;
      case 'c': if (ends("\04" "ance")) break;
                if (ends("\04" "ence")) break; return;
      case 'e': if (ends("\02" "er")) break; return;
      case 'i': if (ends("\02" "ic")) break; return;
      case 'l': if (ends("\04" "able")) break;
                if (ends("\04" "ible")) break; return;
      case 'n': if (ends("\03" "ant")) break;
                if (ends("\05" "ement")) break;
                if (ends("\04" "ment")) break;
                if (ends("\03" "ent")) break; return;
      case 'o': if (ends("\03" "ion") && (b[j] == 's' || b[j] == 't')) break;
                if (ends("\02" "ou")) break; return;
                /* takes care of -ous */
      case 's': if (ends("\03" "ism")) break; return;
      case 't': if (ends("\03" "ate")) break;
                if (ends("\03" "iti")) break; return;
      case 'u': if (ends("\03" "ous")) break; return;
```

```
      case 'v': if (ends("\03" "ive")) break; return;
      case 'z': if (ends("\03" "ize")) break; return;
      default: return;
    }
    if (m() > 1) k = j;
}

/* step5() removes a final -e if m() > 1, and changes -ll to -l if
   m() > 1. */

void step5()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/* In stem(p,i,j), p is a char pointer, and the string to be stemmed is from
   p[i] to p[j] inclusive. Typically i is zero and j is the offset to the last
   character of a string, (p[j+1] == '\0'). The stemmer adjusts the
   characters p[i] ... p[j] and returns the new end-point of the string, k.
   Stemming never increases word length, so i <= k <= j. To turn the stemmer
   into a module, declare 'stem' as extern, and delete the remainder of this
   file.
*/

int stem(char * p, int i, int j)
{  b = p; k = j; k0 = i; /* copy the parameters into statics */
   if (k <= k0+1) return k; /*-DEPARTURE-*/

   /* With this line, strings of length 1 or 2 don't go through the
      stemming process, although no mention is made of this in the
      published algorithm. Remove the line to match the published
      algorithm. */

   step1ab(); step1c(); step2(); step3(); step4(); step5();
   return k;
}

/*--------------------stemmer definition ends here------------------------*/
```

```
stemArray(char **list, int arrayLen)
{
int i;
for (i = 0; i < arrayLen; i++)
        if (list[i] != NULL)
                stemWord(list[i]);
}

stemWord( char * s)
{
s[stem(s,0, strlen(s) - 1) + 1] = 0;
}
```

dialog.c: This is main program of dialog module

```
/*****************************************************************************
dialog.c : The main function for the interactive dialog program. loads all
the global arrays and variables before calling the interacitve function.

Arguments are:
1. The Configuration file for Thesaurus. contains Prompts, index,
   basic thesaurus etc.
2. The Learning Thesaurus. - used to store learnt words and to refer to them.
*****************************************************************************/
#include <stdio.h>
#include <string.h>
#include "arraylib.h"

int numColumn,  numRow, numIndex, numMenu;
int startPoint, eofFlag, topValues;
char **rowTerms, **columnTerms, **prompts, **stopWords;
float **matrix, **cosine, phoneThreshold, webThreshold;
int **indexList, **menuList, **thesaurus;
int numStopWord, numOrgRow;
int numForms, numPF;
struct form **formlist;
char ***Fprompts, *formfile;
int **scoring, numScore = 0;
char *cfg, *lcfg, *fdoc, *xdoc, *sdoc;
int minPromptCount = 1, timeout = 30;

void Interactive(char *);
```

```
main(int argc, char *argv[])
{
int i = 0;

/*if (argc != 5)
        {
        printf("Usage Instructions: dialog config_file learn_file\n");
        printf("*** Exiting, goodbye.\n");
        exit(1);
        }*/
if (argc != 2)
        {
        printf("Usage Instructions: d <ini-file>\n");
        printf("*** Exiting, goodbye.\n");
        exit(1);
        }
readini(argv[1]);
formfile = fdoc;
loadStopWords(sdoc);
numPF = loadFormsList(xdoc);
numForms = loadForms(fdoc);
loadData(cfg, lcfg);
Interactive(lcfg);
}

/*********************************************************************
loaddata : This function will read the configuration files and load the
        information into the relevant global arrays.
*********************************************************************/
loadData(char *filenm, char *file2)
{
char buf[256], word[20];
int i, j, k, l;
int numext;
FILE *fp, *f2;

/****************** open configuration file *******************/
fp = fileOpen(filenm, "r");

/************** open learn(extended thesaurus) file ***************/
f2 = fileOpen(file2, "r");
```

```
prompts = columnTerms = rowTerms = NULL;
scoring = thesaurus = indexList = menuList = NULL;

/* read data in the arrays */
numMenu = loadMenuTree(fp, "[MENUTREE]");
numIndex = readArray(fp, "[PROMPTS]", &prompts, 1, NULL, 0, 0);
numColumn = readArray(fp, "[INDEX]", &columnTerms, 1, &indexList, numIndex, 0);
numOrgRow = readArray(fp, "[THESAURUS]", &rowTerms, 1, &thesaurus, numColumn, 0);
numRow = readArray(f2, "[EXT-THESAURUS]", &rowTerms, 1, &thesaurus, numColumn,
numOrgRow);
numScore = readArray(f2, "[SCORING]", NULL, 0, &scoring, numColumn + 1, 0);

fclose(fp);
fclose(f2);
}

/*****************************************************************************
readArray : Reads the file and fills the rows and columns of the given arrays
*****************************************************************************/
int readArray(FILE *fp, char *head, char ***ch_array, int ccount, int ***int_array, int icount, int
sp)
{
char buf[256];
int i, j, start = 0, wc = 0;
int k, c;
char **tmparray; /*To store the pointers to the words/numbers from the string*/
c = sp;

if (icount != 0)
        tmparray = (char **)malloc((icount + 1) * sizeof(char *));

fseek(fp, 0, 0); /* Go to Top */

while (fgets(buf, 255, fp) != NULL) /* read lines till end of file */
        {
        allTrim(buf);
        j = strlen(buf);
        if (buf[j - 1] == '\n') buf[j - 1] = 0;
        if (start)
                {
                if (strlen(buf) == 0)  /* if blank line, stop reading */
                        break;
                if (icount == 0) /* i.e. no integer array */
```

```
                addWord(ch_array, buf, ++c);
        else    /* read first word string */
                { /* rest are columns of int array */
                wc = readValues(buf, tmparray);
                c++;
                (*int_array) = (int **)realloc(*int_array, c * sizeof(int *));
                (*int_array)[c - 1] = (int *)malloc(icount * sizeof(int));
                if (ccount != 0)
                        addWord(ch_array, tmparray[0], c);
                else
                        (*int_array)[c - 1][0] = atoi(tmparray[0]);
                for (k = 1; k < icount; k++)
                        if (k < wc)
                                (*int_array)[c - 1][k - ccount] = atoi(tmparray[k]);
                        else
                                (*int_array)[c - 1][k - ccount] = 0;
                }
        }
    else
    if (!strcmp(head, buf))
        start = 1;
    }
 return c;
}

/*****************************************************************
loadMenuTree : loads the menutree from file to menuList array
*****************************************************************/
int loadMenuTree (FILE *fp, char *head)
{
char buf[256];
int i, j, start = 0, count = 0;
fseek(fp, 0, 0);
while (fgets(buf, 255, fp) != NULL)
    {
    j = strlen(buf);
    if (buf[j - 1] == '\n')
        buf[j - 1] = 0;
    if (start)
        {
        if (strlen(buf) == 0)
            break;
        menuList = (int **)realloc(menuList, (count + 1) * sizeof(int *));
```

```
                menuList[count] = (int *)malloc(3 * sizeof(int));
                sscanf(buf, "%d,%d,%d\n", &menuList[count][0],
&menuList[count][1],&menuList[count][2]);
                count++;
                }
        else
                if (!strcmp(head, buf))
                start = 1;
        }
return count ;
}


readini(char * filenm)
{
        FILE * fp;
        char buf[80], key[80], value[80], comment[80];
        int cnt;
        if ((fp=fopen(filenm,"r"))==NULL)
        {
                perror(filenm);
                exit(1);
        }
        while (fgets(buf,79,fp)!=NULL)
        {
        sscanf(buf,"%s %s %s",key,value, comment);
        if (!strcmp(key, "sdoc"))
                        sdoc=strdup(value);
        if (!strcmp(key, "fdoc"))
                        fdoc=strdup(value);
        if (!strcmp(key, "xdoc"))
                        xdoc=strdup(value);
        if (!strcmp(key, "cfg"))
                        cfg=strdup(value);
        if (!strcmp(key, "lcfg"))
                        lcfg=strdup(value);
        if (!strcmp(key, "minprompt"))
                        minPromptCount=atoi(value);
        if (!strcmp(key, "timeout"))
                        timeout=atoi(value);
        }
}
```

interactive.c: This program contains funtions related to user interaction

```
/*************************************************************************
Interactive : function to accept a sentence from the user and then
        generate the response.
-------------------------------------------------------------------
thesaurusFlag = is 1 if there is atleast 1 thesaurus/learned word in query
updateFlag = is set to 1 if the program needs to learn (i.e. main menu was
      selected during the prompt navigation)
interPrompts = Intersection of prompts
unionPrompts = Union of prompts
interUnionPrompts = Intersection of Union
numInter  = number of prompts in InterPrompts
numInterUnion = num of prompts in Intersection of Union
numUnion = num of prompts in Union
numUnknown = num of unknown words
*************************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include "globalvar.h"
#include "arraylib.h"
#include "forms.h"

#define max(a,b) (a > b)? a: b
#define min(a,b) (a < b)? a: b
#define swap(a,b) (a ^= b, b ^= a, a ^= b)

extern int numScore, **scoring;
int updateFlag = 0, learnFlag, numQueryList = 0;
FILE *lf, *pf;
char ** uWList=NULL, *queryTerms[50];
int uWNum;
extern int minPromptCount, timeout;
char query[256], **queryList = NULL;
char *affrmWords[] = { "yes", "right", "correct"};
char *negWords[] = { "no", "neither"};
extern char * fdoc;
int otheFlag = 0;
int unknownWords[20], numQuery = 0, numUnknown;
char **uWords;  // Added this array to facilitate learning wven if lateral shift
int numUW;  // Added this to facilitate learning wven if lateral shift
void sayOther();
```

```
void Interactive(char *flnm)
{
 int InterPrompts[20], unionPrompts[20], t1Prompts[20];
 int interUnionPrompts[20], numInterUnion, t2Prompts[20], numT2;
 int i, j, k, l;
 int start, numUnion, numInter, numT1;
 int n, selectedPrompt, thesaurusFlag = 0;
 char *interlog, *processlog, c;

numUnknown = numUW = 0;
for(i=0; i < 20; i++)
        unknownWords[i] = 0;
uWords=NULL;

if ((interlog = (char *)getenv("TIMEOUT")) != NULL)
        timeout = atoi(interlog);

if ((interlog = (char *)getenv("MINPROMPT")) != NULL)
        minPromptCount = atoi(interlog);

if ((interlog = (char *)getenv("INTERLOG")) == NULL)
        interlog = "test.html";

if ((processlog = (char *)getenv("PROCESSLOG")) == NULL)
        processlog = "process.html";

signal(SIGALRM, &sayOther);
lf = fileOpen(interlog,"w");

pf = fileOpen(processlog,"w");
fprintf(lf, "<HTML>\n<TITLE>%s</TITLE>\n<BODY><FONT SIZE=5>\n", interlog);
fprintf(pf, "<HTML>\n<TITLE>%s</TITLE>\n<BODY><FONT SIZE=5>\n", processlog);

system("clear");
printf("Thank you for calling the New Herald.\n");
printf("How may we help you?\n\n");
fprintf(lf,"\nThank you for calling the New Herald.<BR>");
fprintf(lf,"How may we help you. <P>");
fgets(query, 255, stdin);   /* accept the user input */

while (1)
        {
```

```
        addWord(&queryList, query, ++numQueryList);
        numQuery = thesaurusFlag = 0;
        if (strlen(query) == 0)
                break;
        fprintf(lf, "<I> %s</I> <P>", query);

        numQuery = createArray(query, queryTerms);

        /*************************************************************/
        fprintf(pf, "Terms in Query: ");
        for (j = 0; j < numQuery; j++)
                fprintf(pf, " %s", queryTerms[j]);
        fprintf(pf, "<BR>");
        /*************************************************************/

        /* initialize InterPrompts and unionPrompts array */
        for (i = 0; i < 20; i++)
                {
                t2Prompts[i] = t1Prompts[i] = 0;
                InterPrompts[i] = unionPrompts[i] = interUnionPrompts[i] = 0;
                }
        start = 1;
    numInterUnion = numT2 = numT1 = numInter = numUnion = 0;

        /* Scan thru all the words to generate union/intersection of prompts*/
        for (i = 0; i < numQuery; i++)
                {
                /* if not in index words check thesaurus */
                if (!inArray(columnTerms, queryTerms[i], numColumn))
                        {
                        learnFlag = numT1 = numT2 = 0;
                        scanThesaurus(queryTerms[i], t1Prompts, t2Prompts, &numT1,
&numT2);
                        /* if unknown/learned word save it to array */
                        if (learnFlag)
                                {
                                unknownWords[numUnknown] = i, numUnknown++;
                                addWord(&uWords, queryTerms[i], ++numUW);
                                if (numT1 == 0 && numT2 == 0)
                                        continue;
                                else
                                        thesaurusFlag = 1;
                                }
```

```
                else
                        thesaurusFlag = 1;
                }
        else
                {
                numT1 = fetchPrompts(queryTerms[i], t1Prompts);
                numT2 = fetchPrompts(queryTerms[i], t2Prompts);
                /*************************************************/
                fprintf(pf, "%s (index) :", queryTerms[i]);
                for (j = 0; j < numT1; j++)
                        fprintf(pf, " %d", t1Prompts[j]);
                fprintf(pf, "<BR>");
                fflush(pf);
                /*************************************************/
                }

        if (start) /* if first word */
                {
                numInter = PromptUnion(InterPrompts, t2Prompts, numInter, numT2);
                numUnion = PromptUnion(unionPrompts, t1Prompts, numUnion,
numT1);
                numInterUnion = PromptUnion(interUnionPrompts, t1Prompts,
numInterUnion, numT1);
                start = 0;
                }
        else
                {
                numInter = PromptIntersection(InterPrompts, t2Prompts, numInter,
numT2);
                numUnion = PromptUnion(unionPrompts, t1Prompts, numUnion,
numT1);
                numInterUnion = PromptIntersection(interUnionPrompts, t1Prompts,
numInterUnion, numT1);

                }
        }
        /*****************************************************/
        fprintf(pf, "Final Intersection Result: ");
        for (j = 0; j < numInter; j++)
                fprintf(pf," %d", InterPrompts[j]);
        fprintf(pf, "<BR>");
        fprintf(pf, "Final Intersection of Union Result: ");
        for (j = 0; j < numInterUnion; j++)
```

```
                fprintf(pf, " %d", interUnionPrompts[j]);
        fprintf(pf, "<BR>");
        fprintf(pf, "Final Union Result: ");
        for (j = 0;j < numUnion; j++)
                fprintf(pf, " %d", unionPrompts[j]);
        fprintf(pf, "<BR>");
        fflush(pf);
        /*****************************************************/

        if (numInter < minPromptCount && thesaurusFlag)
                {
                if (numInterUnion < minPromptCount)
                        numInter = PromptUnion(InterPrompts, unionPrompts, numInter,
numUnion);
                else
                        numInter = PromptUnion(InterPrompts, interUnionPrompts,
numInter, numInterUnion);
                }

        fprintf(pf, "Final Selection : ");
        fflush(pf);
        for (j = 0; j < numInter; j++)
                fprintf(pf, " %d", InterPrompts[j]);
        fprintf(pf, "<BR>");
        fflush(pf);
        numInter = orderPrompts(InterPrompts, numInter);
        numInter = removeChild(InterPrompts, numInter);
        // eliminate prompts > 3
        for (j = 3; j < numInter; j++)
                InterPrompts[j] = 0;
        numInter = min(numInter, 3);
        fprintf(pf, "Selection After Elimination of descendants: ");
        fflush(pf);
        for (j = 0; j < numInter; j++)
                fprintf(pf, " %d", InterPrompts[j]);
        fprintf(pf, "<BR>");
        fflush(pf);
        selectedPrompt = GetPrompt(InterPrompts, numInter);
        if (selectedPrompt == 100)
                continue;

        // if (updateFlag)
        learnThesaurus(selectedPrompt, unknownWords, numUnknown, flnm);
```

```
        updateFlag = 0;
for(j = 0; (j < numMenu) && (menuList[j][1] != selectedPrompt); j++);
if (menuList[j][2] >= 99)
        {
        for (i = 0; i < numPF; i++)
                {
                if (!strcmp(Fprompts[i][1],prompts[selectedPrompt - 1]))
                        {
                        for(k = 0; k < numForms; k++)
                                if (!strcmp(Fprompts[i][0],formlist[k]->name))
                                        {
                                        fillForm(formlist[k], queryList, numQueryList);
                                        processForm(formlist[k]);
                                        break;
                                        }
                        break;
                        }
                }
        if (i == numPF)
                {
                system("clear");
                printf("\nYour query has been understood.\n");
                printf("Please wait to be transferred to the relevant department.\n\n");
                fprintf(lf,"<P>Your query has been understood.<LI>");
                fprintf(lf,"Please wait to be transferred to the relevant department.<HR>");
                break;
                }
        }
else
        {
        printf("\n%s\n\n",prompts[menuList[j][2] - 1]);
        fprintf(lf, "\n<P>%s<HR>",prompts[menuList[j][2] - 1]);
        }
        // modified for the loop
printf("Do you have another query?\n\n");
fgets(query,80,stdin);
if (!chkNegtn(query))
        {
        for(i = 0; i < numQueryList; i++)
        free(queryList[i]);
        for(i = 0; i < numForms; i++)
                free(formlist[i]);
        free(formlist);
```

```
                free(queryList);
                queryList = NULL;
                numForms = loadForms(fdoc);
                numQueryList = 0;
                //printf("Please tell us your query.\n\n");
                //fgets(query,255,stdin);
                continue;
                }
        system("clear");
        printf("\nThank you for calling.\nGoodbye.\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
        break;
        }

        fprintf(lf, "</HTML>\n</BODY>\n");
        fprintf(pf, "</HTML>\n</BODY>\n");
        fclose(lf);
        fclose(pf);

}

int scanThesaurus(char *word, int *t1Prompts, int *t2Prompts, int* n1, int * n2)
{
int i, j, k = 0, l;
int m, tp[20], sflg = 1;

for (i = 0; i < 20; i++)
        t1Prompts[i] = t2Prompts[i] = tp[i] = 0;

for (l = 0; l < numRow; l++)
        if (!strcmp(word, rowTerms[l]))
                break;
/* if the word is not present in thesaurus */
if (l == numRow)
        {
        fprintf(pf, "%s (unknown) <BR>", word);
        learnFlag = 1;
        return 0;
        }
else
        {
        if (l >= numOrgRow)
                {
                fprintf(pf, "%s (learned):<BR>", word);
```

```
                learnFlag = 1;
                }
        else
                fprintf(pf, "%s (thesaurus):<BR>", word);

        for (j = 0; thesaurus[l][j] && j < numRow; j++)
                {
                m = fetchPrompts(columnTerms[thesaurus[l][j] - 1], tp);
                /*********************************************************/
                fprintf(pf, "<LI>%s (index) :", columnTerms[thesaurus[l][j] - 1]);
                for (k = 0; k < m; k++)
                        fprintf(pf, " %d", tp[k]);
                fprintf(pf, "<BR>");
                /*********************************************************/

                *n1 = PromptUnion(t1Prompts, tp, *n1, m);
                if (sflg)
                        {
                        *n2 = PromptUnion(t2Prompts, tp, *n2, m);
                        sflg = 0;
                        }
                else
                        {
                        *n2 = PromptIntersection(t2Prompts, tp, *n2, m);
                        }
                }
        fprintf(pf, "Union Result: ");
        for (k = 0; k < *n1; k++)
                fprintf(pf, " %d", t1Prompts[k]);
        fprintf(pf, "<BR>");
        fprintf(pf, "Intersection Result: ");
        for (k = 0; k < *n2; k++)
                fprintf(pf, " %d", t2Prompts[k]);
        fprintf(pf, "<BR>");
        }

return k;

}
/*************************************************************
PromptUnion : does a union of arrays pointed by p1 and p2 and
        stores in p1. returns the total elements in result
*************************************************************/
```

```
int PromptUnion(int *p1, int *p2, int n1, int n2)
{
int i, j;
for (i = 0; i < n2; i++)
        {
        for (j = 0; j < n1; j++)
        if (p1[j] == p2[i])
                break;
        if (j == n1)
                {
                p1[j] = p2[i];
                n1++;
                }
        }
return n1;
}

/*************************************************************************
PromptIntersection : does a intersection of arrays pointed by p1 and p2 and
            stores in p1. returns the total elements in result
*************************************************************************/
int PromptIntersection(int *p1, int *p2, int n1, int n2)
{
int i, j;
for (i = 0; i < n1; i++)
        {
        for (j = 0; j < n2; j++)
                if (p1[i] == p2[j])
                        break;
        if (j == n2) /* not there */
                {
                for (j = i; j < n1; j++)
                        p1[j] = p1[j + 1];
                n1--;
                i--;
                }
        }
return n1;
}
/*****************************************************************
fetchPrompts : Will fetch all the prompts for 'word' into
        Arraylist pointed by t1Prompts;
*****************************************************************/
```

```
int fetchPrompts(char *word, int *t1Prompts)
{
int i, j, k, l;
for (i = 0; i < 20; i++)
        t1Prompts[i] = 0;
if ((i = inArray(columnTerms, word, numColumn)) == 0)
        return 0;
i--;
for (j = 0; (t1Prompts[j] = indexList[i][j]) && (j < numIndex); j++);
return j;
}

/*****************************************************************
GetPrompt: Returns the final prompt selected by user
*****************************************************************/
int GetPrompt(int *Parray, int pcnt)
{
int i, j, k, l;
int mmflag = 0, af = 0;
char ans[80];

while (1)
        {
        system("clear");
        printf("\n");
        fprintf(lf, "<P>");
        // Removed the comments to reintroduce last prompt
         if (pcnt == 1 && isLeaf(Parray[0]) && numUnknown > 0)
                af = 1;
        // ----------------------------------------
        if ((pcnt > 1) || (pcnt == 1 && af == 1))
                {
                        // sortPrompts(Parray, pcnt);
                        orderPrompts(Parray, pcnt);
                for (i = 0; i < pcnt; i++)
                        {
                        printf("%s\n\n", prompts[Parray[i] - 1]);
                        fprintf(lf, "<LI>%s", prompts[Parray[i] - 1]);
                        }
                if (!mmflag)
                        {
                        otheFlag = 0;
                        alarm(timeout);
```

```
                }
        fgets(ans, 80, stdin);   /* accept the user input */
        alarm(0);
        fprintf(lf, "<P><I>%s</I><P>", ans);
        fflush(lf);
        if (otheFlag == 1 && chkAfrm(ans))
                j = 0;
        else
        {
                if (chkNegtn(ans) && otheFlag != 1)
                        j = 0;
                else
                {
                        addWord(&queryList, ans, ++numQueryList);
                        j = chkAns(ans, Parray, pcnt);
                }
                otheFlag = 0;
        }
        if ( j == -99)
        {
                updateFlag = 1;
                 return 100;
        }
        if (j < 0)
        {
                pcnt = removeZeros(Parray,pcnt);
                continue;
        }
        mmflag = 0;
        }
else
        j = pcnt;
if (j == 0)
        {
        pcnt = getNodes(j, Parray);
        mmflag = updateFlag = 1;
        }
else
        {
        if (isLeaf(Parray[j - 1]))
                {
                        return Parray[j - 1];
                }
```

```
                else
                        pcnt = getNodes(Parray[j - 1], Parray);
                }
        af = 1;
        }
}

/*************************************************************************
isLeaf: Returns 1 if 'node' is a leaf in the menutree, else 0
*************************************************************************/
int isLeaf(int node)
{
int i;
for (i = 0; i < numMenu; i++)
        if (menuList[i][1] == node)
                break;
if (i == numMenu)
        return 0;
return  menuList[i][2];
}

int getNodes(int pnode, int *parray)
{
int i, j;
for (i = 0, j = 0; i < numMenu; i++)
        if (menuList[i][0] == pnode)
                {
                parray[j] = menuList[i][1];
                j++;
                }
parray[j] = 0;
return j;
}

/*************************************************************************
learnThesaurus : re-writes the thesaurus with relearned pattern and newly
        learned word.
*************************************************************************/
learnThesaurus(int pmpt, int unknownWords[], int numUnknown, char *flnm)
{
int i, j, k, l;
FILE *fp;
int *tmpList, tmpCount;
```

```
/* create and initialize a tmp Array */
tmpList = (int *)calloc(numColumn, sizeof(int));
for (i = tmpCount = 0; i < numColumn; i++)
        tmpList[i] = 0;

/* scan thru the query words  and gather a list of unique keywords in tmp array*/
tmpCount = getKeyWords(queryTerms, numQuery, tmpList);
/* Locate the row for select prompt. if not create new row */
for (k = 0; k < numScore && scoring[k][0] != pmpt; k++);
if (k >= numScore)
{
        scoring = (int **)realloc(scoring, (k + 1) * sizeof(int *));
        scoring[k] = (int *) malloc((numColumn + 1) * sizeof(int));
        for (j = 0; j <= numColumn; j++)
                scoring[k][j] = 0;
        numScore++;
}

scoring[k][0] = pmpt;
for (j = 0; j < tmpCount; j++)
        scoring[k][tmpList[j]]++;

/*****************************************************************/
for (i = tmpCount = 0; i < numColumn; i++)
        tmpList[i] = 0;

for (j = i = 0; j < numColumn; j++)
        {
        for (k = 0; k < numIndex && indexList[j][k] != 0; k++)
                if (indexList[j][k] == pmpt)
                        break;
        if (k < numIndex && indexList[j][k] != 0)
                {
                tmpList[i] = j + 1;
                i++;
                }
        }
tmpCount = i;
fp = fileOpen(flnm, "w");
fprintf(pf, "<BR><B>Learned words</B><BR>");
fprintf(fp, "[%s]\n", "EXT-THESAURUS");
```

```
for (i = numOrgRow; i < numRow; i++)
        {
        fprintf(fp, "%s: ", rowTerms[i]);

        if (updateFlag && inArray(uWords, rowTerms[i], numUW))
                {
                fprintf(pf, "%s (relearned)<BR>original: ", queryTerms[unknownWords[j]]);
                for (k = 0; k < numColumn; k++)
                        {
                        if (thesaurus[i][k] != 0)
                                fprintf(pf, " %d", thesaurus[i][k]);
                        if (thesaurus[i][k] == 0)
                                break;
                        }
                        fprintf(pf, "<BR>");
                        k = PromptUnion(thesaurus[i], tmpList, k, tmpCount);
                        fprintf(pf, "new :");
                        for (j = 0; j < k; j++)
                                fprintf(pf, " %d", thesaurus[i][j]);
                        fprintf(pf, "<BR><BR>");
                }

        for (j = 0; j < numColumn; j++)
                {
                        if (thesaurus[i][j] == 0)
                                break;
                        fprintf(fp, "%d,", thesaurus[i][j]);
                }
        fprintf(fp, "\n");
        }
for (i = 0; updateFlag && i < numUW; i++)
        {
        if (inArray(rowTerms, uWords[i], numRow))
                continue;

        fprintf(fp, "%s: ", uWords[i]);
        fprintf(pf, "%s(new-learned) :", uWords[i]);
        addWord(&rowTerms, uWords[i], ++numRow);
        thesaurus = (int **)realloc(thesaurus, numRow * sizeof(int *));
        thesaurus[numRow - 1] = (int *)malloc(numColumn * sizeof(int));
        for (j = 0; j < numColumn; j++) thesaurus[numRow -1][j] = 0;
        for (j = 0; j < tmpCount; j++)
                        {
```

```
                        thesaurus[numRow - 1][j] = tmpList[j];
                        fprintf(fp, "%d,", tmpList[j]);
                        fprintf(pf, " %d", tmpList[j]);
                        }
        fprintf(fp, "\n");
        fprintf(pf, "<BR><BR>");
        }

fprintf(fp, "\n");

/* write the scoring in the file */
fprintf(fp, "[%s]\n", "SCORING");
for (i = 0; i < numScore; i++)
        {
        fprintf(fp, "%d,", scoring[i][0]);
        for(j = 1; j <= numColumn; j++)
                fprintf(fp, "%d,", scoring[i][j]);
        fprintf(fp, "\n");
        }
fprintf(fp, "\n");
fclose(fp);
}

/*************************************************************************
removeChild: removes descendents of all the elements from the list
*************************************************************************/
int removeChild(int *array, int tot)
{
int i, j, k, cnt = 0;
int *tmparray, rn = 99;

tmparray = (int *)calloc(numIndex, sizeof(int));

/* Remove any prompts that are responses rather than choices */
for (i = 0; i < tot; i++)
        {
        for(j = 0; j < numMenu; j++)
                if (menuList[j][1] == array[i] && menuList[j][2] == 100)
                        array[i] = 0;
        }

/* Remove any prompts that are root node and have a child which is not a leaf rather than
choices */
```

```
for (i = 0; i < tot; i++)
        {
        if (array[i] == 0)  /* already removed so go to next */
                continue;

        /* if (isLeaf(array[i]))
                continue; */

        rn = array[i];
        while(1)
                {
                for(j = 0; j < numMenu; j++)
                        if (menuList[j][1] == rn)
                                break;
                if (menuList[j][0] == 0)
                        break;
                rn = menuList[j][0];
                }
        if ( rn != array[i])
                {
                for (j = 0; j < tot; j++)
                        {
                        if (array[j] == rn)
                                array[j] = 0;
                        }
                }

        }

for (i = 0; i < tot; i++)
        {
        if (array[i] == 0)  /* already removed so go to next */
                continue;
        for (j = 0; j < numIndex; j++)  /* initialize tmparray */
                tmparray[j] = 0;
        cnt = getChildren(array[i], tmparray); /* get children & grand-children of i */
        for (j = 0; j < tot; j++)   /* scan thru the array to check for child */
                if (j != i)  /* ignore self from checking */
                for (k = 0; k < cnt; k++)
                        if (array[j] == tmparray[k])
                                {
                                array[j] = 0; /* if j is child of i, make it 0 */
                                break;
```

```
                        }
        }
/* Shift All non-zeroes upwards */
for (i = 0; i < tot; i++)
        {
        if (array[i] == 0)
                {
                for (j = i + 1; j < tot; j++)
                        if (array[j] != 0)
                                break;
                if (j < tot)
                        {
                        array[i] = array[j];
                        array[j] = 0;
                        }
                }
        }
/* count no of elements */
for (j = 0; j < tot; j++)
        {
        if (array[j] == 0)
                break;
        }
return j;
}

/*************************************************************************
getChildren: fetches all the descendents of pmpt into array
*************************************************************************/
int getChildren(int pmpt, int *array)
{
int i, j, k, l;
int t, t1, t2;
int *tmparray1, *tmparray2;

if (isLeaf(pmpt)) /* if node is leaf no children so return 0 */
        return 0;

tmparray1 = (int *)calloc(numIndex, sizeof(int)); /* child of child in every loop */
tmparray2 = (int *)calloc(numIndex, sizeof(int)); /* union of all scanned children */
t = t1 = t2 = 0;
for (i = 0; i < numMenu; i++)
        {
```

```
        if (menuList[i][0] == pmpt && menuList[i][2] != 100)
                {
                array[t] = menuList[i][1];
                t1 = getChildren(array[t], tmparray1);
                t2 = PromptUnion(tmparray2, tmparray1, t2, t1);
                t++;
                }
        }
t = PromptUnion(array, tmparray2, t, t2);
return t;
}


int chkAns (char * ans, int * Parray, int pcnt)
{
char locquery[256];
int i, j, tmp1cnt = 0, tmp2cnt = 0;
char *resWords[50], start = 'Y';
int numWords, tmpArray1[20], tmpArray2[20];
int uwFlag = 0, rowOrColWord = 0;

strcpy(locquery,ans);
numWords = breakStr(ans, resWords);
if ( strcmp(resWords[0],"other") == 0 && strcmp(resWords[1],"options") == 0)
        {
        return 0;
        }
numWords = processArray(resWords, numWords, 1);
for (i = 0; i < 20; i++)
        tmpArray1[i] = tmpArray2[i] = 0;
fprintf(pf,"<li>Initialized Temp Array\n"); fflush(pf);

for (i = 0; i < numWords; i++)
        {
        if (!inArray(columnTerms, resWords[i], numColumn))
                {
                if (!inArray(rowTerms,resWords[i],numColumn))
                        {
                        if (!inArray(uWList, resWords[i], uWNum))
                                {
                                addWord(&uWList, resWords[i], ++uWNum);
                                fflush(lf);
                                }
```

```
                        else
                                {
                                fprintf(pf,"<li>Unknown Word: %s\n",resWords[i]);
                                fflush(pf);
                                uwFlag = 1; /* unKnown word encountered twice */
                                }
                        }
                else
                        rowOrColWord++;
                continue;
                }
        else
                rowOrColWord++;
        tmp1cnt = fetchPrompts(resWords[i], tmpArray1);
        if (start == 'Y')
                {
                tmp2cnt = PromptUnion(tmpArray2, tmpArray1, tmp2cnt, tmp1cnt);
                start='N';
                }
        else
                tmp2cnt = PromptIntersection(tmpArray2, tmpArray1, tmp2cnt, tmp1cnt);

        tmp2cnt = PromptIntersection(tmpArray2, Parray, tmp2cnt, pcnt);
        }
if (tmp2cnt != 1)
        {
                if (tmp2cnt == 0 && pcnt == 1 && numWords == 1) // i.e. only one prompt &
not selected
                {
                        strcpy(ans, locquery);
                        if (chkAfrm(ans))
                                return 1;
                }
                if (tmp2cnt > 1) // i.e. multiple prompt selection then do score
                {
                        strcpy(ans, locquery);
                        return checkscore(ans, Parray, pcnt);
                }


        if (uwFlag)
                if (AskforOp())
                        return -99;
```

```
                else
                        return -1;
        else
                if (rowOrColWord)
                        {
                        strcpy(query,locquery);
                        return -99;
                        }
        }
for (i = 0; Parray[i]; i++)
        if (Parray[i] == tmpArray2[0])
                return i + 1;
}

int AskforOp()
{
int i, j ;
char *resWords[50];
int numWords ;

system("clear");
printf("Your request was not understood.\n");
printf("Would you prefer to speak to an operator or try again with a new request?\n");
fprintf(lf, "<P>Your request was not understood.<LI>");
fprintf(lf, "Would you prefer to speak to an operator or try again with a new request?\n");
fflush(lf);
fgets(query, 255, stdin);   /* accept the user input */
addWord(&queryList, query, ++numQueryList);
fprintf(lf, "<P><I> %s</I>", query);
numWords = breakStr(query, resWords);
if ( inArray(resWords,"operator",numWords))
        {
        printf("\n\nPlease wait for the operator ...\n");
        fprintf(lf,"<P>Please wait for the operator ...");
        fflush(lf);
        exit(0);
        }
if( inArray(resWords,"try",numWords) && inArray(resWords,"again",numWords))
        {
        system("clear");
        printf("Please tell us your new request\n");
        fprintf(lf, "<P>Please tell us your new request\n");
        fflush(lf);
```

```
        fgets(query, 255, stdin);   /* accept the user input */
        addWord(&queryList, query, ++numQueryList);
        }
return 1;
}

void sayOther()
{
printf("\nWould you like to hear other options?\n\n");
fprintf(lf, "<LI>Would you like to hear other options?<P>");
otheFlag = 1;
}

int checkscore(char *ans, int *Parray, int pcnt)
{
        char * resWords[50], *pmptWords[50];
        int i, j, *score, *score1;
        int numWords, numpWords, maxscore;

        score = (int *)malloc(pcnt * sizeof(int));
        score1 = (int *)malloc(pcnt * sizeof(int));
        for (i = 0; i < pcnt; i++)
                score[i] = score1[i] = 0;

        numWords = breakStr(ans, resWords);
        for (i = 0; i < pcnt; i++)
        {
                numpWords = breakStr(prompts[Parray[i]-1], pmptWords);
                for (j = 0; j < numWords; j++)
                        if (inArray(pmptWords,resWords[j], numpWords))
                                score[i]++;
        }
        for (i = maxscore = 0; i < pcnt; i++)
                maxscore = (maxscore < score[i])?score[i]:maxscore;
        for (i = j = 0; i < pcnt; i++)
                j += (score[i] == maxscore)?1:0;

        if ( j == 1)  /* single prompt selection */
        {
                for (i = 0; i < pcnt; i++)
                        if (score[i] == maxscore)
                                return i + 1;
        }
```

```
        else
        {
                for (i = 0; i < pcnt; i++)
                {
                        numpWords = breakStr(prompts[Parray[i] - 1], pmptWords);
                        score1[i] = getscore1(resWords, numWords, pmptWords, numpWords);
                }
                maxscore = 0;
                for (i = 0; i < pcnt; i++)
                        maxscore = (score1[i] > maxscore)?score1[i]:maxscore;
                for (i = j = 0; i < pcnt; i++)
                        j += (score1[i] == maxscore)?1:0;
                if ( j == 1)  /* single prompt selection */
                {
                        for (i = 0; i < pcnt; i++)
                                if (score1[i] == maxscore)
                                        return i + 1;
                }
                else
                {
                        for (i = 0; i < pcnt; i++)
                                if (score[i] != maxscore)
                                        Parray[i] = 0;
                                return -1;
                }
        }

}

int chkAfrm( char * str)
{
        int i,j, numWords;
        char * resWords[50];

        numWords = breakStr(str, resWords);
        for (i = 0; i < numWords; i ++)
                if (inArray(affrmWords, resWords[i], 3))
                                return 1;
        return 0;
}

int chkNegtn( char * str)
{
```

```
        int i,j, numWords;
        char * resWords[50];

        numWords = breakStr(str, resWords);
        for (i = 0; i < numWords; i ++)
                if (inArray(negWords, resWords[i], 2))
                        return 1;
        return 0;
}

int getscore1(char **Word1, int num1, char **Word2, int num2)
{
        int i, j, scr = 0;
        int lsmatch = 0;

        for(i = 0; i < num1; i++)
        {
                for(j = lsmatch; j < num2; j++)
                        if (!strcmp(Word1[i],Word2[j]))
                        {
                                scr++;
                                break;
                        }
                if (j < num2)
                        lsmatch = j + 1;
        }

        return scr;
}

int orderPrompts(int *InterPrompts,int  numInter)
{
        int i, j, k, l;
        int *tmpArray[2]; /* 0 - score ; 1 - level; 3 - menu order */
        int *tmpList, tmpCount;

        tmpArray[0] = (int *)malloc(numInter * sizeof(int));
        tmpArray[1] = (int *)malloc(numInter * sizeof(int));

        tmpList = (int *)malloc(numColumn * sizeof(int));
        for (i = 0;  i < numColumn; i++)
                tmpList[i] = 0;
        // get the list of keywords from queryTerms
```

```
        tmpCount = getKeyWords(queryTerms, numQuery, tmpList);

        for (i = 0; i < numInter; i++)
        {
                /* get the maxscore for the prompt */
                tmpArray[0][i] = 0;
                for (j = 0; (j < numScore) && (scoring[j][0] != InterPrompts[i]); j++);
                /* if any previous scoring present */
                if ((j < numScore) && (scoring[j][0] == InterPrompts[i]))
                {
                        // get the max score
                        for ( k = 0; k < tmpCount; k++)
                                tmpArray[0][i] = max(tmpArray[0][i] , scoring[j][tmpList[k]]);
                }
                tmpArray[1][i] = getLevel(InterPrompts[i]);
        }

        // sort the array in order of score, level and menu-order
        for ( i = 0; i < (numInter - 1); i++)
        {
                for (j = i + 1; j < numInter; j++)
                        if (!gThan(tmpArray[0][i], tmpArray[1][i], InterPrompts[i],
tmpArray[0][j], tmpArray[1][j], InterPrompts[j]))
                        {
                                swap(tmpArray[0][i], tmpArray[0][j]);
                                swap(tmpArray[1][i], tmpArray[1][j]);
                                swap(InterPrompts[i], InterPrompts[j]);
                        }
        }

        return numInter;
}

int getKeyWords(char **queryTerms, int numQuery, int *tmpList)
{
        int i, j, k, l;
        int count = 0;

for (j = 0; j < numQuery; j++)
{
        /* Check if the word is keyword */
        if ((k = inArray(columnTerms, queryTerms[j], numColumn)) != 0)
        {
```

```
                /* add in temp list only if not present */
                for (i = 0; i < count && tmpList[i] != k ; i++);
                if (i >= count)
                        tmpList[count++] = k;
                continue;
        }

        /* check if the word is Thesaurus/Learned Word */
        if ((k = inArray(rowTerms, queryTerms[j], numRow)) != 0)
        {
                /* pick-up all keywords for that word */
                for (i = 0; thesaurus[k - 1][i] != 0; i++)
                {
                        for (l = 0; l < count && tmpList[l] != thesaurus[k - 1][i] ; l++);
                                if (l >= count)
                                        tmpList[count++] = thesaurus[k - 1][i];
                }
        }

}

return count;
}

int getLevel(int pmpt)
{
        int i, k, l;

        for (i = 0; i < numMenu && menuList[i][1] != pmpt; i++);
        k = menuList[i][0];
        for (l = 0; k > 0;l++)
        {
                for (i = 0; i < numMenu && menuList[i][1] != k; i++);
                k = menuList[i][0];
        }
        return l;
}

int gThan(int a, int b, int c, int p, int q, int r)
{
        if (a > p) return 1;  // Desc order here
        if (a < p) return 0;  // Desc order here
        if (b > q) return 0;  // Asc order here
```

```
        if (b < q) return 1;  // Asc order here
        if (c > r) return 0;  // Asc order here
        return  1;  // Asc order here
}
```

formlib.c: This program contains functions for forms processing

```
#include <stdio.h>
#include <string.h>
#include "arraylib.h"

struct input {
char *Type;
char *APrompt;
char *RPrompt;
char *Name;
char *Value;
char **Choice;
int numChoice;
};

struct form {
char * name;
struct input **fields;
int numFields;
};

char * split(char * , char );

int loadForm(FILE *f, struct form *frm, char *name)
{
int j, start=0;
char buf[512];
char fname[20];

sprintf(fname,"[%s]",name);
fseek(f,SEEK_SET,0);
while(fgets(buf,512,f) != NULL) {
        j = strlen(buf);
        if (buf[j - 1] == '\n') buf[j - 1] = 0;
        if (start)
                {
```

```
                if (strlen(buf) == 0)  /* if blank line, stop reading */
                        break;
                frm->numFields++;
                frm->fields = (struct input **)realloc(frm->fields, (frm->numFields) *
sizeof(struct input *));
                frm->fields[frm->numFields-1] = (struct input *)malloc(sizeof(struct input));
                loadInput(frm->fields[frm->numFields-1], buf);
                }
        else
                if (!strcmp(fname, buf)) {
                start = 1;
                frm->name = strdup(name);
                frm->numFields=0;
                frm->fields=NULL;
                }
        }
 return start;
}

loadInput(struct input *inp, char * str)
{
char ***list, *tmpstr1, *tmpstr2;
int i, j, len;

inp->Type = inp->APrompt = inp->RPrompt = inp->Name = inp->Value = NULL;
list = (char ***)malloc(2 * sizeof(char **));
list[0] = (char **)malloc(2 * sizeof(char *));
list[1] = (char **)malloc(2 * sizeof(char *));
list[0][0] = str;
for(i=0;(list[i+1][0] = split(list[i][0],':'))!=NULL;i++)
        {
        list[i][1] = split(list[i][0],'=');
        list = (char ***)realloc(list,(i+3)*sizeof(char**));
        list[i+2] = (char **)malloc(2 * sizeof(char *));
        }
        list[i][1] = split(list[i][0],'=');
len = i + 1;
for(i=0; i <len;i++)
        {
        if (!strcmp("Type",list[i][0]))
                mystrcp(&inp->Type,list[i][1]);
        if (!strcmp("APrompt",list[i][0]))
                mystrcp(&inp->APrompt,list[i][1]);
```

```
        if (!strcmp("RPrompt",list[i][0]))
                mystrcp(&inp->RPrompt,list[i][1]);
        if (!strcmp("Name",list[i][0]))
                mystrcp(&inp->Name,list[i][1]);
        if (!strcmp("Value",list[i][0]))
                mystrcp(&inp->Value,list[i][1]);
        if (!strcmp("Choice",list[i][0]))
                {
                mystrcp(&tmpstr1, list[i][1]);
                tmpstr2 = tmpstr1;
                inp->Choice = NULL;
                inp->numChoice=0;
                for(j=0;tmpstr1[j];j++)
                        {
                        if (tmpstr1[j]==',')
                                {
                                tmpstr1[j]=0;
                                inp->Choice = (char **)realloc(inp->Choice,(inp-
>numChoice+1)*sizeof(char *));
                                inp->Choice[inp->numChoice++] = strdup(tmpstr2);
                                allTrim(inp->Choice[inp->numChoice-1]);
                                tmpstr2=tmpstr1+j+1;
                                }
                        }
                        inp->Choice = (char **)realloc(inp->Choice,(inp-
>numChoice+1)*sizeof(char *));
                        inp->Choice[inp->numChoice++] = strdup(tmpstr2);
                        allTrim(inp->Choice[inp->numChoice-1]);
                }
        }
}

mystrcp(char **str1, char *str2)
{
int len, i, j;

len = strlen(str2);
if(str2[0]=="" && str2[len-1]=="") // i.e. quoted string;
        for (i = str2[--len] = 0; (str2[i] = str2[i + 1]); i++);
*str1 = (strlen(str2)==0)?NULL:strdup(str2);
}

char * split(char * str, char dlm)
```

```
{
int i;
for (i = 0; str[i]; i++)
        if (str[i] == dlm)
                {
                str[i] = 0;
                return str + i + 1;
                }
return NULL;
}

acceptForm(struct form *frm)
{
int i;
char ans[256];
struct input cnfm;

cnfm.Type = "MChoice";
cnfm.APrompt = strdup("Is this information correct?");
cnfm.numChoice = 4;
cnfm.Choice = (char **)malloc(2 * sizeof(char *));
cnfm.Choice[0] = strdup("no");
cnfm.Choice[1] = strdup("yes");
cnfm.Choice[2] = strdup("right");
cnfm.Choice[3] = strdup("correct");
cnfm.Value = NULL;

system("clear");
printf("\n");
for(i = 0; i<frm->numFields; i++)
        {
        if (!strcmp(frm->fields[i]->Type,"Say"))
                sayText(frm->fields[i]);
        if (frm->fields[i]->Value != NULL)
                continue;
        if (!strcmp(frm->fields[i]->Type,"AcceptResponse"))
                getText(frm->fields[i]);
        if (!strcmp(frm->fields[i]->Type,"MChoice"))
                getChoice(frm->fields[i]);
        }
while (1)
        {
        system("clear");
```

```
        printf("\n");
        for(i = 0; i<frm->numFields; i++)
                {
                if (!strcmp(frm->fields[i]->Type,"AcceptResponse"))
                        sayText(frm->fields[i]);
                if (!strcmp(frm->fields[i]->Type,"MChoice"))
                        sayText(frm->fields[i]);
                }
        printf("\n");
        getChoice(&cnfm);
        if (strcmp(cnfm.Value,"no"))
                return 1;
        system("clear");
        printf("\n");
        for(i = 0; i<frm->numFields; i++)
                {
                if (!strcmp(frm->fields[i]->Type,"AcceptResponse"))
                        getText(frm->fields[i]);
                if (!strcmp(frm->fields[i]->Type,"MChoice"))
                        getChoice(frm->fields[i]);
                }
        }
}

getText(struct input * inp)
{
char buf[256];
printf("\n%s\n\n",inp->APrompt);
fgets(buf,255,stdin);
allTrim(buf);
inp->Value = strdup(buf);
}

sayText(struct input * inp)
{
if (inp->RPrompt != NULL)
        printf("%s",inp->RPrompt);
if (inp->Value != NULL)
        printf("%s",inp->Value);
if (inp->RPrompt != NULL || inp->Value != NULL)
printf(".\n");
}
```

```
fillForm(struct form * frm, char ** Array, int arrCount)
{
int i, j, wrdCount = 0, tmpCount = 0;
char **wordList = NULL;
char  *tmparray[50];
for(i = 0; i < arrCount; i++)
        {
        tmpCount = breakStr(Array[i], tmparray);
        wrdCount = mergeArray(&wordList,tmparray, wrdCount, tmpCount);
        }
wrdCount = processArray(wordList, wrdCount, 1);
for(i = 0; i < frm->numFields; i++)
        if(!strcmp(frm->fields[i]->Type,"MChoice"))
                selectValue(frm->fields[i], wordList, wrdCount);
}

int selectValue(struct input * inp, char **array, int arrCount)
{
int i, j, *score;
char *tmparray[20] ;
int max, maxcount, tmpCount;

score = (int *) malloc(inp->numChoice * sizeof(int));
for (i = 0; i < inp->numChoice; i++)
        {
        score[i] = 0;
        tmpCount = breakStr(inp->Choice[i], tmparray);
        if (tmpCount > 1) // Basically to avoid filtering of 'yes', 'no' etc
                filterStopWords(tmparray, tmpCount);
        tmpCount = processArray(tmparray, tmpCount, 0);
        for(j = 0; j < tmpCount; j++)
                if (inArray(array, tmparray[j], arrCount))
                        score[i]++;
        }
for(i = max = 0; i < inp->numChoice; i++)
        if (score[i] > max) max = score[i];
for(i = maxcount = 0; i < inp->numChoice; i++)
        if (score[i] == max) maxcount++;
if (maxcount != 1)
        return 0;
for(i = 0; i < inp->numChoice; i++)
        if (score[i] == max)
                {
```

```
                inp->Value = strdup(inp->Choice[i]);
                break;
                }
return 1;
}

processForm (struct form *frm)
{
int i, j;
char *formType = NULL, *formAction = NULL;

for(i = 0; i<frm->numFields; i++)
        {
        if (frm->fields[i]->Value == NULL)
                continue;
        if (!strcmp(frm->fields[i]->Type,"FormType"))
                formType = strdup(frm->fields[i]->Value);
        if (!strcmp(frm->fields[i]->Type,"FormAction"))
                formAction = strdup(frm->fields[i]->Value);
        }

// If not defined the form type use 'AcceptFrom' as default.
if (formType == NULL)
        formType = strdup("AcceptForm");

if (!strcmp(formType,"AcceptForm"))
        j = acceptForm(frm);

if (!strcmp(formType,"ResponseForm"))
        j = responseForm(frm);

if (j != 0 && formAction != NULL)
        performAction(frm, formAction);
}

responseForm(struct form *frm)
{
int i;

system("clear");
printf("\n");
for (i = 0; i < frm->numFields; i++)
        {
```

```
        if (strcmp("Say",frm->fields[i]->Type))
                continue;
        sayText(frm->fields[i]);
        }
printf("\n");
}

getChoice(struct input * inp)
{
char buf[256], *tmparray[50];
int tmpCount;

while (1)
        {
        printf("\n%s\n\n", inp->APrompt);
        fgets(buf,255,stdin);
        tmpCount = breakStr(buf, tmparray);
        if (tmpCount > 1) // Basically to avoid filtering of 'yes', 'no' etc
                filterStopWords(tmparray, tmpCount);
        tmpCount = processArray(tmparray, tmpCount, 0);
        if (selectValue(inp, tmparray, tmpCount))
                        return;
        }
}

performAction(struct form *frm, char *action)
{
struct form f;
char * cmd = NULL;
char buf [256];
int i, j,len1, len2;
FILE *pd;

sprintf(buf, "%s <<EOD\n", action);
cmd = strdup(buf);
for (i = 0; i < frm->numFields; i++)
        {
        if (frm->fields[i]->Name == NULL)
                continue;
        sprintf(buf,"%s=%c%s%c\n",frm->fields[i]->Name,"",frm->fields[i]->Value,"");
        len1 = strlen(buf);
        len2 = strlen(cmd);
        cmd = (char *) realloc(cmd, (len1 + len2 + 1) * sizeof(char));
```

```
        strcat(cmd, buf);
        }
sprintf(buf,"EOD\n");
len1 = strlen(buf);
len2 = strlen(cmd);
cmd = (char *) realloc(cmd, (len1 + len2 + 1) * sizeof(char));
strcat(cmd, buf);
if ((pd = popen(cmd, "r")) == NULL)
        {
        fprintf(stderr, "Error in command execution\n");
        exit(1);
        }
f.name = NULL;
f.numFields = 0;
f.fields = NULL;
while ((fgets(buf, 255, pd)!= NULL))
        {
        j = strlen(buf);
        if (buf[j - 1] == '\n') buf[j - 1] = 0;
        if (strlen(buf) == 0)  /* if blank line, stop reading */
                continue;
        f.numFields++;
        f.fields = (struct input **)realloc(f.fields, (f.numFields) * sizeof(struct input *));
        f.fields[f.numFields-1] = (struct input *)malloc(sizeof(struct input));
        loadInput(f.fields[f.numFields-1], buf);
        }
pclose(pd);
processForm(&f);
}
```

HEADER FILES (C)

globalvar.h: Header file for global variables

```
extern FILE *webDoc, *phoneDoc;
extern int numColumn,  numRow, numIndex, numMenu;
extern int startPoint, eofFlag, topValues;
extern char **rowTerms, **columnTerms, **prompts, **stopWords;
extern double **matrix, **cosine;
float phoneThreshold, webThreshold;
extern int **indexList, **menuList, **thesaurus;
```

```
extern int numStopWord, numOrgRow;
void stemArray(char **list, int arrayLen);
extern int numForms, numPF;
int stemWord(char *);
extern char ***Fprompts;
extern int numForms, numPF;
extern struct form **formlist;
```

process.h: Header file declaring functions in process.c

```
/*****************************************************************************\
* Process.h:                                                                  *
\*****************************************************************************/

int processFile(char *filename, char ***cArray, float threshold);
void loadStopWords( char * filename) ;
// int allTrim( char *str);
void fillIndex();
void updateThesaurus( char *str, int pmpt);
void createMatrix(char * filename);
// int readPara(FILE *fp );
// int wordsInPara (FILE *fp);
void calcCosine();
int eraseZeroes();
void createThesaurus();
// void floatSort(int *colnum, float *tmpcos, int numRow);
void saveData(char *filenm);
```

arraylib.h: Header file declaring functions in arraylib.c

```
/*****************************************************************************\
* ArrayLib.h                                                                  *
\*****************************************************************************/
int fetchWord(FILE *f, char * wrd);
int inArray(char **array, char *word, int length);
int removeNulls(char **strarray, int numWords);
int mergeArray(char ***Array1, char **Array2, int numArray1, int numArray2);
int readValues(char *str, char **array);
void sortArray(char *allwords[], int numwords);
int loadPrompts(char *filename);
void loadStopWords(char *);
```

```
FILE * fileOpen( char *filename, char *mode);
void addWord(char *** cArray, char * word, int c);
int breakStr(char * str, char **strarray);
void filterStopWords(char ** strarray, int numWords);
void filterDuplicates(char ** strarray, int numWords);
int loadFormsList( char *filename);
int loadForms(char * filename);
int allTrim( char *str);
int createArray(char *, char **);
int processArray( char **, int, int);
```

forms.h: Header file declaring functions in formlib.c

```
#include <stdio.h>

extern struct input {
char *Type;
char *APrompt;
char *RPrompt;
char *Name;
char *Value;
char **Choice;
int numChoice;
} a;

extern struct form {
char * name;
struct input **fields;
int numFields;
} b;

char * split(char * , char );
int loadForm(FILE *, struct form *, char *);

void loadInput(struct input *, char * );
void mystrcp(char **, char *);
void dumpInput(FILE *, struct input *);
void dumpForm(FILE *, struct form *);
void acceptForm(struct form *);
void getText(struct input * );
void fillForm(struct form * frm, char ** Array, int arrCount);
```

MAKE FILE

makefile: Makefile for compiling the source code.

```
all: t d demorun
t: main.o process.o arraylib.o stemlib.o formlib.o
        cc -g main.o process.o arraylib.o stemlib.o formlib.o -o t -lm
d: dialog.o interactive.o arraylib.o stemlib.o formlib.o
        cc -g dialog.o interactive.o arraylib.o stemlib.o formlib.o -o d
demorun: demorun.c
        cc demorun.c -o demorun
main.o: main.c process.h arraylib.h forms.h
        cc -c -g main.c
process.o: process.c globalvar.h forms.h
        cc -c -g process.c
arraylib.o: arraylib.c globalvar.h
        cc -c -g arraylib.c
dialog.o: dialog.c  arraylib.h
        cc -c -g dialog.c
interactive.o: interactive.c globalvar.h
        cc -c -g interactive.c
stemlib.o: stemlib.c
        cc -c -g stemlib.c
formlib.o: formlib.c
        cc -c -g formlib.c
clean:
        rm -f *.o t d core demorun
bkup: clean
        tar cvzf ../stem`date "+%d%m"`.tgz .
```

PARAMETER FILES

t.ini: This file contains parameters required for program 't'

```
pdoc p # phonedoc
wdoc w # webdoc
sdoc s # stopwords
fdoc f # forms
xdoc x # link of forms & prompts
cfg z.cfg # config file
pt 0.02 # phoneThreshold
wt 0.0006 # webThreshold
tv 5 # topValues for cosine
```

d.ini: This file contains parameters required for program 'd'

```
cfg t.cfg # config file
lcfg l.cfg # learn file
sdoc s # stopwords
fdoc f # forms
xdoc x # x
minprompt 2 # minimum no of prompts
timeout 30 # timeout secs for other options
```

DATA FILES

p: Document 'p'

```
Are you calling about subscriptions?
        Would you like to order a subscription?
        Would you like to pay your subscription fees?
        Would you like to give a gift subscription?
        Would you like to change your address or change any other information?
        Do you have any billing enquiries or concerns?
                Would you like information about your account balance or your payments?
                Would you like to speak to a customer care representative?
        Would you like to temporarily suspend your delivery?
Is there a problem with your paper or delivery?
        Did you miss today's paper?
        Did you miss yesterday's newspaper and would you like credit for yesterday?
        Did you receive a wet paper?
Would you like information about the New Herald website?
        Would you like to obtain your New Herald website password?
        The website address is www.newherald.com. Would you like any other information about
the website?
Are you calling about advertisements?
        Would you like to advertise in the New Herald?
                Is it a classified ad?
                Is it a full-page, half-page, or quarter-page ad?
        Would you like to place an ad?
                Is it a classified ad?
                Is it a full-page, half-page, or quarter-page ad?
Are you calling about something else?
        Would you like to write to the New Herald?
                Would you like to submit an article to the op-ed page?
                        Please email your article to oped@newherald.com.
                Would you like to send a letter to the editor?
```

Please email your letter to letters@newherald.com.
Would you like to work for the New Herald?
Would you like to write for the New Herald?
Would you like to work for the editorial division or for the administrative division?

w: Document 'w'

Now, it's easier than ever to manage your Herald. Welcome to The New Herald Subscription & Customer Care Web site. You expect all the news that's fit to print in each issue of The New Herald. And you can expect responsive, round-the-clock on-line customer care that allows you to review and update your delivery and billing information, stop delivery of your newspaper when you're away, discover special promotions and notify us of any questions or comments you have. And if you're not a subscriber, browse our Web site and consider subscribing to home delivery. Please enjoy your visit.

With convenient home delivery, you will be sure to receive all the wit, the wisdom, the news, the views offered in every issue of The New Herald. And, through this special offer, you will get 50% off the regular rate for the first eight weeks. To subscribe enter your ZIP code below and submit.

ZIP Code:

Expect the World Around the Clock

We are pleased to offer our subscribers instant, 24-hour on-line customer care to meet your service needs. Now, it's easier than ever to order home delivery, review your bill, or change your service -- and to find out about special customer benefits and promotions.

Take Our Survey

Help us provide you with the highest quality customer service. This short survey asks for vital information about you and your service needs. The New Herald may perform statistical analysis of reader interests to identify ways to improve our services and products to better meet the needs of our subscribers. Personal information about you as an individual subscriber will not be provided to any third party. Our privacy policy is posted online to disclose our guidelines for the use of customer information.

You can handle most of your subscription service requests online, including:

* Suspending your delivery while you're away

* Reporting missed deliveries
* Checking the status of your account
* Checking your billing history
* Changing your delivery or billing address
* Changing your method of payment

To subscribe

Customer care:

Account Summary

Update Account

Activity History

Billing History

Paper not received

Suspend delivery

Complaints

Order Home Delivery at 50% Off (US Customers Only)

With convenient home delivery, you can start each day with all the news, the views, the wit and the wisdom you expect from The New Herald. And, through this special offer, you can save 50% on the first eight weeks when you order today. It's a smart, easy way to keep up with The Herald.

This offer expires December 31, 2001 and is valid in areas served by The New Herald' delivery service. Subscribers who have had Herald home delivery within the past 90 days are not eligible for this introductory offer. To subscribe enter your ZIP code below and submit.

GIFT SUBSCRIPTION OFFER

There's no present like The Herald. And, when you order a gift subscription of 12-week home delivery of The New Herald, you'll save 50% on the regular rate. Hurry. This offer expires

December 31, 2001. To order, enter the ZIP code of the gift recipient below and submit.

Order Home Delivery at 50% Off (US Customers Only)

With convenient home delivery, you can start each day with all the news, the views, the wit and the wisdom you expect from The New Herald. And, through this special offer, you can save 50% on the first eight weeks when you order today. It's a smart, easy way to keep up with The Herald.

This offer expires December 31, 2001 and is valid in areas served by The New Herald' delivery service. Subscribers who have had Herald home delivery within the past 90 days are not eligible for this introductory offer. To subscribe enter your ZIP code below and submit.

ZIP Code:

GIFT SUBSCRIPTION OFFER

There's no present like The Herald. And, when you order a gift subscription of 12-week home delivery of The New Herald, you'll save 50% on the regular rate. Hurry. This offer expires December 31, 2001. To order, enter the ZIP code of the gift recipient below and submit.

ZIP Code:

LARGE TYPE WEEKLY

Developed especially for people with low vision, The New Herald Large Type Weekly offers a select package of the week's news printed in 16-point type--about twice the size of the regular type size. With its updated, color-enhanced design, The Large Type Weekly is a striking--and clearly readable way to enjoy The New Herald. A mail subscription of The New Herald Large Type Weekly makes a great gift for yourself or someone you care for.

To order, select a country/region below and submit.

Order Home Delivery at 50% Off (US Customers Only)

With convenient home delivery, you can start each day with all the news, the views, the wit and the wisdom you expect from The New Herald. And, through this special offer, you can save 50% on the first eight weeks when you order today. It's a smart, easy way to keep up with The Herald.

This offer expires December 31, 2001 and is valid in areas served by The New Herald' delivery service. Subscribers who have had Herald home delivery within the past 90 days are not eligible for this introductory offer. To subscribe enter your ZIP code below and submit.

PATENT
Docket No.: 4428-4001

ZIP Code:

GIFT SUBSCRIPTION OFFER

There's no present like The Herald. And, when you order a gift subscription of 12-week home delivery of The New Herald, you'll save 50% on the regular rate. Hurry. This offer expires December 31, 2001. To order, enter the ZIP code of the gift recipient below and submit.

ZIP Code:

LARGE TYPE WEEKLY

Developed especially for people with low vision, The New Herald Large Type Weekly offers a select package of the week's news printed in 16-point type--about twice the size of the regular type size. With its updated, color-enhanced design, The Large Type Weekly is a striking--and clearly readable way to enjoy The New Herald. A mail subscription of The New Herald Large Type Weekly makes a great gift for yourself or someone you care for.

To order, select a country/region below and submit.

Country:

The New Herald Book Review

Get a head start on the latest book reviews, the acclaimed New Herald Best Sellers lists and everything new and noteworthy in the literary world. When you order a mail subscription to The New Herald Sunday Book Review, you'll receive it days in advance of the Sunday New Herald.

To order, select a country/region below and submit.

Country:

To Subscribe - Foreign Mail Subscriptions

Stay informed with all the news in the United States and throughout the world, including sharp analyses, reports and reviews from the world of business, sports and the arts. Discerning readers across the country and around the globe depend on The Herald for inside revelations, outside opinions, all sides of the story. Now you can too -- with the convenience of home delivery by mail. Order now.

To change your address, method of billing or any of the account information featured below, please enter the updated information in the appropriate box. Once you have completed all

information requested, please click Submit at the bottom of the page. (Please note: It is necessary for all bold fields to be filled out to process your updated information.)

Your billing and payment history provides an at-a-glance summary of your account and makes it simple to check on your balance, last payment and new charges. Recent invoices are listed below for your review. Questions may be submitted to our customer care billing representatives by going to the Complaints page. Please be sure to indicate Billing Inquiry as the nature of your complaint.

Did you miss a paper? The New Herald is committed to making sure you get every issue you've ordered. If you did not receive your paper or any of its sections, simply select one of the following redelivery options so that we may deliver one to you.

Please note: Only same day redelivery is available on-line; you must submit this information on the same day on which your paper or section was to be delivered. For credits on past issues, please phone our customer care representatives at 1(800) 555-9876.

I would like to have today's paper delivered tomorrow. Please credit my account for today's missed paper.

Did you miss a paper? The New Herald is committed to making sure you get every issue you've ordered. If you did not receive your paper or any of its sections, simply select one of the following redelivery options so that we may deliver one to you.

Please note: Only same day redelivery is available on-line; you must submit this information on the same day on which your paper or section was to be delivered. For credits on past issues, please phone our customer care representatives at 1(800) 555-9876.

PAPER NOT RECEIVED

I did not receive today's paper.

ACCOUNT NUMBER: 060095544

Reason:

Select One: I would like to have today's paper delivered tomorrow.

Please credit my account for today's missed paper.

SECTIONS NOT RECEIVED

PATENT
Docket No.: 4428-4001

I received today's paper with the section(s) checked below missing

Sections not received: ARTS & LEISURE BUSINESS

DINING IN/OUT MAIN NEWS SEC

METROPOLITAN SPORTS

Select One: I would like to have today's paper delivered tomorrow.

Please credit my account for today's missed paper.

As a newspaper home delivery subscriber, you may suspend your service for any amount of time. When you suspend your home delivery service, you may elect to take part in our vacation donation program (see description below). Please indicate your suspension

and restart dates below:

ACCOUNT NUMBER: 060095544

SUSPEND/RESUME

Suspend:

Resume:

Vacation Donation Program

During your next vacation, sit back, relax -- and at the same time enrich your community. Through The New Herald Newspaper in Education program, you can donate your subscription to students for the time period in which you will be out of town. For each copy you donate, at least two students will receive their own copy of The New Herald. To donate your vacation copies, please indicate below.

Choose One: Donate the vacation period papers to local schools through the Newspapers in Education program. Credit my account for the period of my vacation. SUSPEND/RESUME 2

Suspend:

Resume:

PATENT
Docket No.: 4428-4001

Choose One: Donate the vacation period papers to local schools through the Newspapers in Education program. Credit my account for the period of my vacation. SUSPEND/RESUME 3

Suspend:

Resume:

Choose One: Donate the vacation period papers to local schools through the Newspapers in Education program. Credit my account for the period of my vacation.

To best provide you with responsive, accessible customer service, we encourage your comments and suggestions. Please let us know about any dissatisfaction you may have with your delivery or billing service. Customers who have not received a paper can order another paper or receive credit for today by clicking here.

For all other subscription concerns, please use the form below to send us an email indicating the nature of your complaint and explaining how we may help you. A customer care representative will respond to your request within 24 hours to the email address provided on this form.

s: Document 's'

a
also
an
am
and
any
as
at
be
but
by
can
could
do
for
from
go
got
have
he
her

here
his
how
i
if
in
into
it
its
m
my
of
on
or
our
say
she
that
the
their
there
therefore
they
there
their
this
these
those
through
to
until
we
what
when
where
which
while
who
with
would
you
your
about

[redacted]

across
are
around
did
during
each
ever
every
had
must
no
now
only
other
so
than
too
was
within
everything
is
like
want
please
s
cannot
then
d
ll
me
paper
going
having
ve
been
being
some
speak
know

[redacted]

f: Document 'f'

[FORM1]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="Please tell us your account number.":Value="":RPrompt="Your account number is "
Type="AcceptResponse":Name="date":APrompt="When would you like to start suspending the paper?":Value="":RPrompt="The delivery will stop on "
Type="MChoice":Name="Duration":APrompt="Would you like to suspend the paper for one month, two months, or three months?":Choice="one month,two months,three months":Value="":RPrompt="The delivery will be suspended for "
Type="FormAction":Value="./susp_deli"

[FORM2]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="Name":APrompt="Please tell us your name.":Value="":RPrompt="Your name is "
Type="AcceptResponse":Name="Address":APrompt="What city do you live in?":Value="":RPrompt="You live in "
Type="MChoice":Name="SubType":APrompt="Would you like the newspaper daily or just the Sunday paper?":Value="":Choice="a daily newspaper,the Sunday newspaper":RPrompt="You have opted for "
Type="MChoice":Name="SubPrd":APrompt="Would you like a half-yearly or annual subscription?":Value="":Choice="a half-yearly subscription,an annual subscription":RPrompt="You have chosen "
Type="FormAction":Value="./add_acct"

[FORM3]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="What is your account number?":Value="":RPrompt="Your account number is "
Type="FormAction":Value="./acct_info"

[FORM4]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="What is your account number?":Value="":RPrompt="Your account number is "
Type="FormAction":Value="./get_pymt"

[FORM5]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="Name":APrompt="Whom would you like to gift the subscription to?":Value="":RPrompt="You are gifting this subscription to "

Type="AcceptResponse":Name="Address":APrompt="In which city does the person live?":Value="":RPrompt="The person lives in "
Type="MChoice":Name="SubType":APrompt="Would you like to give a daily newspaper or just the Sunday paper?":Value="":Choice="a daily newspaper,the Sunday newspaper":RPrompt="You have opted for "
Type="MChoice":Name="SubPrd":APrompt="Would you like a half-yearly or annual subscription?":Value="":Choice="a half-yearly subscription,an annual subscription":RPrompt="You have chosen "
Type="FormAction":Value="./add_acct"

[FORM6]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="What is your account number?":Value="":RPrompt="Your account number is "
Type="MChoice":Name="preference":APrompt="Would you like the newspaper or would you prefer credit for it?":Value="":Choice="the newspaper,credit":RPrompt="You prefer "
Type="FormAction":Value="./prefer"

[FORM7]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="What is your account number?":Value="":RPrompt="Your account number is "
Type="Hidden":Name="preference":Value="credit"
Type="FormAction":Value="./prefer"

[FORM8]
Type="FormType":Value="AcceptForm"
Type="AcceptResponse":Name="AcctNo":APrompt="What is your account number?":Value="":RPrompt="Your account number is "
Type="FormAction":Value="./chg_acct"

x: Document 'x'

FORM1:Would you like to temporarily suspend your delivery?
FORM2:Would you like to order a subscription?
FORM3:Would you like information about your account balance or your payments?
FORM4:Would you like to pay your subscription fees?
FORM5:Would you like to give a gift subscription?
FORM6:Did you miss today's paper?
FORM6:Did you receive a wet paper?
FORM7:Did you miss yesterday's newspaper and would you like credit for yesterday?
FORM8:Would you like to change your address or change any other information?

a: Datafile 'a' contains data about subscription
1|1|2|01-01-2002|365|315|01-01-2002|50|||Frege|Jena
2|2|2|01-02-2002|52|32|01-02-2002|20|||Russell|Cambridge
3|2|2|01-02-2002|52|32|01-02-2002|20|||Wittgenstein|Vienna
4|1|2|01-04-2002|364|314|01-04-2002|50|||Austin|Oxford
5|1|2|01-05-2002|365|264|01-05-2002|100|||Grice|Berkeley
6|1|1|01-06-2002|180|49|01-06-2002|130|||Parikh|New York

CONFIGURATION FILES

t.cfg: Thesaurus configuration file. This is generated by program 't'

[PROMPTS]
Are you calling about subscriptions?
Would you like to order a subscription?
Would you like to pay your subscription fees?
Would you like to give a gift subscription?
Would you like to change your address or change any other information?
Do you have any billing enquiries or concerns?
Would you like information about your account balance or your payments?
Would you like to speak to a customer care representative?
Would you like to temporarily suspend your delivery?
Is there a problem with your paper or delivery?
Did you miss today's paper?
Did you miss yesterday's newspaper and would you like credit for yesterday?
Did you receive a wet paper?
Would you like information about the New Herald website?
Would you like to obtain your New Herald website password?
The website address is www.newherald.com. Would you like any other information about the website?
Are you calling about advertisements?
Would you like to advertise in the New Herald?
Is it a classified ad?
Is it a full-page, half-page, or quarter-page ad?
Would you like to place an ad?
Are you calling about something else?
Would you like to write to the New Herald?
Would you like to submit an article to the op-ed page?
Please email your article to oped@newherald.com.
Would you like to send a letter to the editor?
Please email your letter to letters@newherald.com.
Would you like to work for the New Herald?

[redacted]

Would you like to write for the New Herald?
Would you like to work for the editorial division or for the administrative division?

[MENUTREE]
0,1,0
1,2,99
1,3,99
1,4,99
1,5,99
1,6,0
6,7,99
6,8,99
1,9,99
0,10,0
10,11,99
10,12,99
10,13,99
0,14,0
14,15,99
14,16,99
0,17,0
17,18,0
18,19,99
18,20,99
17,21,0
21,19,99
21,20,99
0,22,0
22,23,0
23,24,25
24,25,100
23,26,27
26,27,100
22,28,0
28,29,99
28,30,99

[INDEX]
account 7,
ad 19,20,21,
address 5,16,
administr 30,
advertis 17,18,

annual 2,4,
articl 24,25,
balanc 7,
bill 6,
call 1,17,22,
care 8,
chang 5,
classifi 19,
com 16,25,27,
concern 6,
credit 12,11,13,
custom 8,
daili 2,4,
deliveri 9,10,
divis 30,
ed 24,
editor 26,
editori 30,
els 22,
email 25,27,
enquiri 6,
fee 3,
full 20,
gift 4,
give 4,
half 20,2,4,
inform 5,7,14,16,
letter 26,27,
miss 11,12,
month 9,
newspap 12,2,4,11,13,
nytim 16,25,27,
obtain 15,
on 9,
op 24,25,
order 2,
pai 3,
password 15,
payment 7,
place 21,
problem 10,
quarter 20,
receiv 13,

repres 8,
send 26,
someth 22,
submit 24,
subscript 1,2,3,4,
sundai 2,4,
suspend 9,
temporarili 9,
three 9,
todai 11,
two 9,
websit 14,15,16,
wet 13,
work 28,30,
write 23,29,
www 16,
yearli 2,4,
yesterdai 12,

[THESAURUS]
access 58,41,48,19,
acclaim 54,48,53,41,
account 16,36,39,34,
address 12,9,15,50,25,
advanc 54,48,53,41,
allow 11,17,9,32,
amount 55,36,19,
analys 19,41,32,52,
analysi 32,17,49,
anoth 58,41,48,19,
appropri 12,3,9,32,
ask 32,17,49,
back 59,48,36,53,
balanc 44,49,11,9,
benefit 9,12,17,19,
better 32,17,49,
bill 12,3,44,
bold 12,3,9,32,
bottom 12,3,9,32,
box 12,3,9,32,
brows 11,17,9,32,
care 49,17,8,32,
chang 9,3,44,19,

charg 8,44,49,11,9,
commun 59,48,36,53,
complet 12,3,9,32,
concern 50,25,49,11,3,
consid 11,17,9,32,
credit 1,36,34,39,58,
custom 11,49,32,
deliveri 41,52,
depend 19,41,32,52,
descript 55,36,19,
discern 19,41,32,52,
disclos 32,17,49,
discov 11,17,9,32,
dissatisfact 58,41,48,19,
elect 55,36,19,
email 15,50,49,3,11,17,
encourag 58,41,48,19,
enrich 59,48,36,53,
explain 15,50,25,49,11,3,
featur 12,3,9,32,
field 12,3,9,32,
fill 12,3,9,32,
find 9,12,17,19,
fit 11,17,9,32,
foreign 53,29,59,54,
gift 53,41,52,19,
glanc 8,44,49,11,9,
globe 19,41,32,52,
guidelin 32,17,49,
handl 53,29,59,54,
head 54,48,53,41,
highest 32,17,49,
identifi 32,17,49,
improv 32,17,49,
individu 32,17,49,
inform 17,49,11,
inquiri 8,44,49,11,9,
insid 19,41,32,52,
instant 9,12,17,19,
interest 32,17,49,
invoic 8,44,49,11,9,
last 8,44,49,11,9,
latest 54,48,53,41,

least 59,48,36,53,
leisur 48,59,54,34,58,
let 58,41,48,19,
literari 54,48,53,41,
manag 11,17,9,32,
miss 58,16,39,1,48,
most 53,29,59,54,
necessari 12,3,9,32,
newspap 16,55,39,1,
next 59,48,36,53,
noteworthi 54,48,53,41,
notifi 11,17,9,32,
on 36,16,34,1,58,
onc 12,3,9,32,
opinion 19,41,32,52,
order 19,52,29,
outsid 19,41,32,52,
own 59,48,36,53,
part 55,36,19,
parti 32,17,49,
payment 8,9,12,49,11,
perform 32,17,49,
person 32,17,49,
pleas 9,12,17,19,
polici 32,17,49,
post 32,17,49,
privaci 32,17,49,
process 12,3,9,32,
product 32,17,49,
qualiti 32,17,49,
receiv 34,59,
recent 8,44,49,11,9,
relax 59,48,36,53,
repres 11,17,8,15,50,32,
respond 15,50,25,49,11,3,
revel 19,41,32,52,
round 11,17,9,32,
see 55,36,19,
seller 54,48,53,41,
send 15,25,49,11,3,
sharp 19,41,32,52,
short 32,17,49,
side 19,41,32,52,

simpl 8,44,49,11,9,
sit 59,48,36,53,
stai 19,41,32,52,
state 19,41,32,52,
statist 32,17,49,
statu 1,16,8,36,
stop 11,17,9,32,
stori 19,41,32,52,
submit 19,41,49,11,17,
subscript 29,11,
suggest 58,41,48,19,
sundai 48,53,41,59,
suspend 36,39,16,19,
suspens 55,36,19,
third 32,17,49,
throughout 19,41,32,52,
todai 34,16,48,41,
town 59,48,36,53,
two 48,36,53,
unit 19,41,32,52,
visit 11,17,9,32,
vital 32,17,49,
welcom 11,17,9,32,

l.cfg: Thesaurus learn file. newly learned meanings are stored in this file

[EXT-THESAURUS]

[SCORING]
2,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,
0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
13,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

```
cnt=`grep -c "^$acctno|" a`
if [ $cnt -eq 0 ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value=""'
        exit 0
fi
line=` grep "^$acctno|" a`
echo 'Type="FormType":Value="ResponseForm"'

name=`echo $line | cut -d'|' -f 11`
echo 'Type="Say":RPrompt="Your last name is ":Value="'$name'"'
city=`echo $line | cut -d'|' -f 12`
echo 'Type="Say":RPrompt="You live in ":Value="'$city'"'
sub_type=`echo $line | cut -d'|' -f 2`
if [ $sub_type -eq 1 ]
then
        sub_type="a daily newspaper"
else
        sub_type="the Sunday newspaper"
fi
echo 'Type="Say":RPrompt="You have subscribed for ":Value="'$sub_type'"'
sub_prd=`echo $line | cut -d'|' -f 3`
sdate=`echo $line | cut -d'|' -f 4`
if [ $sub_prd -eq 1 ]
then
        sub_prd="six months"
else
        sub_prd="one year"
fi
echo 'Type="Say":RPrompt="The subscription starts on '$sdate' for a period of
":Value="'$sub_prd'"'

fee=`echo $line | cut -d'|' -f 5`
echo 'Type="Say":RPrompt="The subscription fee is $":Value="'$fee'"'
bal=`echo $line | cut -d'|' -f 6`
echo 'Type="Say":RPrompt="Your balance is $":Value="'$bal'"'
pdate=`echo $line | cut -d'|' -f 7`
pymt=`echo $line | cut -d'|' -f 8`
echo 'Type="Say":RPrompt="Your last payment was $'$pymt' on ":Value="'$pdate'"'
sudate=`echo $line | cut -d'|' -f 9`
if [ "X$sudate" != "X" ]
```

```
then
        suprd=`echo $line | cut -d'|' -f 10`
        case $suprd in
        1) suprd="one month";;
        2) suprd="two months";;
        3) suprd="three months";;
        esac
        echo 'Type="Say":RPrompt="Your account is suspended from '$sudate' for
":Value="'$suprd'"'
fi
```

add_acct: Script to add new account into 'a'

```
#!/bin/sh
# arrange all the values of input into a single line
cp /dev/null /tmp/param
cut -d'=' -f 2 | sed "s/\"//g
s/ /\\\\\\\ /g" | while read aa
do
        echo -n $aa' ' >> /tmp/param
done
echo "" >>/tmp/param

# now transfer them into env variables.
read NAME CITY SUB_TYPE SUB_PRD < /tmp/param
if [ "$SUB_TYPE" = "a daily newspaper" ]
then
        SUB_TYPE=1
        FEE=182
else
        SUB_TYPE=2
        FEE=26
fi
if [ "$SUB_PRD" = "a half-yearly subscription" ]
then
        SUB_PRD=1
else
        SUB_PRD=2
        FEE=`expr $FEE \* 2`
fi
cnt=1
while true
```

```
do
        if [ "`grep -c \"^$cnt|\" a`" -ne 0 ]
        then
                cnt=`expr $cnt + 1`
                continue
        fi
        echo $cnt'|'$SUB_TYPE'|'$SUB_PRD'|'`date +%d-%m-
%Y`"|$FEE|$FEE||0|||$NAME|$CITY" >> a
        echo 'Type="FormType":Value="AcceptForm"'
        echo 'Type="Say":RPrompt="Your subscription request has been entered":Value=""'
        echo 'Type="Say":Name="acct_no":RPrompt="Your account number is ":Value="'$cnt'"'
        echo 'Type="Say":RPrompt="Your fee for the subscription is $":Value="'$FEE'"'
        echo 'Type="AcceptResponse":Name="payment":APrompt="Your minimum initial
payment is $25. How much would you like to pay now?":Value="":RPrompt="You have chosen
to pay $"'
        echo 'Type="FormAction":Value="./updt_pymt"'
        break
done
#rm /tmp/param
```

chg_acct: Script to generate a form to change account information

```
#!/bin/sh
acctno=`cut -d'=' -f 2 | sed s/\"//g`
#echo $acctno
cnt=`grep -c "^$acctno|" a`
if [ $cnt -eq 0 ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value=""'
        exit 0
fi
line=`grep "^$acctno|" a`
echo 'Type="FormType":Value="AcceptForm"'
echo 'Type="Hidden":Name="acctno":Value='$acctno''
#---------------------------------- Response info
name=`echo $line | cut -d'|' -f 11`
echo 'Type="Say":RPrompt="Your last name is ":Value="'$name'"'
city=`echo $line | cut -d'|' -f 12`
echo 'Type="Say":RPrompt="You live in ":Value="'$city'"'
sub_type=`echo $line | cut -d'|' -f 2`
```

```
if [ $sub_type -eq 1 ]
then
        sub_type="a daily newspaper"
else
        sub_type="the Sunday newspaper"
fi
echo 'Type="Say":RPrompt="You have subscribed for ":Value="'$sub_type'"'
sub_prd=`echo $line | cut -d'|' -f 3`
sdate=`echo $line | cut -d'|' -f 4`
if [ $sub_prd -eq 1 ]
then
        sub_prd="six months"
else
        sub_prd="one year"
fi
echo 'Type="Say":RPrompt="The subscription starts on '$sdate' for a period of
":Value="'$sub_prd'"'

fee=`echo $line | cut -d'|' -f 5`
echo 'Type="Say":RPrompt="The subscription fee is $":Value="'$fee'"'
bal=`echo $line | cut -d'|' -f 6`
echo 'Type="Say":RPrompt="Your balance is $":Value="'$bal'"'
pdate=`echo $line | cut -d'|' -f 7`
pymt=`echo $line | cut -d'|' -f 8`
echo 'Type="Say":RPrompt="Your last payment was $'$pymt' on ":Value="'$pdate'"'
sudate=`echo $line | cut -d'|' -f 9`
if [ "X$sudate" != "X" ]
then
        suprd=`echo $line | cut -d'|' -f 10`
        case $suprd in
        1) suprd="one month";;
        2) suprd="two months";;
        3) suprd="three months";;
        esac
        echo 'Type="Say":RPrompt="Your account is suspended from '$sudate' for
":Value="'$suprd'"'
fi
#-------------------------------------------------------------
echo 'Type="AcceptResponse":Name="Name":APrompt="What name would you like to
use?":Value="":RPrompt="The name you would like to use is "'
echo 'Type="AcceptResponse":Name="Address":APrompt="What city would you like the
newspaper sent to?":Value="":RPrompt="The city you would like the newspaper sent to is "'
```

```
echo 'Type="MChoice":Name="SubType":APrompt="Would you like the newspaper daily or
just the Sunday paper?":Value="":Choice="a daily newspaper,the Sunday
newspaper":RPrompt="You have opted for "'
echo 'Type="MChoice":Name="SubPrd":APrompt="Would you like a half-yearly or annual
subscription?":Value="":Choice="a half-yearly subscription,an annual
subscription":RPrompt="You have chosen "'
echo 'Type="FormAction":Value="./updt_acct"'
```

get_pymt: Script to generate a form to accept payment for a particular account

```
#!/bin/sh
acctno=`cut -d'=' -f 2 | sed s/\"//g`
#echo $acctno
cnt=`grep -c "^$acctno|" a`
if [ $cnt -eq 0 ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value=""'
        exit 0
fi

line=` grep "^$acctno|" a`

fee=`echo $line | cut -d'|' -f 5`
bal=`echo $line | cut -d'|' -f 6`
pdate=`echo $line | cut -d'|' -f 7`
pymt=`echo $line | cut -d'|' -f 8`
if [ $bal -le 0 ]
then
        echo 'Type="FormType":Value="ResponseForm"'
else
        echo 'Type="FormType":Value="AcceptForm"'
fi
echo 'Type="Say":RPrompt="The subscription fee is $":Value="'$fee'"'
echo 'Type="Say":RPrompt="Your last payment was $'$pymt' on":Value="'$pdate'"'
echo 'Type="Say":RPrompt="Your balance is $":Value="'$bal'"'
if [ $bal -ne 0 ]
then
        echo 'Type="Hidden":Name="acctno":Value="'$acctno'"'
        echo 'Type="AcceptResponse":Name="payment":APrompt="How much would you like
to pay now?":Value="":RPrompt="You have paid $"'
```

```
        echo 'Type="FormAction":Value="./updt_pymt"'
fi
```

updt_pymt: Script to update the data file 'a' using form information

```
#!/bin/sh
cp /dev/null /tmp/param1
cut -d'=' -f 2 | sed "s/\"//g
s/ /\\\\\\\ /g" | while read aa
do
        echo -n $aa' ' >> /tmp/param1
done
echo "" >>/tmp/param1
read acctno payment < /tmp/param1
touch /tmp/tmpa
echo "no" > /tmp/found
cat a | while read line
do
        cacno=`echo $line | cut -d'|' -f 1`
        if [ $cacno -eq $acctno ]
        then
                echo "yes" > /tmp/found
                echo -n $cacno'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 2`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 3`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 4`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 5`'|' >> /tmp/tmpa
                bal=`echo $line | cut -d'|' -f 6`
                bal=`expr $bal - $payment`
                echo -n $bal'|' >> /tmp/tmpa
                echo -n `date +%d-%m-%Y`'|' >> /tmp/tmpa
                echo -n $payment'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 9`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 10`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 11`'|' >> /tmp/tmpa
                echo `echo $line | cut -d'|' -f 12` >> /tmp/tmpa
        else
                echo $line >> /tmp/tmpa
        fi
done
mv /tmp/tmpa a
read ans < /tmp/found
```

```
if [ "$ans" = "yes" ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Thank you for the subscription":Value=""'
else
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value=""'
fi
rm /tmp/param1 /tmp/found
```

susp_deli: Script to suspend delivery for a particular account

```
#!/bin/sh
cp /dev/null /tmp/param1
cut -d'=' -f 2 | sed "s/\"//g
s/ /\\\\\\\ /g" | while read aa
do
        echo -n $aa' ' >> /tmp/param1
done
echo "" >>/tmp/param1
read acctno sdate period < /tmp/param1
if [ "$period" = "one month" ]
then
        period=1
fi
if [ "$period" = "two month" ]
then
        period=2
fi
if [ "$period" = "three months" ]
then
        period=3
fi
echo "no" > /tmp/found
touch /tmp/tmpa
cat a | while read line
do
        cacno=`echo $line | cut -d'|' -f 1`
        if [ $cacno -eq $acctno ]
        then
                echo "yes" > /tmp/found
```

```
                echo -n $cacno'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 2`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 3`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 4`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 5`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 6`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 7`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 8`'|' >> /tmp/tmpa
                echo -n $sdate'|' >> /tmp/tmpa
                echo -n $period'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 11`'|' >> /tmp/tmpa
                echo `echo $line | cut -d'|' -f 12` >> /tmp/tmpa
        else
                echo $line >> /tmp/tmpa
        fi
done
mv /tmp/tmpa a
read ans < /tmp/found
if [ "$ans" = "yes" ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Thank you. The information has been updated":Value=""'
else
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value=""'
fi

rm /tmp/param1 /tmp/found
```

updt_acct: Script to update data file 'a' with changed information

```
#!/bin/sh
# arrange all the values of input into a single line
cp /dev/null /tmp/param
cut -d'=' -f 2 | sed "s/\"//g
s/ /\\\\\\\ /g" | while read aa
do
        echo -n $aa' ' >> /tmp/param
done
echo "" >>/tmp/param
cp /dev/null /tmp/tmpa
```

```
read acctno name city sub_type sub_prd < /tmp/param
cat a | while read line
do
        cacno=`echo $line | cut -d'|' -f 1`
        if [ $cacno -eq $acctno ]
        then
                echo -n $cacno'|' >> /tmp/tmpa
                if [ "$sub_type" = "a daily newspaper" ]
                then
                        sub_type=1
                        newfee=182
                else
                        sub_type=2
                        newfee=26
                fi
                echo -n $sub_type'|' >> /tmp/tmpa
                if [ "$sub_prd" = "a half-yearly subscription" ]
                then
                        sub_prd=1
                else
                        sub_prd=2
                        newfee=`expr $newfee \* 2`
                fi
                echo -n $sub_prd'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 4`'|' >> /tmp/tmpa
                echo -n $newfee'|' >> /tmp/tmpa
                oldfee=`echo $line | cut -d'|' -f 5`
                oldbal=`echo $line | cut -d'|' -f 6`
                newbal=`expr $newfee - $oldfee + $oldbal`
                echo $newfee' '$newbal > /tmp/newbal
                if [ $newbal -gt 0 ]
                then
                        echo -n $newbal'|' >> /tmp/tmpa
                else
                        echo -n '0|' >> /tmp/tmpa
                fi
                #echo -n `date +%d-%m-%Y`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 7`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 8`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 9`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 10`'|' >> /tmp/tmpa
                echo -n $name'|' >> /tmp/tmpa
                echo $city >> /tmp/tmpa
```

```
        else
                echo $line >> /tmp/tmpa
        fi
done
mv /tmp/tmpa a
echo 'Type="FormType":Value="ResponseForm"'
read newfee newbal < /tmp/newbal
echo 'Type="Say":RPrompt="Your fee for the subscription is $":Value="'$newfee'"'
if [ $newbal -lt 0 ]
then
        newbal=`expr $newbal \* -1`
        echo 'Type="Say":RPrompt="A cheque of $'$newbal' will be sent to you to compensate
for excess balance"'
else
        echo 'Type="Say":RPrompt="Your balance is $":Value="'$newbal'"'
fi
echo 'Type="Say":RPrompt="Thank you":Value="'"
rm /tmp/param /tmp/newbal
```

prefer: Script to generate form for damaged / missing newspaper complaint

```
#!/bin/sh
cp /dev/null /tmp/param1
cut -d'=' -f 2 | sed "s/\"//g
s/ /\\\\\\\ /g" | while read aa
do
        echo -n $aa' ' >> /tmp/param1
done
echo "" >>/tmp/param1
read acctno preference < /tmp/param1
cnt=`grep -c "^$acctno|" a`
if [ $cnt -eq 0 ]
then
        echo 'Type="FormType":Value="ResponseForm"'
        echo 'Type="Say":RPrompt="Sorry, the account number you provided does not
exist":Value="'"
        exit 0
fi

if [ "$preference" = "the newspaper" ]
then
        echo 'Type="FormType":Value="ResponseForm"'
```

```
        echo 'Type="Say":RPrompt="You will be sent today'\"s newspaper":Value=""'
        echo 'Type="Say":RPrompt="Thank you":Value=""'
        exit 0
fi
touch /tmp/tmpa
cat a | while read line
do
        cacno=`echo $line | cut -d'|' -f 1`
        if [ $cacno -eq $acctno ]
        then
                echo "yes" > /tmp/found
                echo -n $cacno'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 2`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 3`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 4`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 5`'|' >> /tmp/tmpa
                bal=`echo $line | cut -d'|' -f 6`
                bal=`expr $bal - 1`
                echo -n "$bal|" >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 7`'|' >> /tmp/tmpa
                echo -n `echo $line | cut -d'|' -f 8`'|' >> /tmp/tmpa
                echo -n $sdate'|' >> /tmp/tmpa
                echo $period >> /tmp/tmpa
        else
                echo $line >> /tmp/tmpa
        fi
done
mv /tmp/tmpa a
echo 'Type="FormType":Value="ResponseForm"'
echo 'Type="Say":RPrompt="Your account has been credited":Value=""'
echo 'Type="Say":RPrompt="Thank you":Value=""'
rm /tmp/param1 /tmp/found
```

PATENT
Docket No.: 4428-4001

WE CLAIM:

1. A method performed in a system having multiple navigable nodes interconnected in a hierarchical arrangement comprising:

at a first node, receiving an input from a user of the system, the input containing at least one word identifiable with at least one keyword from among multiple keywords,

identifying at least one node, other than the first node, that is not directly connected to the first node but is associated with the at least one keyword, and

jumping to the at least one node.

2. The method of claim 1 further comprising:

providing a verbal description associated with the at least one node to the user.

3. The method of claim 1 further comprising:

searching a thesaurus correlating keywords with synonyms.

4. The method of claim 3 wherein the searching further comprises:

identifying the at least one word as synonymous with the at least one keyword.

5. The method of claim 1 further comprising:

determining that the at least one word is neither a keyword nor a synonym of any keyword; and

learning a meaning for the word so that the word will be treated as a learned synonym for at least one particular keyword of the multiple keywords.

6. The method of claim 5 further comprising:

adding the word to a thesaurus so that, when the word is input by a subsequent user, the word will be treated as synonymous with the at least one particular keyword.

7. A method performed in connection with an arrangement of nodes representable as a hierarchical graph containing vertices and edges connecting at least two of the vertices, the method comprising:

receiving an input from a user as a response to a verbal description associated with a first vertex;

analyzing the input to identify a meaningful term that can be associated with at least one keyword;

selecting a vertex in the graph structure that is not connected by an edge to the first vertex, based upon an association between the meaningful term and the at least one keyword and a correlation between the at least one keyword and the vertex; and

jumping to the vertex.

8. A method performed in connection with an arrangement of nodes representable as a hierarchical graph comprising:

correlating keywords with nodes in which the keywords appear to create an inverted index so that the keywords each appear only once and all nodes containing each of the keywords are indexed to those keywords;

maintaining a thesaurus of synonyms for at least some of the keywords;

receiving an input from a user containing a meaningful word;

searching the inverted index to determine whether the meaningful word is a keyword and, if the meaningful word is a keyword, jumping to a node identified in the inverted index as correlated to that keyword, otherwise,

searching the thesaurus to determine if the meaningful word is a synonym for at least one particular keyword and, if the meaningful word is the synonym, using the synonym to identify the at least one particular keyword, and

jumping to at least one node correlated to the at least one particular keyword.

9. The method of claim 8 further comprising:

creating the thesaurus by analyzing at least two files and determining synonymy among application meaningful words contained therein based upon a frequency of co-occurrence among the application meaningful words.

10. A system comprising:

a hierarchically arranged series of nodes;

an inverted index correlating keywords with the nodes;

a thesaurus correlating at least some keywords with synonyms for those keywords;

a processor executable learning procedure configured to, upon receipt of a term that is identified as neither a synonym nor a keyword based upon a search of both the inverted index and the thesaurus,

(a) identify the term as at last one particular synonym for at least one particular keyword and

(b) correlate the term with the at least one particular keyword,

so that when a subsequent user provides the term the system will operate as if the term was synonymous with the at least one particular keyword.

11. The system of claim 10 further comprising:

a set of verbal descriptions for at least some of the nodes.

12. The system of claim 10 wherein at least one of the nodes is a service node.

13. The system of claim 10 further comprising an interactive voice response system and wherein the hierarchically arranged series of nodes is part of the interactive voice response system.

14. The system of claim 10 wherein the hierarchically arranged series of nodes is part of a file system browser application.

15. The system of claim 10 wherein the hierarchically arranged series of nodes is part of a navigation system for television listings.

16. The system of claim 10 wherein the hierarchically arranged series of nodes is part of one of a document navigation or a document retrieval system.

17. The system of claim 10 wherein the hierarchically arranged series of nodes is part of a geographic information system.

18. A transaction processing system, having a hierarchical arrangement of nodes and configured to interact with a user so that the user can navigate among the nodes in the hierarchy, the system comprising:

an inverted index correlating keywords with at least some of the nodes in the hierarchical arrangement so that when the user interacts with the system and provides an input in response to a verbal description from one node in the hierarchy and the response includes a meaningful word correlatable with a keyword, the system will identify at least one node that is correlated to the meaningful word by the inverted index and jump to that at least one node without first traversing any other node.

19. The system of claim 18 further comprising:

a thesaurus correlating at least some of the keywords with synonyms for the at least some keywords.

20. The system of claim 18 further comprising:

at least one stored learned word correlated to a keyword.

21. A method performed by a program executed by a processor to navigate among a hierarchically arranged group of nodes, each of the nodes having an associated verbal description, the method comprising:

eliminating stop words and duplicates from the verbal descriptions to create a list of keywords;

creating a list of thesaurus words;

creating a first matrix comprising a correlation of at least some thesaurus words with at least some keywords;

creating a second matrix by calculating cosine values from a co-occurrence analysis of the entries in the first matrix;

determining a synonymy among the at least some thesaurus words and the at least some keywords; and

creating a thesaurus configured as an inverted index based upon the synonomy.

22. The method of claim 21 further comprising:

tracking frequency of use of the nodes.

23. The method of claim 22 further comprising:

ranking the nodes based upon a result of the tracking.

24. The method of claim 21 further comprising:

pruning a node from the group of nodes based upon a frequency of usage criterion.

25. The method of claim 21 further comprising:

adding a synonym entry into the thesaurus based upon a result of an unknown word analysis.

26. The method of claim 21 wherein the thesaurus further comprises at least some learned entries, the method further comprising:

deleting a learned entry based upon satisfaction of a frequency of use criterion.

ABSTRACT

A method performed in a system having multiple navigable nodes interconnected in a hierarchical arrangement involves receiving an input containing at least one word identifiable with at least one keyword, identifying at least one node, other than the first node, not directly connected to the first node, but associated with the at least one keyword, and jumping to the identified node. A transaction processing system having a hierarchical arrangement of nodes and is configured for user navigation among the nodes. The system has an inverted index correlating keywords with at least some nodes in the arrangement so that when the user provides an input in response to a verbal description and the response includes a meaningful word correlatable with a keyword, the system will identify at least one node correlated to the meaningful word by the inverted index and jump to that node without first traversing any other node.

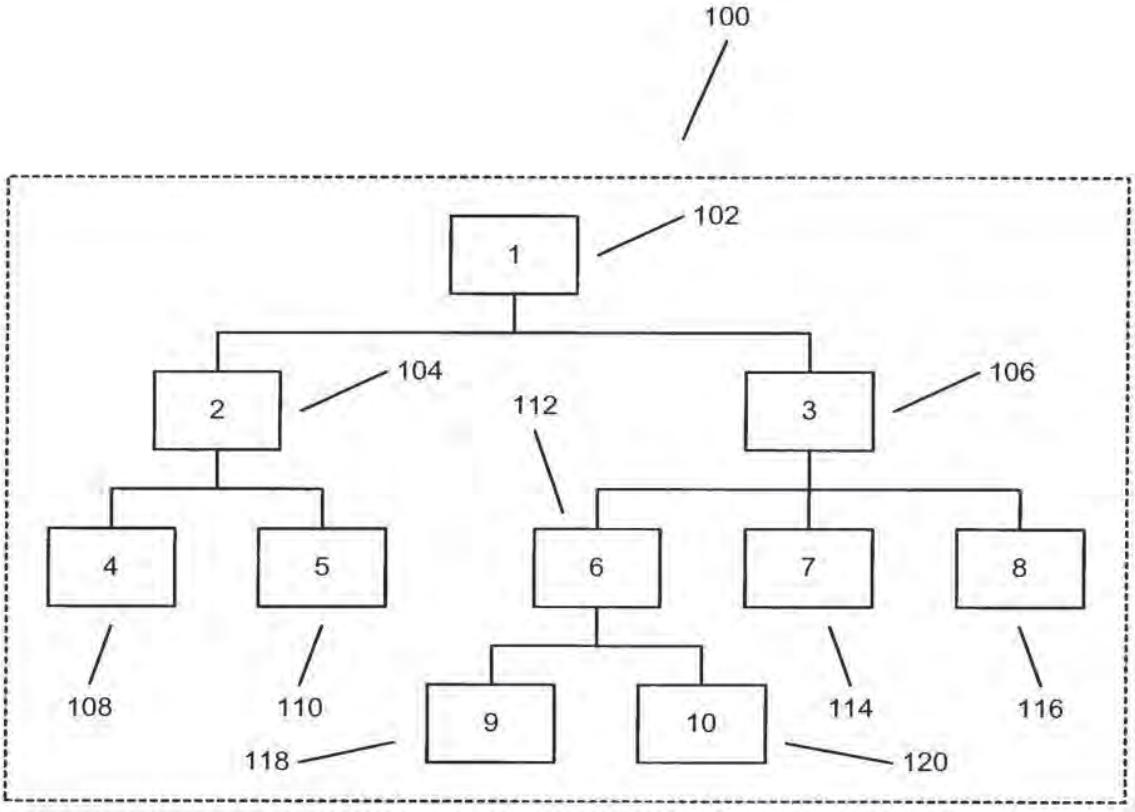
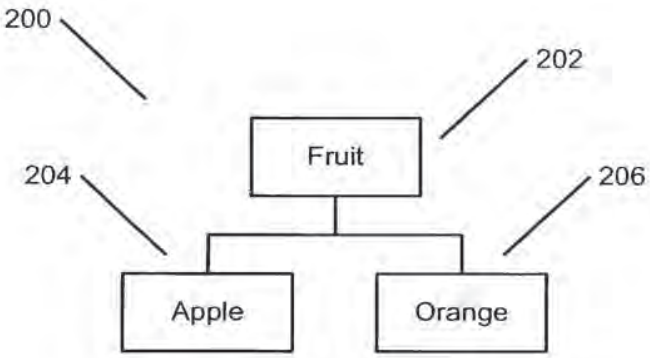


FIG. 1

FIG. 2


200
202
Fruit
204
206
Apple
Orange

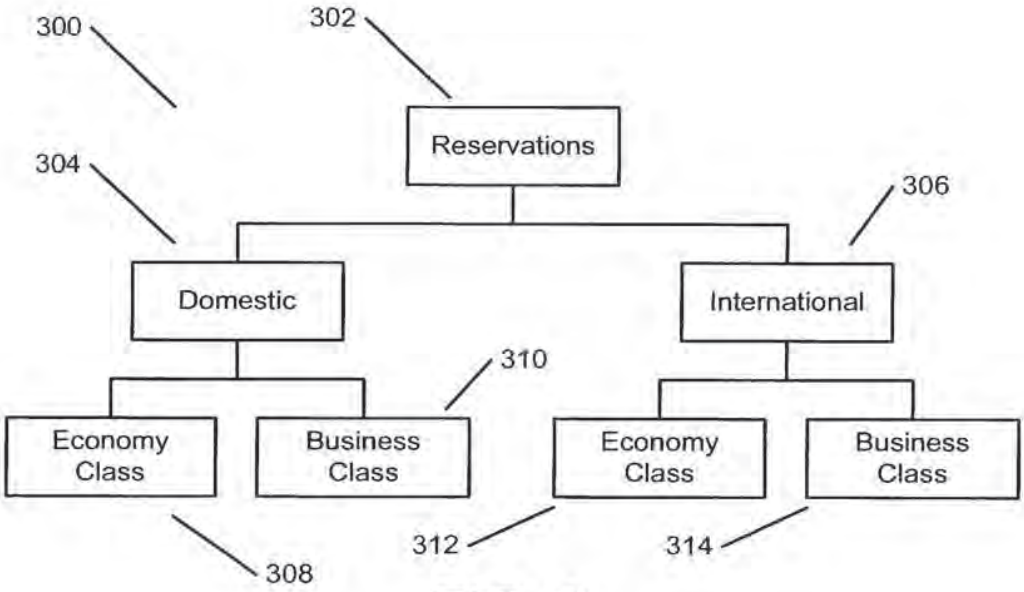

300
302
Reservations
304
306
Domestic
International
310
Economy Class
Business Class
Economy Class
Business Class
312
314
308

FIG. 3

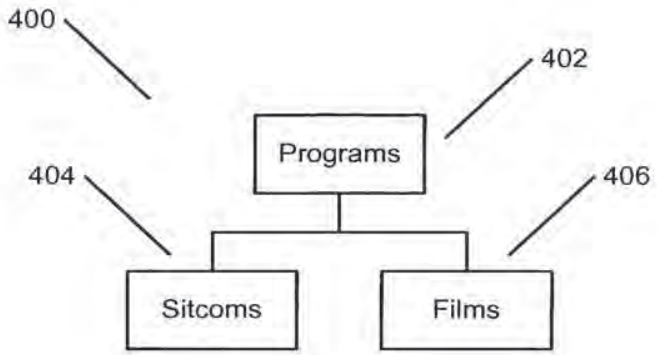

400
402
Programs
404
406
Sitcoms
Films

FIG. 4

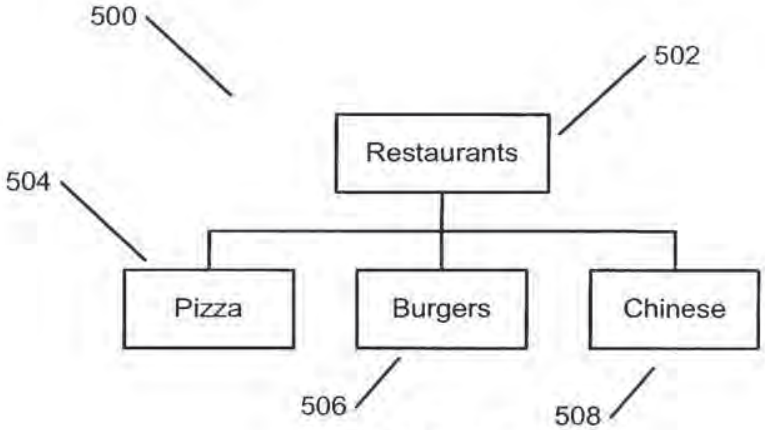

500
502
Restaurants
504
Pizza
Burgers
Chinese
506
508

FIG. 5

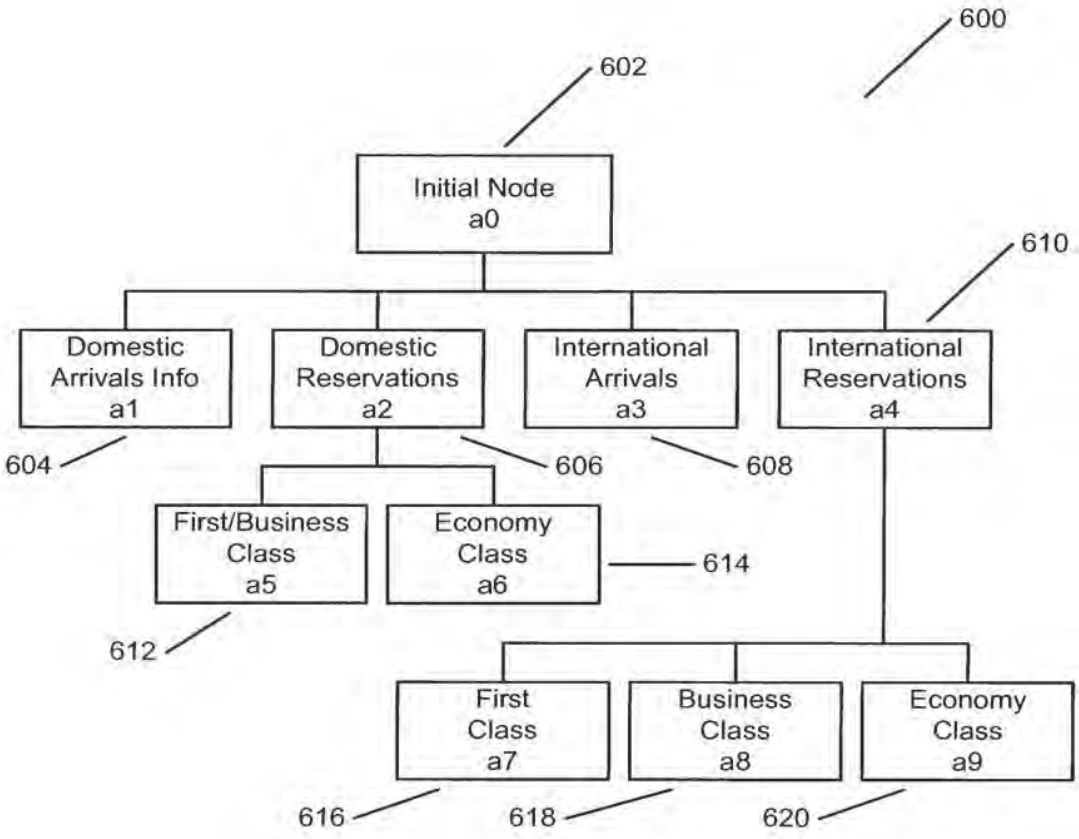

600
602
Initial Node
a0
610
Domestic
Arrivals Info
a1
Domestic
Reservations
a2
International
Arrivals
a3
International
Reservations
a4
604
606
608
First/Business
Class
a5
Economy
Class
a6
614
612
First
Class
a7
Business
Class
a8
Economy
Class
a9
616
618
620

FIG. 6

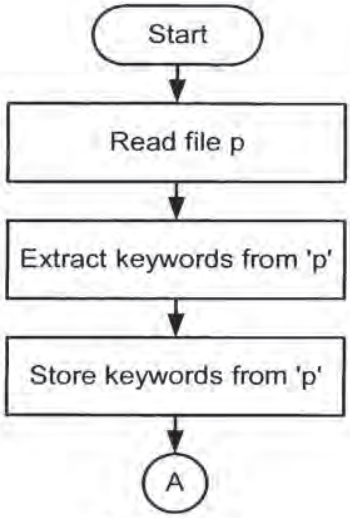


FIG. 7A

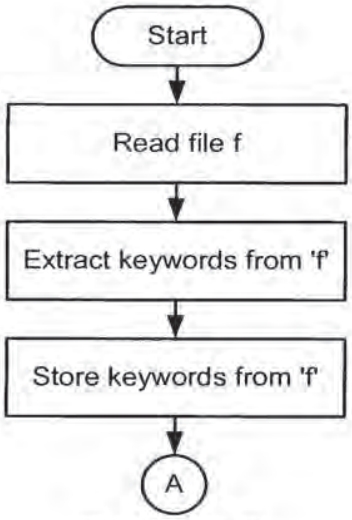
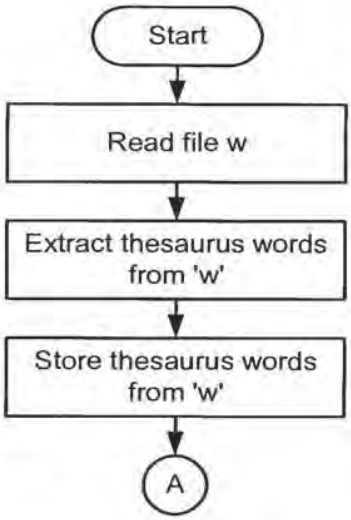


FIG. 7B

FIG. 8



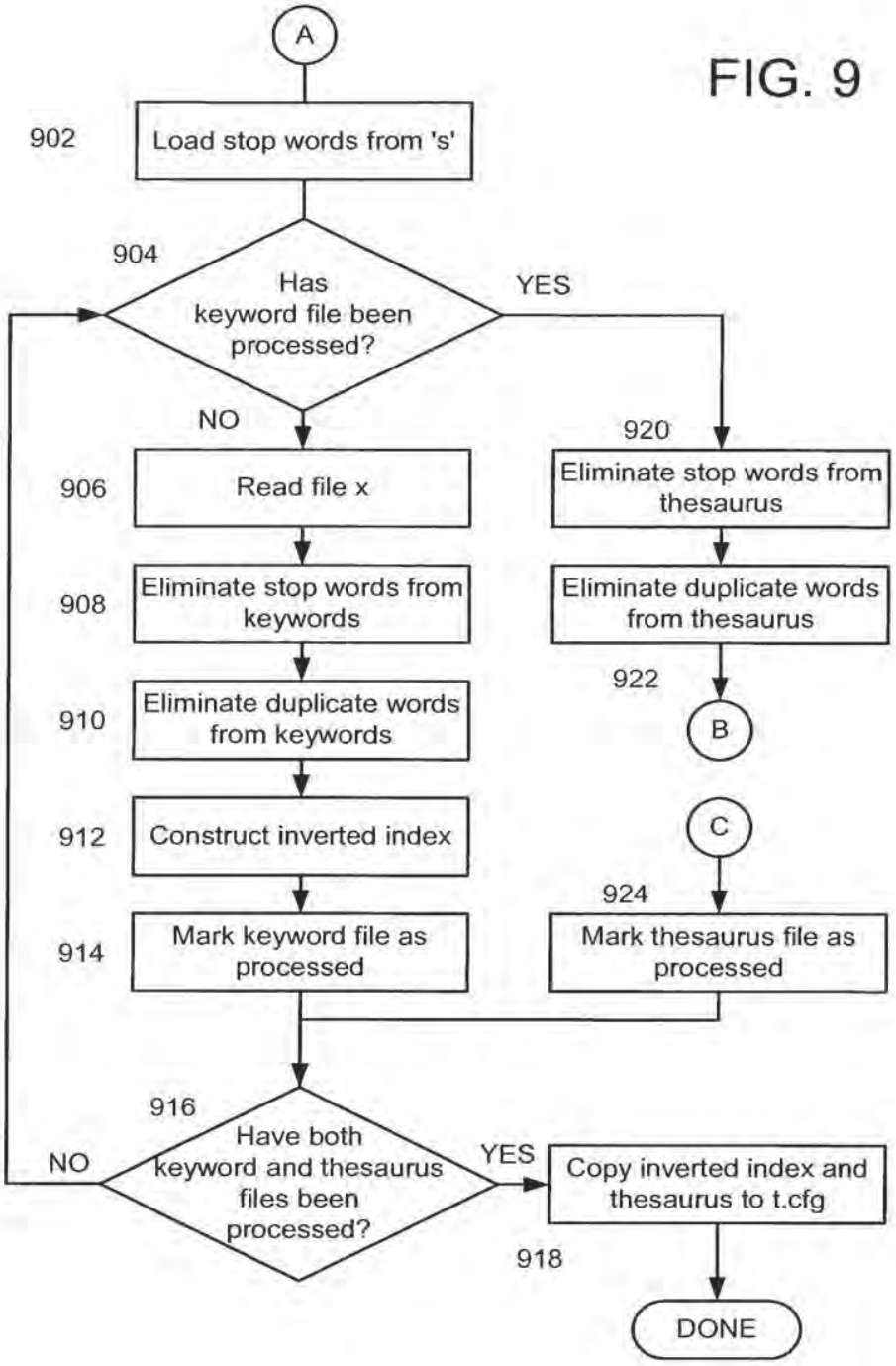
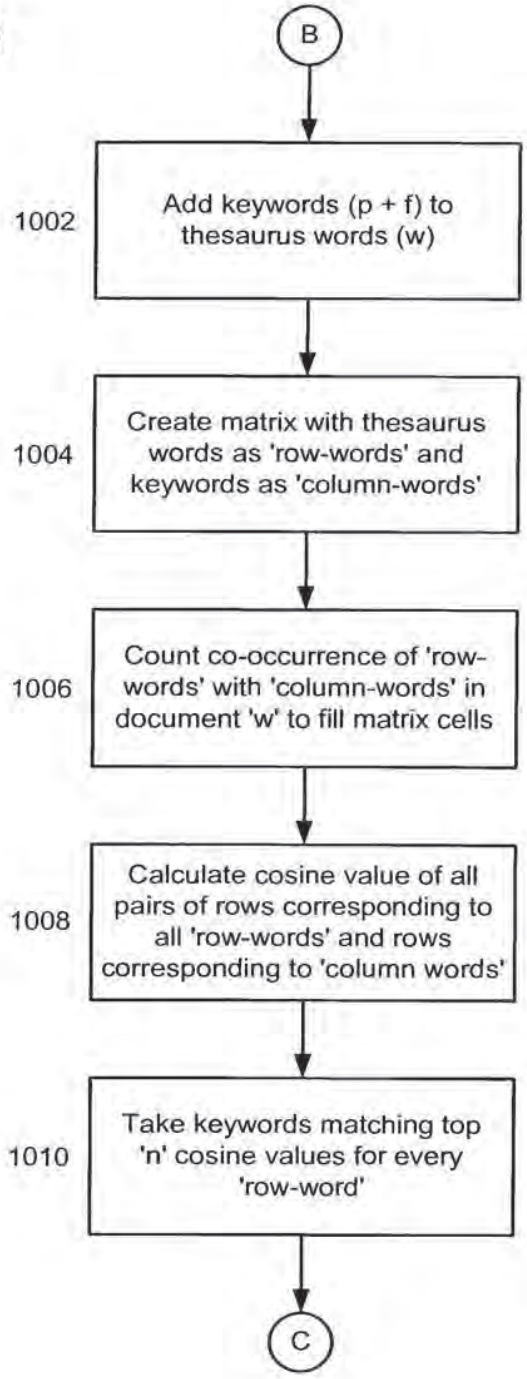

A
902
Load stop words from 's'
904
Has keyword file been processed?
YES
NO
906
Read file x
908
Eliminate stop words from keywords
910
Eliminate duplicate words from keywords
912
Construct inverted index
914
Mark keyword file as processed
920
Eliminate stop words from thesaurus
Eliminate duplicate words from thesaurus
922
B
C
924
Mark thesaurus file as processed
916
Have both keyword and thesaurus files been processed?
NO
YES
Copy inverted index and thesaurus to t.cfg
918
DONE

FIG. 9

FIG. 10


B
1002
Add keywords (p + f) to thesaurus words (w)
1004
Create matrix with thesaurus words as 'row-words' and keywords as 'column-words'
1006
Count co-occurrence of 'row-words' with 'column-words' in document 'w' to fill matrix cells
1008
Calculate cosine value of all pairs of rows corresponding to all 'row-words' and rows corresponding to 'column words'
1010
Take keywords matching top 'n' cosine values for every 'row-word'
C

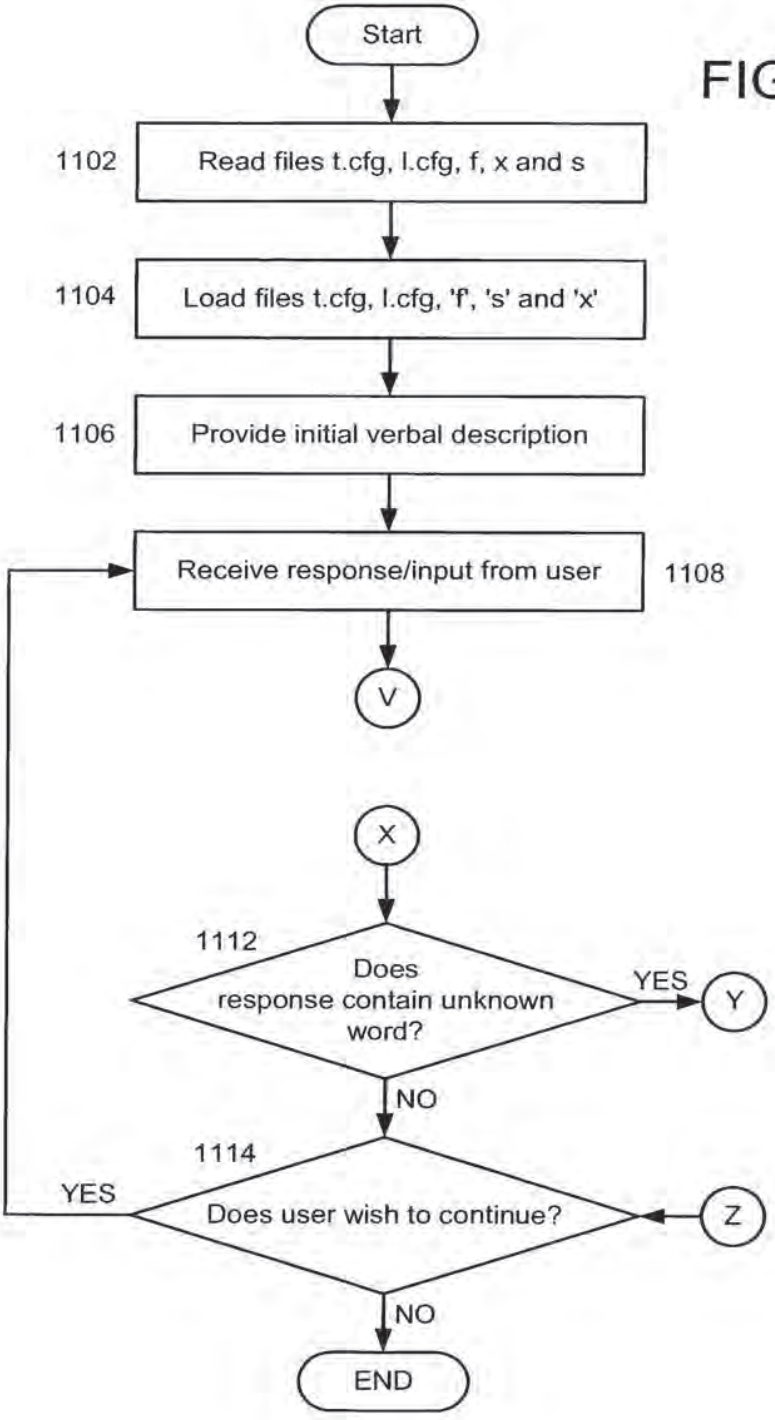

Start
1102
Read files t.cfg, l.cfg, f, x and s
1104
Load files t.cfg, l.cfg, 'f', 's' and 'x'
1106
Provide initial verbal description
Receive response/input from user
1108
V
X
1112
Does response contain unknown word?
YES
Y
NO
1114
YES
Does user wish to continue?
Z
NO
END

FIG. 11

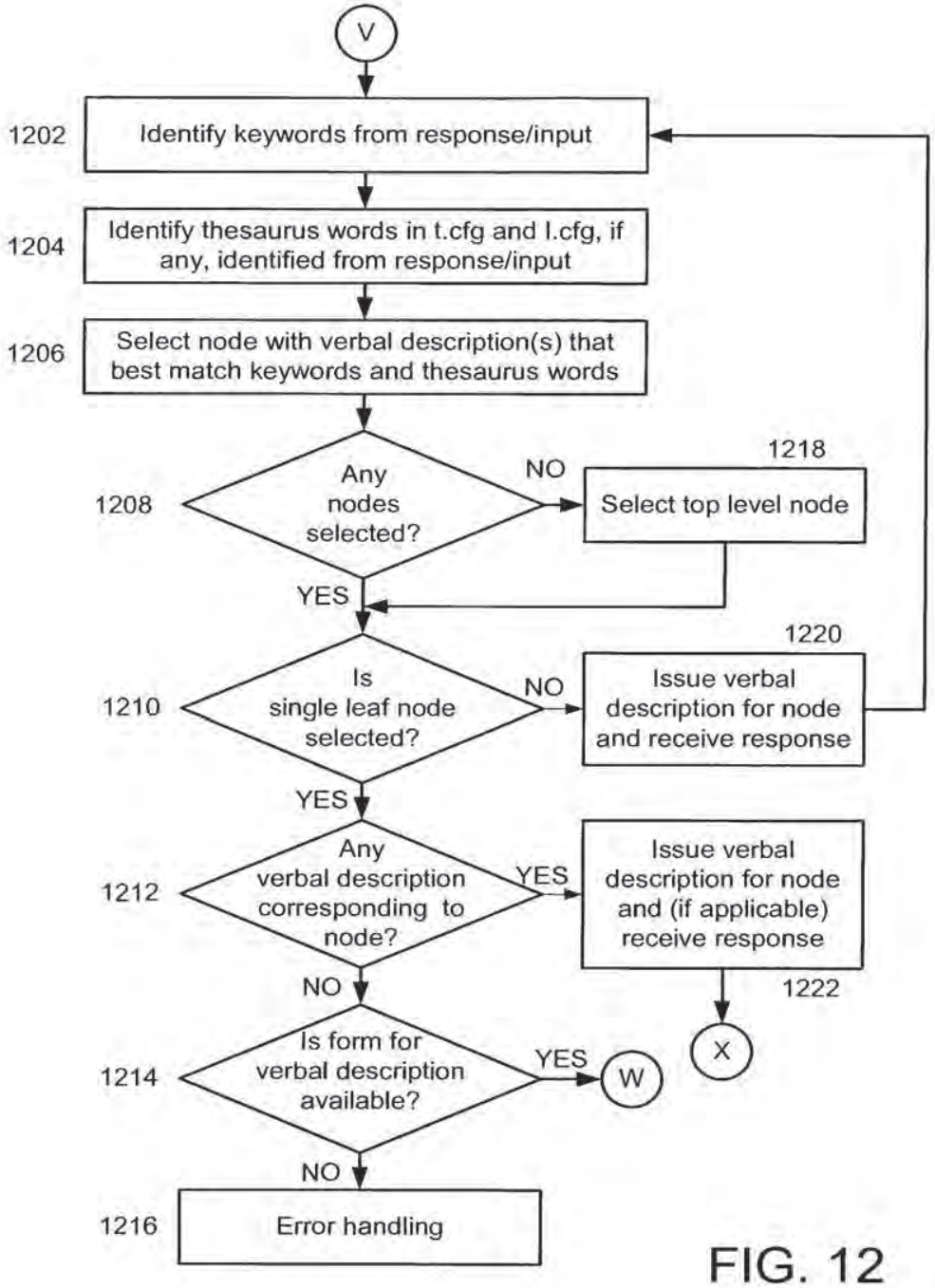

V
1202
Identify keywords from response/input
1204
Identify thesaurus words in t.cfg and l.cfg, if any, identified from response/input
1206
Select node with verbal description(s) that best match keywords and thesaurus words
1208
Any nodes selected?
NO
1218
Select top level node
YES
1210
Is single leaf node selected?
NO
1220
Issue verbal description for node and receive response
YES
1212
Any verbal description corresponding to node?
YES
Issue verbal description for node and (if applicable) receive response
1222
X
NO
1214
Is form for verbal description available?
YES
W
NO
1216
Error handling

FIG. 12

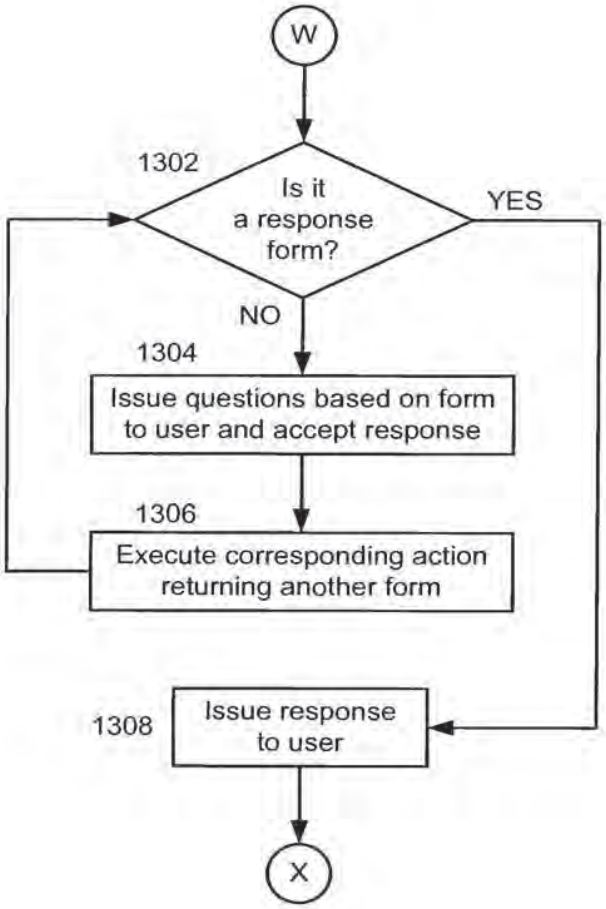

W
1302
Is it
a response
form?
YES
NO
1304
Issue questions based on form
to user and accept response
1306
Execute corresponding action
returning another form
1308
Issue response
to user
X

FIG. 13

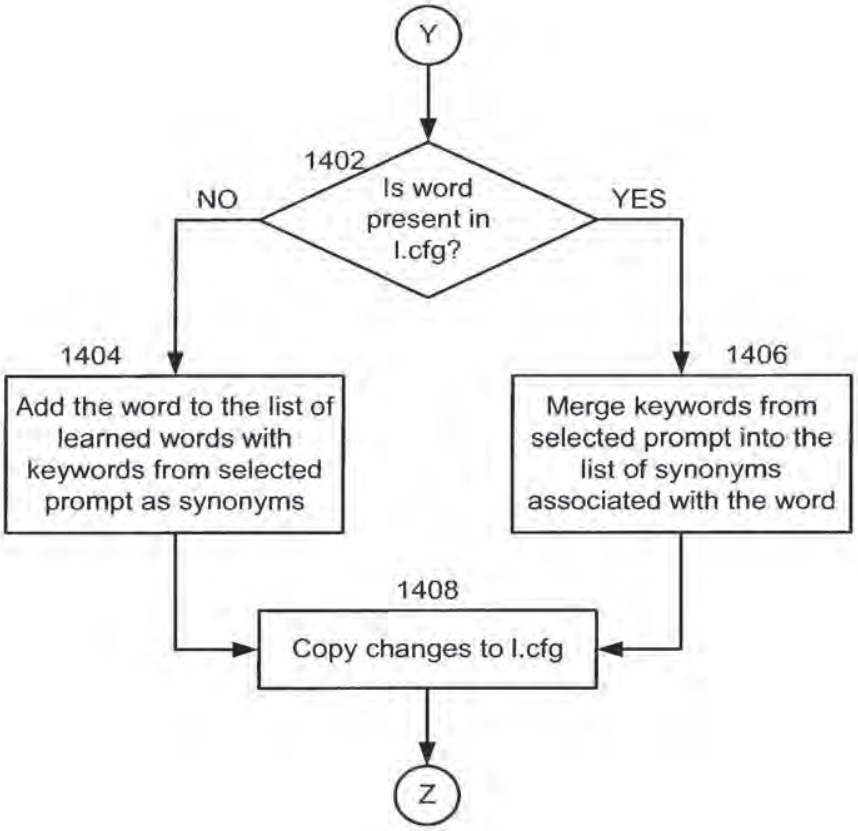

Y
1402
Is word present in l.cfg?
NO
YES
1404
Add the word to the list of learned words with keywords from selected prompt as synonyms
1406
Merge keywords from selected prompt into the list of synonyms associated with the word
1408
Copy changes to l.cfg
Z

FIG. 14

Docket No.: 4428-4001

COMBINED DECLARATION AND POWER OF ATTORNEY FOR ORIGINAL, DESIGN, NATIONAL STAGE OF PCT, SUPPLEMENTAL, DIVISIONAL, CONTINUATION OR CONTINUATION-IN-PART APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION PROCESSING SYSTEM

the specification of which

a. ☒ is attached hereto.

b. ☐ was filed on _____ as application Serial No. _____ and was amended on _____ (if applicable).

PCT FILED APPLICATION ENTERING NATIONAL STAGE

c. ☐ was described and claimed in International Application No. ___ filed on _____ and as amended on ____. (if any).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 C.F.R. § 1.56.

I hereby specify the following as the correspondence address to which all communications about this application are to be directed:

SEND CORRESPONDENCE TO:

☒ Bar Code label attached (see right)

☐ Address Shown (see below)

MORGAN & FINNEGAN, L.L.P.
345 Park Avenue
New York, N.Y. 10154

27123
PATENT TRADEMARK OFFICE

↑AFFIX CUSTOMER NO. LABEL ABOVE ↑

DIRECT TELEPHONE CALLS TO: 212-758-4800

☐ I hereby claim foreign priority benefits under Title 35, United States Code § 119 (a)-(d) or under § 365(b) of any foreign application(s) for patent or inventor's certificate or under § 365(a) of any PCT international application(s) designating at least one country other than the U.S. listed below and also have identified below such foreign application(s) for patent or inventor's certificate or such PCT international application(s) filed by me on the same subject matter having a filing date within twelve (12) months before that of the application on which priority is claimed:

☐ The attached 35 U.S.C. § 119 claim for priority for the application(s) listed below forms a part of this declaration.

Country/PCT	Application Number	Date of filing (day, month, yr)	Date of issue (day, month, yr)	Priority Claimed
				☐ Y ☐ N
				☐ Y ☐ N
				☐ Y ☐ N

☐ I hereby claim the benefit under 35 U.S.C. § 119(e) of any U.S. provisional application(s) listed below.

Provisional Application No.	Date of filing (day, month, yr)

ADDITIONAL STATEMENTS FOR DIVISIONAL, CONTINUATION OR CONTINUATION-IN-PART OR PCT INTERNATIONAL APPLICATION(S DESIGNATING THE U.S.)

I hereby claim the benefit under Title 35, United States Code § 120 of any United States application(s) or under § 365(c) of any PCT international application(s) designating the U.S. listed below.

US/PCT Application Serial No.	Filing Date	Status (patented, pending, abandoned)/ U.S. application no. assigned (For PCT)

☐ In this continuation-in-part application, insofar as the subject matter of any of the claims of this application is not disclosed in the above listed prior United States or PCT international application(s) in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56(a) which occurred between the filing date of the prior application(s) and the national or PCT international filing date of this application.

Docket No.: 4428-4001

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

I hereby appoint the following attorneys and/or agents with full power of substitution and revocation, to prosecute this application, to receive the patent, and to transact all business in the Patent and Trademark Office connected therewith: David H. Pfeffer (Reg. No. 19,825), Harry C. Marcus (Reg. No. 22,390), Robert E. Paulson (Reg. No. 21,046), Stephen R. Smith (Reg. No. 22,615), Kurt E. Richter (Reg. No. 24,052), J. Robert Dailey (Reg. No. 27,434), Eugene Moroz (Reg. No. 25,237), John F. Sweeney (Reg. No. 27,471), Arnold I. Rady (Reg. No. 26,601), Christopher A. Hughes (Reg. No. 26,914), William S. Feiler (Reg. No. 26,728), Joseph A. Calvaruso (Reg. No. 28,287), James W. Gould (Reg. No. 28,859), Richard C. Komson (Reg. No. 27,913), Israel Blum (Reg. No. 26,710), Bartholomew Verdirame (Reg. No. 28,483), Maria C.H. Lin (Reg. No. 29,323), Joseph A. DeGirolamo (Reg. No. 28,595), Michael P. Dougherty (Reg. No. 32,730), Seth J. Atlas (Reg. No. 32,454), Andrew M. Riddles (Reg. No. 31,657), Bruce D. DeRenzi (Reg. No. 33,676), Mark J. Abate (Reg. No. 32,527), John T. Gallagher (Reg. No. 35,516), Steven F. Meyer (Reg. No. 35,613), Kenneth H. Sonnenfeld (Reg. No. 33,285), Tony V. Pezzano (Reg. No. 38,271), Andrea L. Wayda (Reg. 43,979), Walter G. Hanchuk (Reg. No. 35,179), John W. Osborne (Reg. No. 36,231), Robert K. Goethals (Reg. No. 36,813), Peter N. Fill (Reg. No. 38,876), Mary J. Morry (Reg. No. 34,398) and Kenneth S. Weitzman (Reg. No. 36,306) of Morgan & Finnegan, L.L.P. whose address is: 345 Park Avenue, New York, New York, 10154; and Michael S. Marcus (Reg. No. 31,727), and John E. Hoel (Reg. No. 26,279), of Morgan & Finnegan, L.L.P., whose address is 1775 Eye Street, Suite 400, Washington, D.C. 20006.

☒ I hereby authorize the U.S. attorneys and/or agents named hereinabove to accept and follow instructions from us as to any action to be taken in the U.S. Patent and Trademark Office regarding this application without direct communication between the U.S. attorneys and/or agents and me. In the event of a change in the person(s) from whom instructions may be taken I will so notify the U.S. attorneys and/or agents named hereinabove.

Full name of second inventor:	**PRASHANT PARIKH**
Inventor's signature*	_____ 18 Nov 2002 Date
Residence:	**254 East 68th Street, Apart. 21D, New York, New York 10021**
Citizenship:	**Indian**
Post Office Address:	**Same as residence**

Docket No.: 4428-4001

Full name of second inventor:	**STANLEY PETERS**
Inventor's signature*	Stanley Peters 13 Nov. 2002 (Date)
Residence:	**128 Hillside Avenue, Menlo Park, CA 94025**
Citizenship:	**U.S.A.**
Post Office Address:	**Same as residence**

☐ ATTACHED IS ADDED PAGE TO COMBINED DECLARATION AND POWER OF ATTORNEY FOR SIGNATURE BY THIRD AND SUBSEQUENT INVENTORS FORM.

*Before signing this declaration, each person signing must:

1. Review the declaration and verify the correctness of all information therein; and

2. Review the specification and the claims, including any amendments made to the claims.

After the declaration is signed, the specification and claims are not to be altered.

To the inventor(s):

The following are cited in or pertinent to the declaration attached to the accompanying application:

Title 37, Code of Federal Regulation, §1.56

Duty to disclose information material to patentability

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclose information exists with respect to each pending claim until the claim is cancelled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is cancelled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§ 1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

(1) Prior art cited in search reports of a foreign patent office in a counterpart application, and

(2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.

(b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and

(1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or

(2) It refutes, or is inconsistent with, a position the applicant takes in:

(i) Opposing an argument of unpatentability relied on by the Office, or

(ii) Asserting an argument of patentability.

(iii) A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

(c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:

(1) Each inventor named in the application;

(2) Each attorney or agent who prepares or prosecutes the application; and

(3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.

(d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.

(e) In any continuation-in-part application, the duty under this section includes the duty to disclose to the Office all information known to the person to be material to patentability, as defined in paragraph (b) of this section, which became available between the filing date of the prior application and the National or PCT international filing date of the continuation-in-part application.

Title 35, U.S. Code § 101

Inventions patentable

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Title 35 U.S. Code § 102

Conditions for patentability; novelty and loss of right to patent

A person shall be entitled to a patent unless --

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for patent, or

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States, or

(c) The has abandoned the invention, or

(d) the invention was first patented or caused to be patented, or was the subject of an inventor's certificate, by the applicant or his legal representatives or assigns in a foreign country prior to the date of the application for patent in this country on an application for patent or inventor's certificate filed more than twelve months before the filing of the application in the United States, or

(e) The invention was described in--

(1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effect under this subsection of a national application published under section 122(b) only if the international application designating the United States was published under Article 21(2)(a) of such treaty in the English language; or

(2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that a patent shall not be deemed filed in the United States for the purposes of this subsection based on the filing of an international application filed under the treaty defined in section 351(a); or

(f) he did not himself invent the subject matter sought to be patented, or

(g) (1) during the course of an interference conducted under section 135 or section 291, another inventor involved therein establishes, to the extent permitted in section 104, that before such person's invention thereof the invention was made by such other inventor and not abandoned, suppressed, or concealed, or (2) before such person's invention thereof, the invention was made in this country by another inventor who had not abandoned, suppressed, or concealed it. In determining priority of invention under this subsection, there shall be considered not only the respective dates of conception and reduction to practice of the invention, but also the reasonable diligence of one who was first to conceive and last to reduce to practice, from a time prior to conception by the other.

Title 35, U.S. Code § 103

103. Conditions for patentability; non obvious subject matter

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

(b) (1) Notwithstanding subsection (a), and upon timely election by the applicant for patent to proceed under this subsection, a biotechnological process using or resulting in a composition of matter that is novel under section 102 and nonobvious under subsection (a) of this section shall be considered nonobvious if—

(A) claims to the process and the composition of matter are contained in either the same application for patent or in separate applications having the same effective filing date; and

(B) the composition of matter, and the process at the time it was invented, were owned by the same person or subject to an obligation of assignment to the same person.

(2) A patent issued on a process under paragraph (1)—

(A) shall also contain the claims to the composition of matter used in or made by that process, or

(B) shall, if such composition of matter is claimed in another patent, be set to expire on the same date as such other patent, notwithstanding section 154.

(3) For purposes of paragraph (1), the term "biotechnological process" means--

(A) a process of genetically altering or otherwise inducing a single- or multi-celled organism to--

(i) express an exogenous nucleotide sequence,

(ii) inhibit, eliminate, augment, or alter expression of an endogenous nucleotide sequence, or

(iii) express a specific physiological characteristic not naturally associated with said organism;

(B) cell fusion procedures yielding a cell line that expresses a specific protein, such as a monoclonal antibody; and

(C) a method of using a product produced by a process defined by subparagraph (A) or (B), or a combination of subparagraphs (A) and (B).

(c) Subject matter developed by another person, which qualifies as prior art only under one or more of subsections (e), (f), and (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person.

Title 35, U.S. Code § 112 (in part)

Specification

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention.

The specification shall conclude with one or more claims particularly printing out and distinctly claiming the subject matter which the applicant regards as his invention.

Title 35, U.S. Code, § 119

Benefit of earlier filing date; right of priority

(a) An application for patent for an invention filed in this country by any person who has, or whose legal representatives or assigns have, previously regularly filed an application for a patent for the same invention in a foreign country which affords similar privileges in the case of applications filed in the United States or to citizens of the United States, or in a WTO member country, shall have the same effect as the same

application would have if filed in this country on the date on which the application for patent for the same invention was first filed in such foreign country, if the application in this country is filed within twelve months from the earliest date on which such foreign application was filed; but no patent shall be granted on any application for patent for an invention which had been patented or described in a printed publication in any country more than one year before the date of the actual filing of the application in this country, or which had been in public use or on sale in this country more than one year prior to such filing.

(b)

(1) No application for patent shall be entitled to this right of priority unless a claim is filed in the Patent and Trademark Office, identifying the foreign application by specifying the application number on that foreign application, the intellectual property authority or country in or for which the application was filed, and the date of filing the application, at such time during the pendency of the application as required by the Director.

(2) The Director may consider the failure of the applicant to file a timely claim for priority as a waiver of any such claim. The Director may establish procedures, including the payment of a surcharge, to accept an unintentionally delayed claim under this section.

(3) The Director may require a certified copy of the original foreign application, specification, and drawings upon which it is based, a translation if not in the English language, and such other information as the Director considers necessary. Any such certification shall be made by the foreign intellectual property authority in which the foreign application was filed and show the date of the application and of the filing of the specification and other papers.

(c) In like manner and subject to the same conditions and requirements, the right provided in this section may be based upon a subsequent regularly filed application in the same foreign country instead of the first filed foreign application, provided that any foreign application filed prior to such subsequent application has been withdrawn, abandoned, or otherwise disposed of, without having been laid open to public inspection and without leaving any rights outstanding, and has not served, nor thereafter shall serve, as a basis for claiming a right of priority.

(d) Applications for inventors' certificates filed in a foreign country in which applicants have a right to apply, at their discretion, either for a patent or for an inventor's certificate shall be treated in this country in the same manner and have the same effect for purpose of the right of priority under this section as applications for patents, subject to the same conditions and requirements of this section as apply to applications for patents, provided such applicants are entitled to the benefits of the Stockholm Revision of the Paris Convention at the time of such filing.

(e)

(1) An application for patent filed under section 111(a) or section 363 of this title for an invention disclosed in the manner provided by the first paragraph of section 112 of this title in a provisional application filed under section 111(b) of this title, by an inventor or inventors named in the provisional application, shall have the same effect, as to such invention, as though filed on the date of the provisional application filed under section 111(b) of this title, if the application for patent filed under section 111(a) or section 363 of this title is filed not later than 12 months after the date on which the provisional application was filed and if it contains or is amended to contain a specific reference to the provisional application. No application shall be entitled to the benefit of an earlier filed provisional application under this subsection unless an amendment containing the specific reference to the earlier filed provisional application is submitted at such time during the pendency of the application as required by the Director. The Director may consider the failure to submit such an amendment within that time period as a waiver of any benefit under this subsection. The Director may establish procedures, including the payment of a surcharge, to accept an unintentionally delayed submission of an amendment under this subsection during the pendency of the application.

(2) A provisional application filed under section 111(b) of this title may not be relied upon in any

proceeding in the Patent and Trademark Office unless the fee set forth in subparagraph (A) or (C) of section 41(a)(1) of this title has been paid.

(3) If the day that is 12 months after the filing date of a provisional application falls on a Saturday, Sunday, or Federal holiday within the District of Columbia, the period of pendency of the provisional application shall be extended to the next succeeding secular or business day.

(f) Applications for plant breeder's rights filed in a WTO member country (or in a foreign UPOV Contracting Party) shall have the same effect for the purpose of the right of priority under subsections (a) through (c) of this section as applications for patents, subject to the same conditions and requirements of this section as apply to applications for patents.

(g) As used in this section--

(1) the term "WTO member country" has the same meaning as the term is defined in section 104(b)(2) of this title; and

(2) the term "UPOV Contracting Party" means a member of the International Convention for the Protection of New Varieties of Plants.

Title 35, U.S. Code, § 120

Benefit of earlier filing date in the United States

An application for patent for an invention disclosed in the manner provided by the first paragraph of section 112 of this title in an application previously filed in the United States, or as provided by section 363 of this title, which is filed by an inventor or inventors named in the previously filed application shall have the same effect, as to such invention, as though filed on the date of the prior application, if filed before the patenting or abandonment of or termination of proceedings on the first application or on an application similarly entitled to the benefit of the filing date of the first application and if it contains or is amended to contain a specific reference to the earlier filed application. No application shall be entitled to the benefit of an earlier filed application under this section unless an amendment containing the specific reference to the earlier filed application is submitted at such time during the pendency of the application as required by the Director. The Director may consider the failure to submit such an amendment within that time period as a waiver of any benefit under this section. The Director may establish procedures, including the payment of a surcharge, to accept an unintentionally delayed submission of an amendment under this section.

Please read carefully before signing the Declaration attached to the accompanying Application. If you have any questions, please contact Morgan & Finnegan, L.L.P.

PATENT **Docket No. 4428-4001**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Prashant Parikh
Stanley Peters

Serial No.: To Be Assigned **Examiner:** To Be Assigned

Filed: Herewith **Group Art Unit:** To Be Assigned

For: NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION P ROCESSING SYSTEM

Commissioner Of Patents
Washington, D.C. 20231

ASSOCIATE POWER OF ATTORNEY (37 C.F.R. 1.34)

Please recognize as an Associate Practitioner in this case:

Richard Straussman
Morgan & Finnegan, LLP
345 Park Avenue
New York, NY 10154
Reg. No.: 39,847

Date: 18 Nov. 2002 Signed: _____
Prashant Parikh

Date: 13 Nov. 2002 Signed: _____
Stanley Peters

SEMIOSIS L.L.C.

Date: 18 Nov. 2002 Signed: _____

Name (Print): PRASHANT PARIKH

Its (Title): CHAIRMAN + CEO

Correspondence Address:
Morgan & Finnegan, LLP
345 Park Avenue
New York, NY 10154
Tel.:(212)758-4800/Fax: (212)751-6849

J1011 U.S. PTO
10/299359
11/19/02

PATENT NUMBER and ISSUE DATE

U.S. UTILITY Patent Application

APPL NUM	FILING DATE	CLASS	SUBCLASS	GAU	EXAMINER
10299359	11/19/2002	~~711~~		2186	

**APPLICANTS: Parikh Prashant; Peters Stanley;

**CONTINUING DATA VERIFIED:

** FOREIGN APPLICATIONS VERIFIED:

PG-PUB	DO NOT PUBLISH ☐	RESCIND ☐	
Foreign priority claimed	☐ yes ☐ no	ATTORNEY DOCKET NO	
35 USC 119 conditions met	☐ yes ☐ no	4428-4001	
Verified and Acknowledged Examiners's initials			

TITLE : Navigation in a hierarchical structured transaction processing system

U.S DEPT. OF COMM./PAT.& TM-PTO-436(Rev. 12-94)

NOTICE OF ALLOWANCE MAILED		CLAIMS ALLOWED	
	Assistant Examiner	Total Claims	Print Claim for O.G
ISSUE FEE		DRAWING	
Amount Due / Date Paid		Sheets Drwg. / Figs.Drwg. / Print Fig.	
☐ TERMINAL DISCLAMER	Primary Examiner	Application Examiner	
	PREPARED FOR ISSUE		

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368, Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

FILED WITH: ☐ DISK (CRF) ☐ CD-ROM
(Attached in pocket on right inside flap)

Best Available Copy

SEARCH

Class	Sub.	Date	Exmr.
712	1	2/18/04	bv
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		
	101		
	102		
	103		
	1041		

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

SEARCH NOTES

(List databases searched. Attach search strategy inside.)

	Date	Exmr.
Inventor search Double Patenting	2/18/04	bv
USPTO		
USPGPUB		
EPO		
JPO		
IBM Tech		
Derwent		
ACM		
IEEE		
NPL		
INTERNET		
Search Strategies attached		
Consulted Dor Podoricci for	2/18/04	bv

Best Available Copy

ISSUE SLIP STAPLE AREA (for additional cross-references)

ISSUING CLASSIFICATION

ORIGINAL		CROSS REFERENCE(S)	
CLASS	SUBCLASS	CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)
INTERNATIONAL CLASSIFICATION			
/			
/			
/			
/			
/			^ Continued on Issue Slip Inside File Jacket

INDEX OF CLAIMS

✓ ..... Rejected - (Through numeral) ... Canceled N ..... Non-elected A ..... Appeal
= ..... Allowed + ... Restricted I ..... Interference O ..... Objected

Claim Final	Claim Original	Date	Claim Final	Claim Original	Date	Claim Final	Claim Original	Date
	1			51			101	
	2			52			102	
	3			53			103	
	4			54			104	
	5			55			105	
	6			56			106	
	7			57			107	
	8			58			108	
	9			59			109	
	10			60			110	
	11			61			111	
	12			62			112	
	13			63			113	
	14			64			114	
	15			65			115	
	16			66			116	
	17			67			117	
	18			68			118	
	19			69			119	
	20			70			120	
	21			71			121	
	22			72			122	
	23			73			123	
	24			74			124	
	25			75			125	
	26			76			126	
	27			77			127	
	28			78			128	
	29			79			129	
	30			80			130	
	31			81			131	
	32			82			132	
	33			83			133	
	34			84			134	
	35			85			135	
	36			86			136	
	37			87			137	
	38			88			138	
	39			89			139	
	40			90			140	
	41			91			141	
	42			92			142	
	43			93			143	
	44			94			144	
	45			95			145	
	46			96			146	
	47			97			147	
	48			98			148	
	49			99			149	
	50			100			150	

If more than 150 claims or 9 actions staple additional sheet here

Best Available Copy

11/19/02 J1131 U.S. PTO

11-20-02 10299359.111902A

Docket No. 4428-4001

J1011 U.S. PTO 10/299359 11/19/02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Prashant Parikh and Stanley Peters

Serial No.: To Be Assigned

Filed: Herewith

For: **NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION PROCESSING SYSTEM**

EXPRESS MAIL CERTIFICATE

Express Mail Label No.: **EV062749235US**

Date of Deposit: **November 19, 2002**

I hereby certify that the following attached paper(s) and/or fee

1. Utility Application and Application Fee Transmittal (in duplicate); enclosing Specification (147 pages), claims 1-26 (7 pages), abstract (1 page), 11 sheets of drawings (FIGS. 1-6, 7A, 7B and 8-14);
2. Executed Declaration And Power Of Attorney For Patent Application (9 pages);
3. Executed Associate Power of Attorney (1 page);
4. Recordation Form Cover Sheet (2 pages);
5. Executed Assignment (3 pages)
6. Checks in the amounts of $550.00 and $40.00; and
7. Return postcard.

is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. §1.10 on the date indicated above and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

JAFET COTTO
(Typed or printed name of person mailing papers(s) and/or fee)

(Signature of person mailing paper(s) and/or fee)

Correspondence Address:
MORGAN & FINNEGAN, L.L.P.
345 Park Avenue
New York, NY 10154-0053
(212) 758-4800 Telephone
(212) 751-6849 Facsimile

730304 v1

10/299359
27123
PATENT TRADEMARK OFFICE

Docket No. 4428-4001

Express Mail No. EV062749235US

J1131 U.S. PTO
11/19/02
J1011 U.S. PTO
10/299359
11/19/02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

UTILITY APPLICATION AND FEE TRANSMITTAL §(1.53(b))

Commissioner for Patents
Box Patent Application
Washington, D.C. 20231

Sir:

Transmitted herewith for filing is the patent application of

Inventor(s) names and addresses:

(1) **Prashant Parikh, 254 East 68th Street, Apartment 21D, New York, New York 10021**
Stanley Peters, 128 Hillside Avenue, Menlo Park, California 94025

☐ Additional inventors are listed on a separate sheet

For: **NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION PROCESSING SYSTEM**

Enclosed Are:

147 **page(s) of specification**
1 **page(s) of Abstract**
7 **page(s) of claims (numbered 1-26)**
11 **sheets of Formal Drawings, (FIGS. 16, 7A, 7B and 8-14)**
9 **page(s) of Declaration and Power of Attorney**

☐ Unsigned
☒ **Newly Executed**
☐ Copy from prior application
☐ Deletion of inventors including Signed Statement under 37 C.F.R. §1.63(d)(2)

☐ **REQUEST AND CERTIFICATION UNDER 35 U.S.C. §122(b)(2)(B)(i) (form PTO/SB/35)**
As indicated on the attached Request and Certification, Applicant(s) certify that the invention disclosed in the attached application HAS NOT and WILL NOT be the subject of an application filed in another country, or under a multilateral agreement, that requires publication at eighteen months after filing. Applicant(s) therefore request(s) that the attached application NOT be published under 35 U.S.C. §122(b).

729449 v1

☐ Incorporation by Reference:

☐ The entire disclosure of the prior application, from which a copy of the combined Declaration and Power of Attorney is supplied herein, is considered as being part of the disclosure of the accompanying application and is incorporated herein by reference.

☐ Deletion of Inventors (37 C.F.R. §1.63(d) and §1.33(b)

Signed statement attached deleting inventor(s) named in the prior application serial no. _____, filed _____.

☐ Microfiche Computer Program (Appendix)

☐ page(s) of Sequence Listing

☐ computer readable disk containing Sequence Listing

☐ Statement under 37 C.F.R. §1.821(f) that computer and paper copies of the Sequence Listing are the same

☒ **Assignment Papers (assignment cover sheet and assignment documents)**

☒ **A check in the amount of $40.00 for recording the Assignment**

☐ Charge the Assignment Recordation Fee to Deposit Account No. 13-4500, Order No. _____.

☐ Assignment Papers filed in the parent provisional application Serial No. _____.

☒ **Executed Associate Power of Attorney**

☐ Certification of chain of title pursuant to 37 C.F.R. §3.73(b)

☐ Priority is claimed under 35 U.S.C. §119 for:
Application No(s). _____ , filed _____, in _____ (country).

☐ Certified Copy of Priority Document(s) [_____]

☐ filed herewith

☐ filed in application Serial No. _____, filed _____.

☐ English translation document(s) [_____]

☐ filed herewith

☐ filed in application Serial No. _____, filed _____.

☐ Priority is claimed under 35 U.S.C. §119(e) for _____, filed _____.

☐ Information Disclosure Statement

☐ Copy of [_____] cited references

☐ PTO Form-1449

☐ References cited in parent application Serial No. _____, filed _____.

☐ Related Case Statement under 37 C.F.R. §1.98(a)(2)(iii)

- ☐ A copy of related pending U.S. Application(s) Serial No(s): _____, filed _____, respectively, is attached hereto.
- ☐ A copy of related pending U.S. Application(s) entitled, _____, filed _____ to inventor(s) _____, respectively, is attached hereto.
- ☐ A copy of each related application(s) was submitted in parent application serial no. _____, filed _____.

☐ Preliminary Amendment

☒ **Return receipt postcard (MPEP 503)**

☐ This is a ☐ continuation ☐ divisional ☐ continuation-in-part of prior application serial no. _____, filed _____, to which priority under 35 U.S.C. §120 is claimed.

- ☐ Cancel in this application original claims _____ of the parent application before calculating the filing fee. (At least one original independent claim must be retained for filing purposes.)
- ☐ A Preliminary Amendment is enclosed. (Claims added by this Amendment have been properly numbered consecutively beginning with the number following the highest numbered original claim in the prior application).

☐ The status of the parent application is as follows:

- ☐ A Petition for Extension of Time and a Fee therefor has been or is being filed in the parent application to extend the term for action in the parent application until _____.
- ☐ A copy of the Petition for Extension of Time in the co-pending parent application is attached.
- ☐ No Petition for Extension of Time and Fee therefor are necessary in the co-pending parent application.

☐ Please abandon the parent application at a time while the parent application is pending or at a time when the petition for extension of time in that application is granted and while this application is pending has been granted a filing date, so as to make this application co-pending.

☐ Transfer the drawing(s) from the parent application to this application

☐ Amend the specification by inserting before the first line the sentence:
This is ☐ continuation ☐ divisional ☐ continuation-in-part of co-pending application Serial No. _____, filed _____.

I. CALCULATION OF APPLICATION FEE				
	Number Filed	Number Extra	Rate	Basic Fee $740.00/370.00
Total Claims	26-20 =	6 x	$18.00/ $9.00	$ 54.00
Independent Claims	6-3 =	3 x	$84.00/ $42.00	$ 126.00
☐ Multiple Dependent Claims		If marked, add fee of $270.00 ($135.00)		$0
			TOTAL:	$550.00

☒ **Small entity status is or has been claimed. Reduced fees under 37 C.F.R. §1.9 (f) paid herewith $550.00.**

☒ **A check in the amount of $550.00 in payment of the application filing fees is attached.**

☐ Charge fee to Deposit Account No. 13-4500, Order No. _____. A DUPLICATE COPY OF THIS SHEET IS ATTACHED.

☒ The Commissioner is hereby authorized to charge any additional fees which may be **required for filing this application pursuant to 37 CFR §1.16, including all extension of time fees pursuant to 37 C.F.R. § 1.17 for maintaining copendency with the parent application, or credit any overpayment to Deposit Account No. 13-4500, Order No. 4428-4001. A DUPLICATE COPY OF THIS SHEET IS ATTACHED.**

Respectfully submitted,
MORGAN & FINNEGAN, L.L.P.

Dated: November 19, 2002 By: _____
Richard Straussman
Registration No. 39,847

Correspondence Address:
MORGAN & FINNEGAN, L.L.P.
345 Park Avenue
New York, NY 10154-0053
(212) 758-4800 Telephone
(212) 751-6849 Facsimile

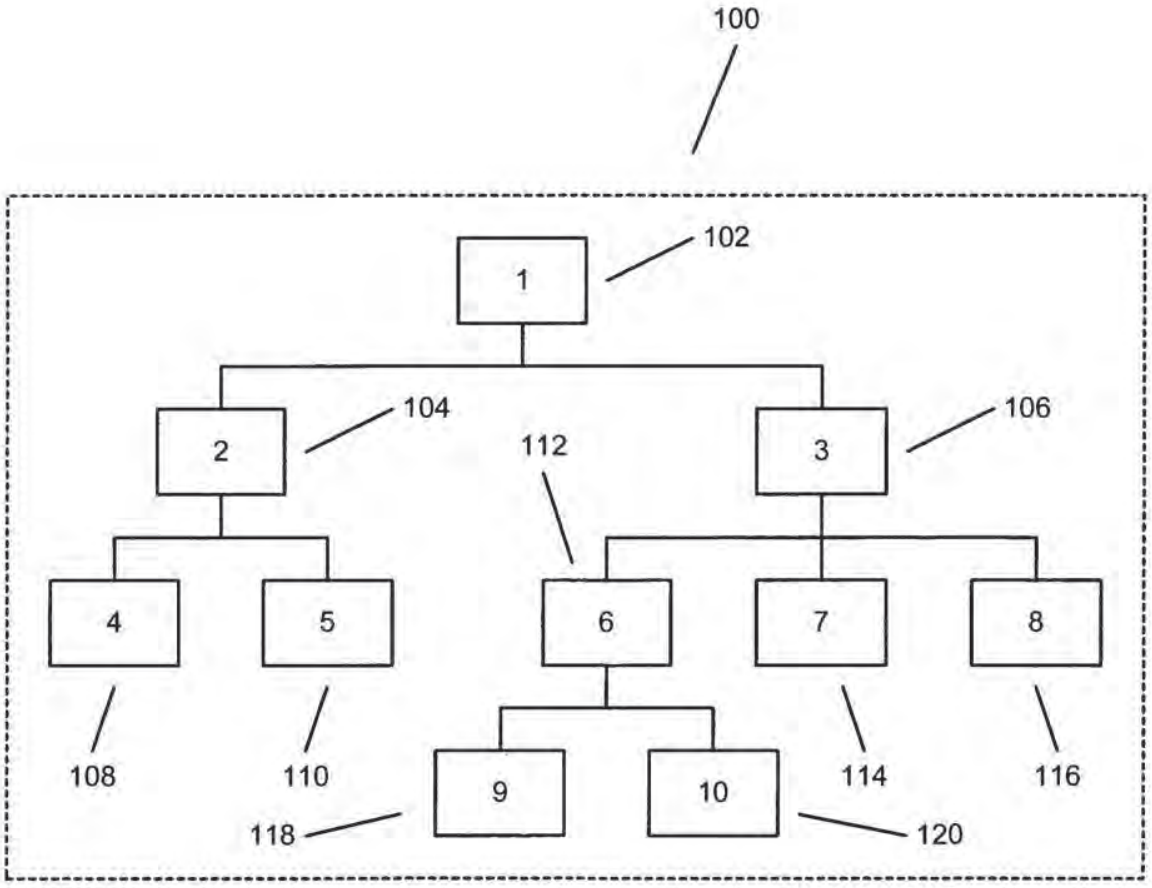
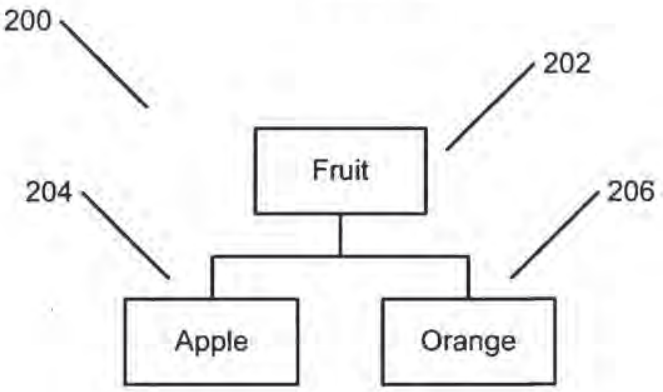


FIG. 1

FIG. 2


200
202
Fruit
204
206
Apple
Orange

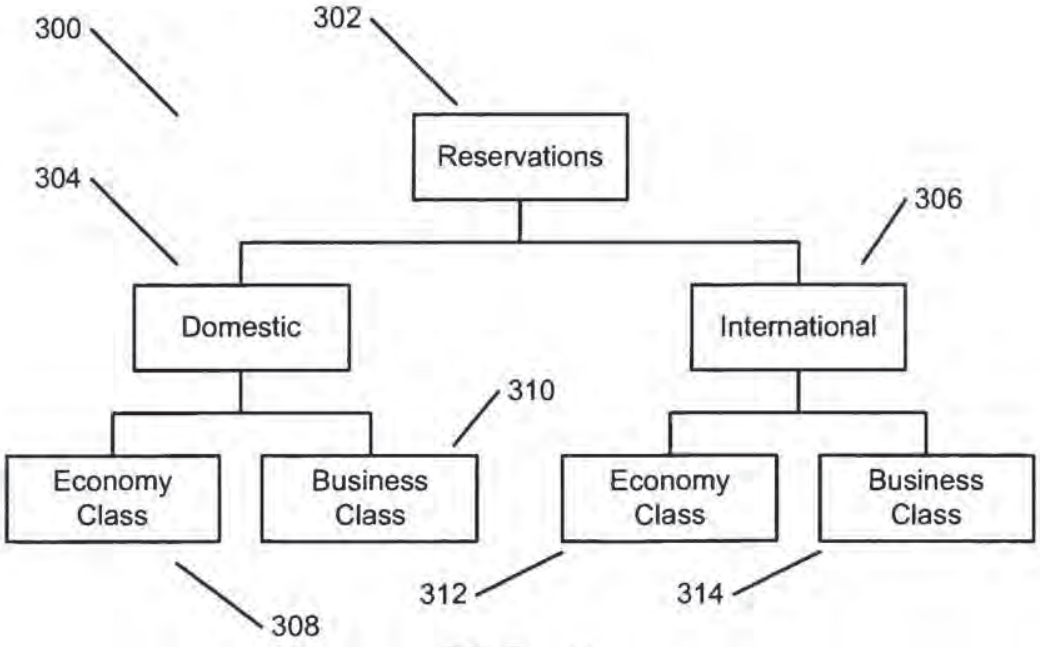

300
302
Reservations
304
306
Domestic
International
310
Economy Class
Business Class
Economy Class
Business Class
308
312
314

FIG. 3

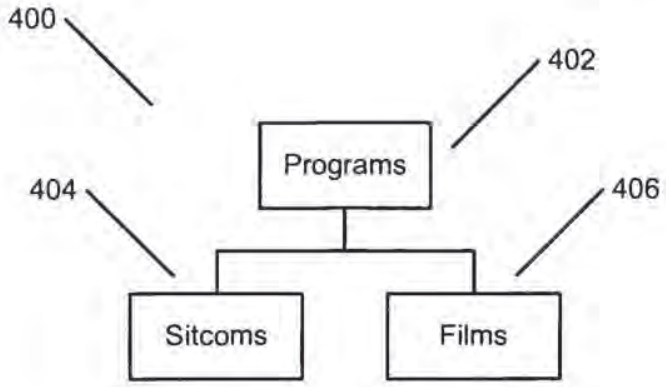

400
402
Programs
404
406
Sitcoms
Films

FIG. 4

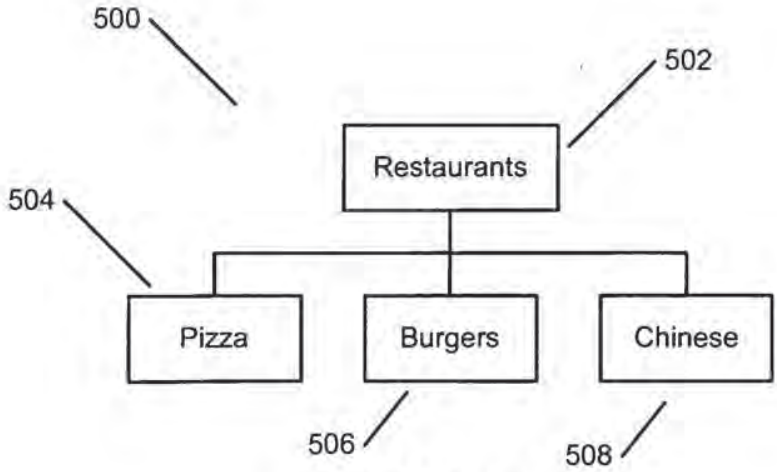

500
502
Restaurants
504
Pizza
Burgers
Chinese
506
508

FIG. 5

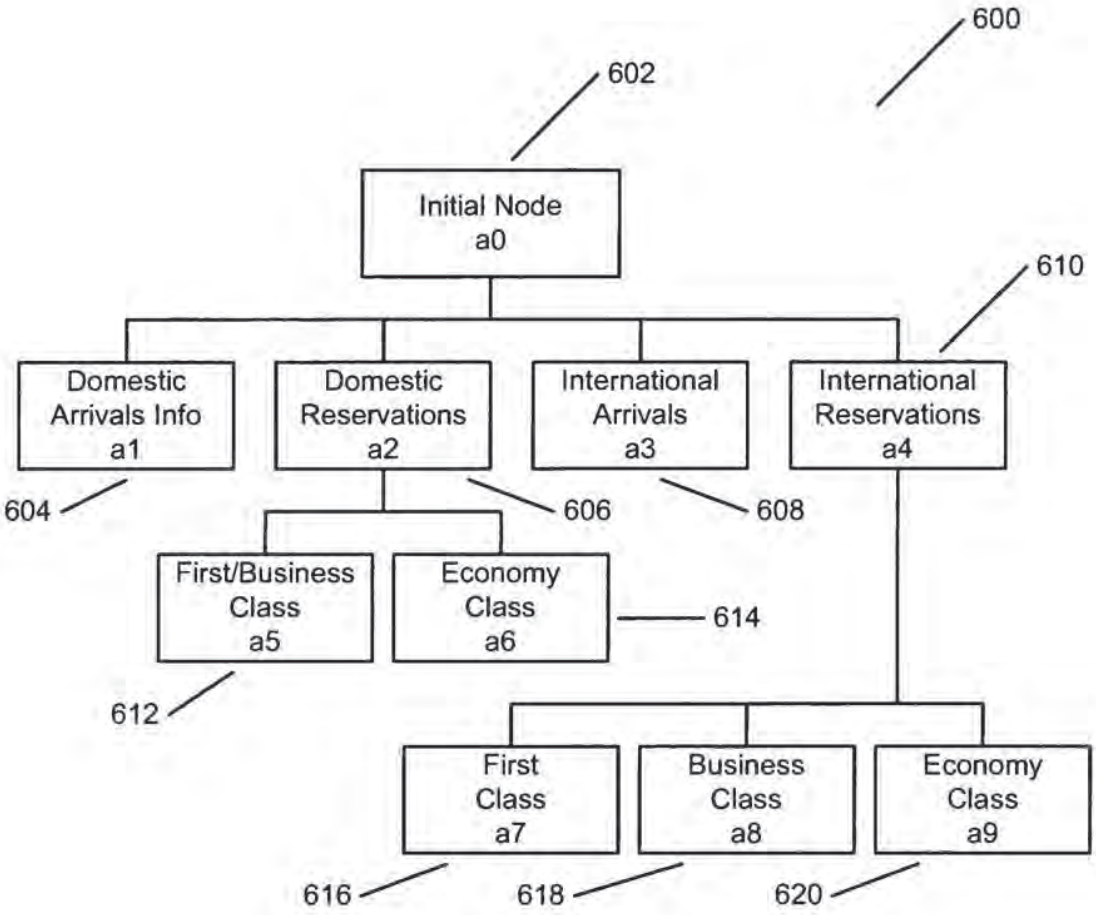

600
602
Initial Node
a0
610
Domestic
Arrivals Info
a1
Domestic
Reservations
a2
International
Arrivals
a3
International
Reservations
a4
604
606
608
First/Business
Class
a5
Economy
Class
a6
614
612
First
Class
a7
Business
Class
a8
Economy
Class
a9
616
618
620

FIG. 6

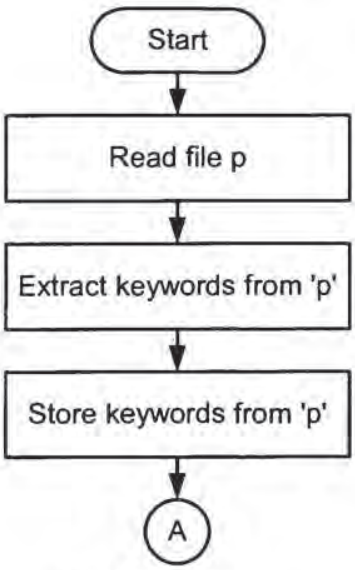


FIG. 7A

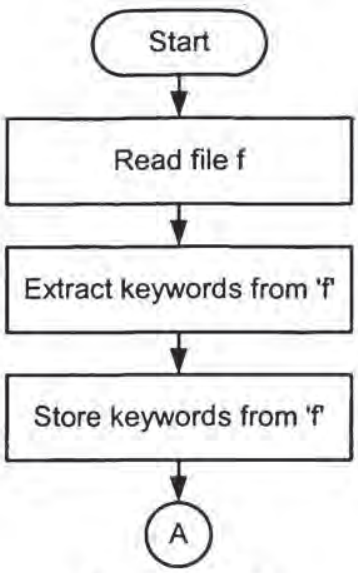


FIG. 7B

FIG. 8

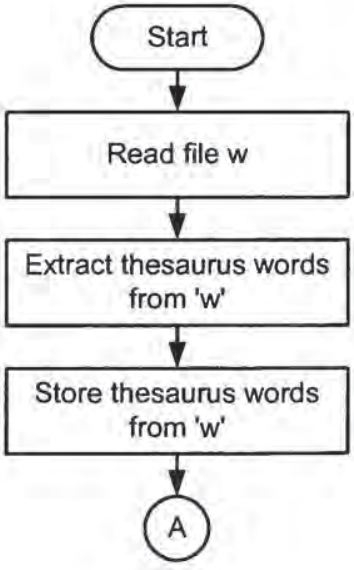


FIG. 9

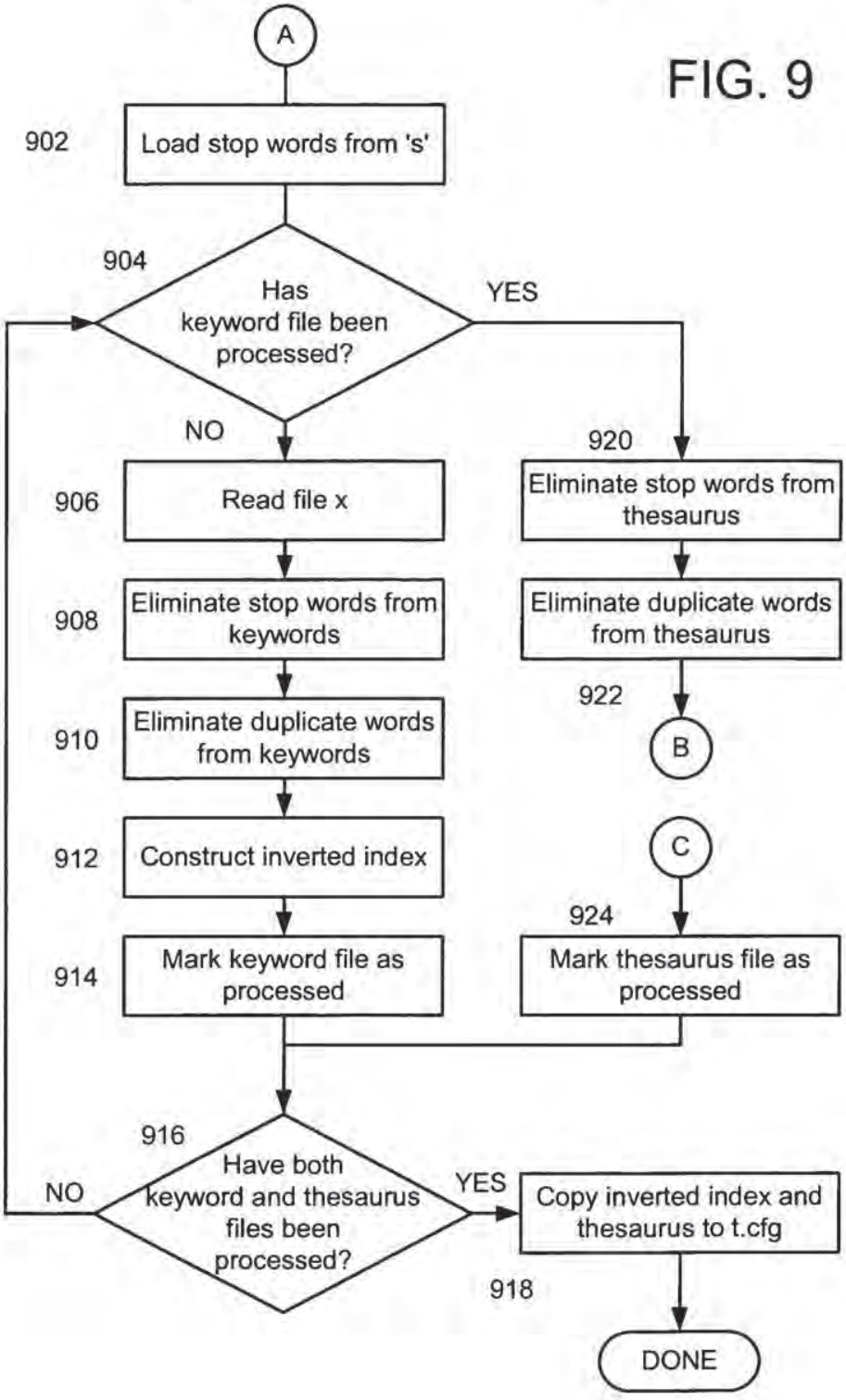

A
902
Load stop words from 's'
904
Has keyword file been processed?
YES
NO
906
Read file x
908
Eliminate stop words from keywords
910
Eliminate duplicate words from keywords
912
Construct inverted index
914
Mark keyword file as processed
916
Have both keyword and thesaurus files been processed?
NO
YES
920
Eliminate stop words from thesaurus
Eliminate duplicate words from thesaurus
922
B
C
924
Mark thesaurus file as processed
Copy inverted index and thesaurus to t.cfg
918
DONE

FIG. 10

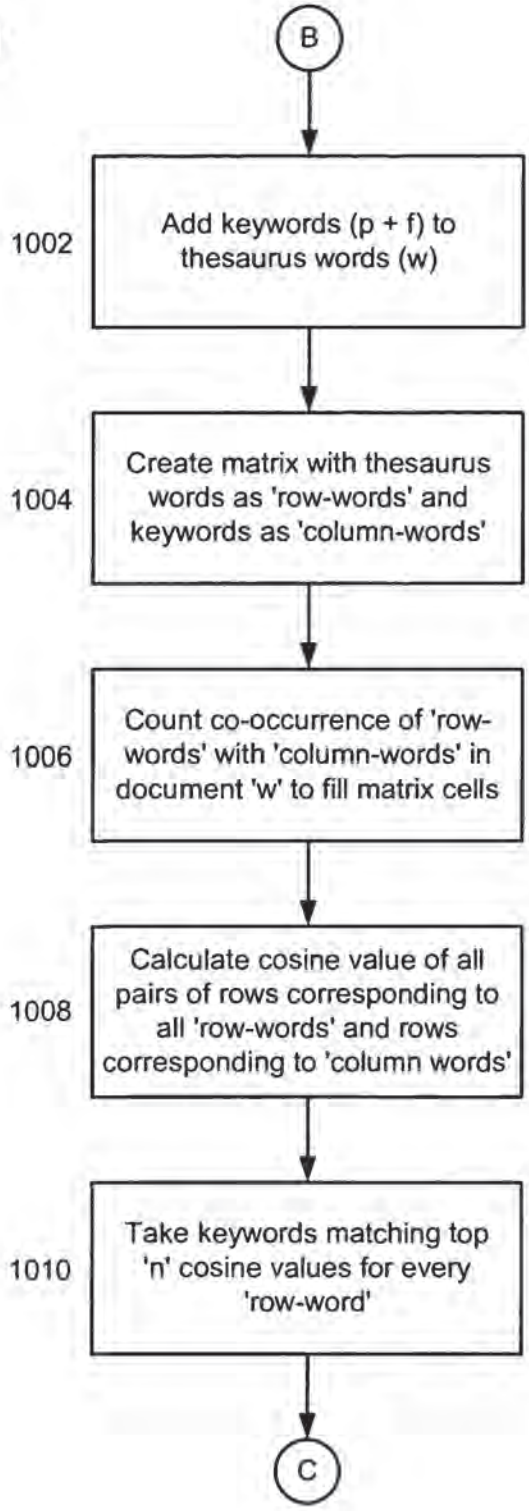
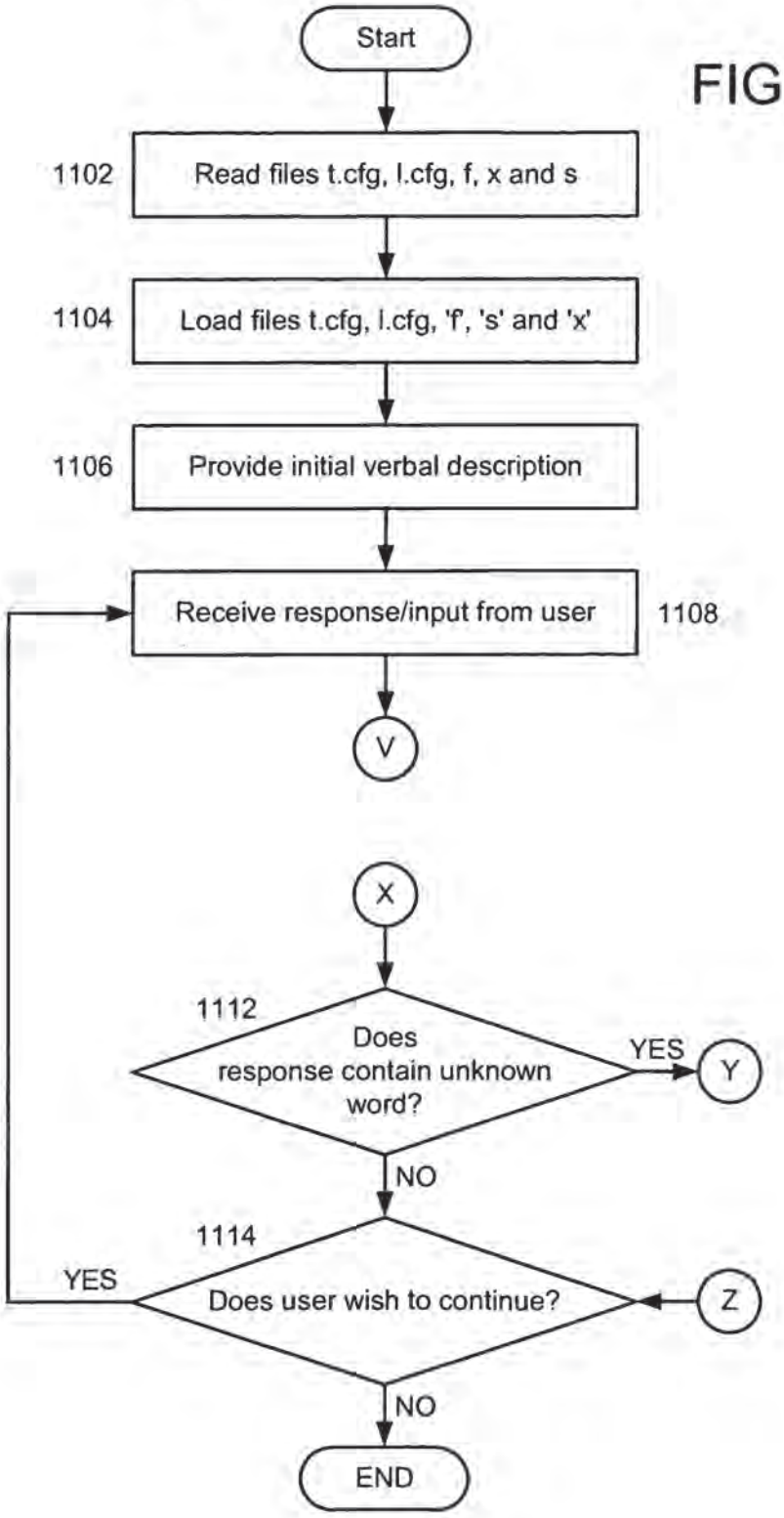

B
1002
Add keywords (p + f) to thesaurus words (w)
1004
Create matrix with thesaurus words as 'row-words' and keywords as 'column-words'
1006
Count co-occurrence of 'row-words' with 'column-words' in document 'w' to fill matrix cells
1008
Calculate cosine value of all pairs of rows corresponding to all 'row-words' and rows corresponding to 'column words'
1010
Take keywords matching top 'n' cosine values for every 'row-word'
C

FIG. 11


Start
1102
Read files t.cfg, l.cfg, f, x and s
1104
Load files t.cfg, l.cfg, 'f', 's' and 'x'
1106
Provide initial verbal description
Receive response/input from user
1108
V
X
1112
Does response contain unknown word?
YES
Y
NO
1114
YES
Does user wish to continue?
Z
NO
END

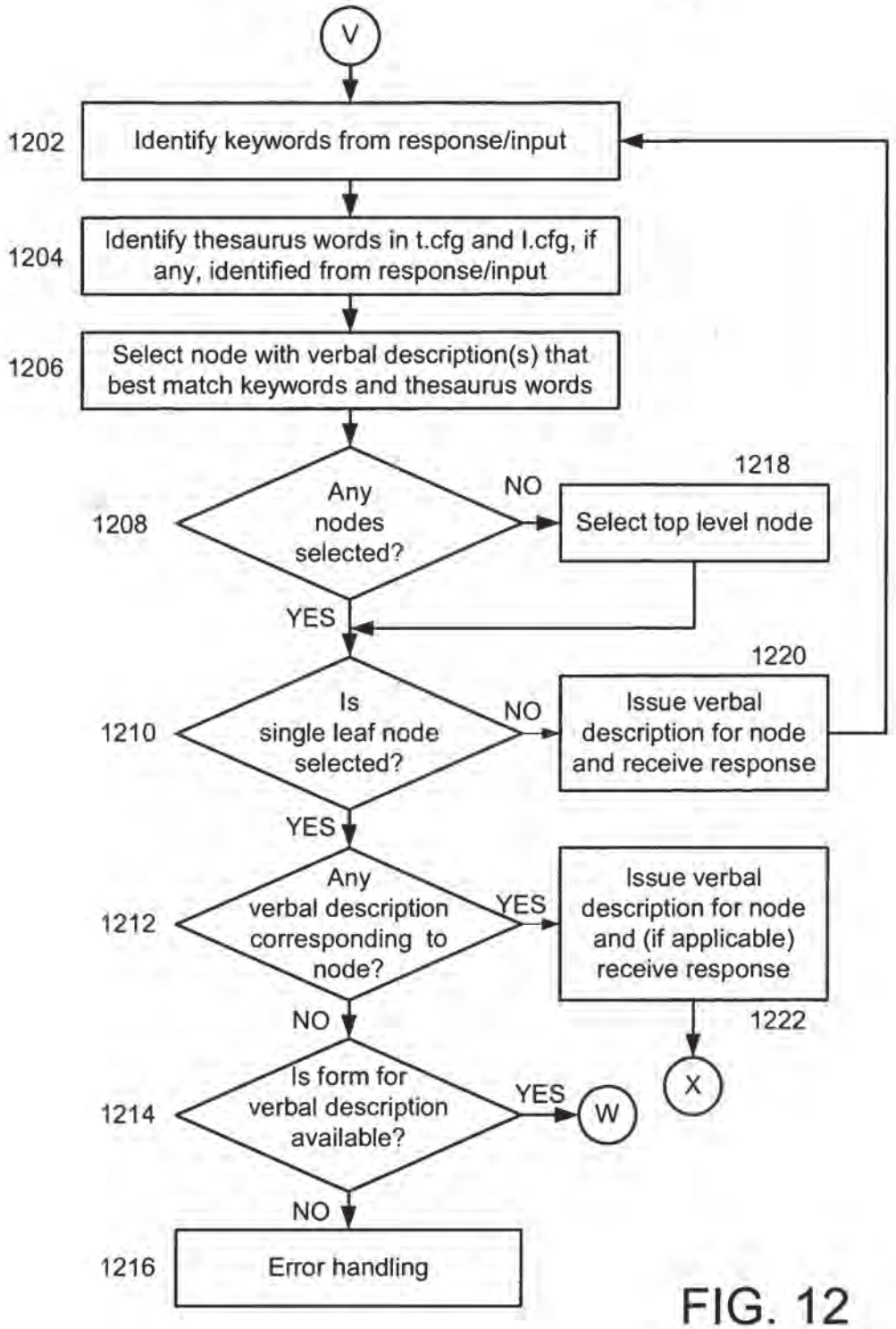

V
1202
Identify keywords from response/input
1204
Identify thesaurus words in t.cfg and l.cfg, if any, identified from response/input
1206
Select node with verbal description(s) that best match keywords and thesaurus words
1208
Any nodes selected?
NO
1218
Select top level node
YES
1210
Is single leaf node selected?
NO
1220
Issue verbal description for node and receive response
YES
1212
Any verbal description corresponding to node?
YES
Issue verbal description for node and (if applicable) receive response
1222
NO
1214
Is form for verbal description available?
YES
W
X
NO
1216
Error handling

FIG. 12

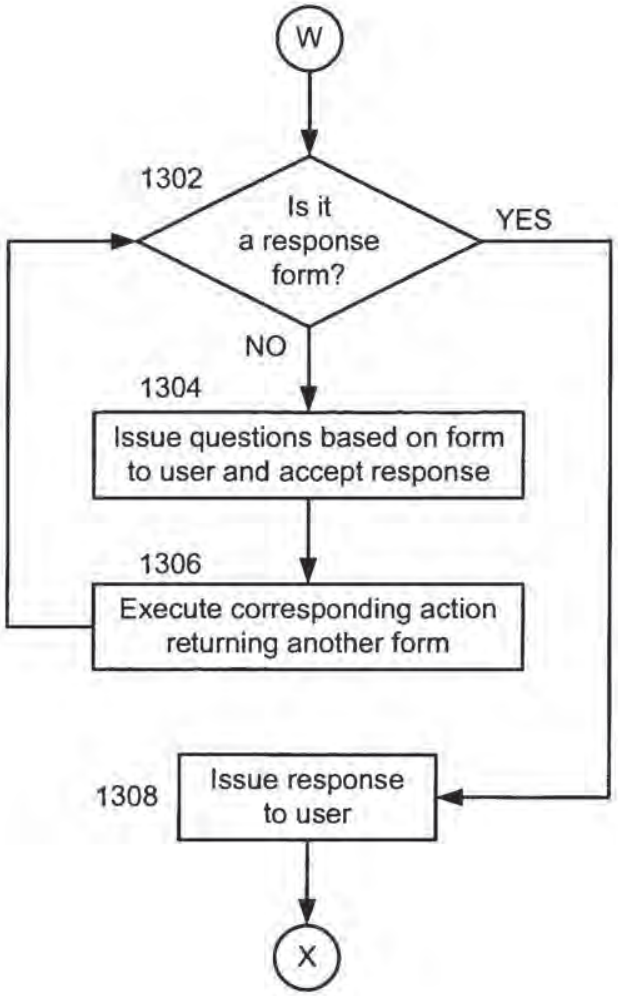

W
1302
Is it a response form?
YES
NO
1304
Issue questions based on form to user and accept response
1306
Execute corresponding action returning another form
1308
Issue response to user
X

FIG. 13

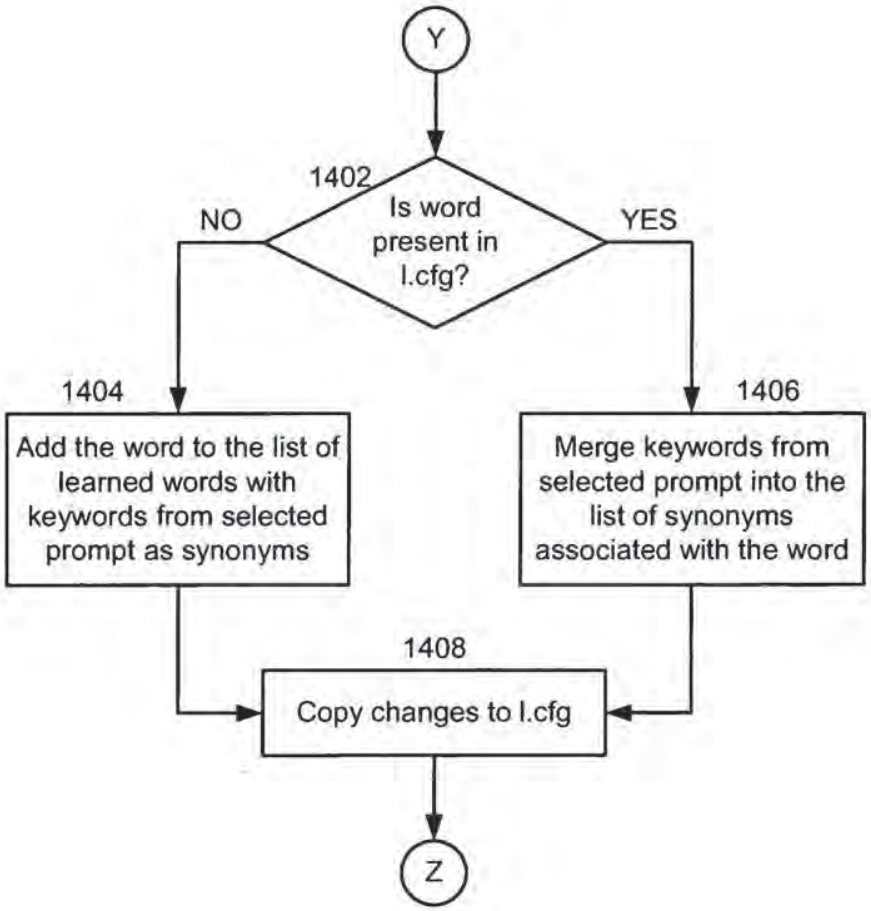

Y
1402
Is word present in l.cfg?
NO
YES
1404
Add the word to the list of learned words with keywords from selected prompt as synonyms
1406
Merge keywords from selected prompt into the list of synonyms associated with the word
1408
Copy changes to l.cfg
Z

FIG. 14

The United States
Patent and Trademark Office

UNITED STATES

PATENT APPLICATION

FOR

NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION PROCESSING SYSTEM

Inventor(s):

Prashant Parikh
Stanley Peters

PATENT
Docket No.: 4428-4001

NAVIGATION IN A HIERARCHICAL STRUCTURED TRANSACTION PROCESSING SYSTEM

FIELD OF THE INVENTION

The present invention relates to information processing and, more particularly, computer based transaction processing.

NOTICE OF COPYRIGHT RIGHTS

A portion of the disclosure of this patent document, particularly the Appendix, contains material that is protected by copyright. The copyright owner has no objection to the facsimile reproduction of the patent document or the patent disclosure as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

In everyday life, networks of choices set forth in a particular order or hierarchy are encountered with increasing frequency. Usually, it is desired to traverse the network in the most efficient manner possible to accomplish a particular goal.

In modern mathematics, graph theory is used to study networks of hierarchical choices. The hierarchical networks can be represented as a graph structure. Graph theory finds practical applications in chemistry, computer science, economics, electronics and linguistics.

A graph structure is a collection of points, called "vertices", and a collection of lines, called "edges". Each edge joins a pair of vertices or a single point to itself.

A simple example of a network represented by a graph structure is a road map. The vertices represent towns or cities. The edges represent the roads that connect the towns and cities.

Another type of network familiar to anyone who has a telephone is an automated telephone voice response system, such as commonly utilized by many large companies, to direct incoming calls to particular individuals or departments or to assist the caller in performing a transaction, such as making a purchase.

That type of telephone network can also be represented as a graph structure. When the system answers an incoming call, it transmits a verbal description or prompt to the caller: "If you would like to speak to Harry, press 1; if you would like to speak to Fred, press 2". (In general, we will use "verbal description" to mean a set of words relating to the subject matter whether presented audibly or in written form. The verbal descriptions may range from a few words to an entire document worth of text). A first vertex on the graph represents the initial prompt, which a caller hears upon reaching the telephone response system. If the user's response is pressing 1, calls are directed along a first edge to Harry, represented by a second vertex. If the response is pressing 2, the call is directed along a second edge to Fred, represented by a third vertex. Then, if the chosen person is not available, the caller is asked whether the caller wishes to leave a message. If the response is positive, the caller is directed along another edge to the selected person's voice mail, which would be represented by another vertex of the graph.

In general, whether for a telephone response network or for any other application representable by a graph structure, the caller or user of the system will have some goal. By "goal" we mean a combination of transactions and information accesses which the user seeks to accomplish. By "transaction" we mean an operation performed electronically with a user. In general, there will also be a combination of vertices or nodes in the graph that best represent or are closest to the goal the user is trying to accomplish. We call these vertices the "goal vertices".

For the user, the object in navigating the graph is to get from the first vertex to the goal vertices. If this is not done as quickly and efficiently as possible the user may become frustrated and give up. Moreover, as the number of possible choices or nodes in the network becomes larger, the number of possible pathways between the first vertex and the goal vertices multiplies rapidly. Therefore, the ability to reach the goal vertex can become more difficult, require navigation of an excessive number of choices or nodes, or discourage a user before the goal vertex is even reached.

SUMMARY OF THE INVENTION

The present invention creates a method for navigating efficiently and naturally through a series of choices to obtain information, perform transactions, or accomplish some similar goal. The invention is implemented in a programmed computer that has a hierarchically configured decisional network that must be navigated as part of the processing and is constructed to accept inputs or data and process them in a manner that facilitates navigation of the network vertices more efficiently.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an example graph representing a simple, generic hierarchically arranged transaction processing or decisional system suitable for use with the invention;

FIG. 2 is an example portion of a graph used to illustrate jumping among nodes in accordance with one variant of the invention;

FIG. 3 is an example portion of a graph in a simple interactive voice response ("IVR") system used to illustrate grouping in accordance with one variant of the invention;

FIG. 4 is an example portion of a graph in a simple interactive television program listing used to illustrate another variant of the invention;

FIG. 5 is an example portion of a graph in a simple geographic information system used to illustrate a further variant of the invention;

FIG. 6 is an example portion of a graph for a simple automated voice response system used to illustrate a more complex variant of the invention;

FIGS. 7A, 7B, and 8-10 are collectively a flowchart illustrating an example setup process for use in accordance with an example implementation of one variant of the present invention; and

FIGS. 11-14 are collectively an overall flowchart illustrating an example process in accordance with a further variant of the present invention.

DETAILED DESCRIPTION

In graph theory, mathematicians refer to a "path" from one vertex in a graph to another specified vertex in the graph as consisting of a sequence of edges that connect the vertices between the first vertex and the final vertex. If the path contains an edge sequence that is "closed", meaning that it loops back on itself, the path is called a "circuit" or a "cycle". A graph structure is considered to be "connected" if there is at least one path connecting every pair of vertices.

Our invention is particularly applicable to transactional processing as applied to instances where graph theory can be used to represent the transactions as a set of options and when the options are structured according to a connected graph that contains no circuits. We call such a graph a "tree". We use the term "menu tree" for a network that provides a "menu" of

options, typically presented as verbal descriptions, to assist a user in making a series of choices through which he or she is able to accomplish one or more of his or her information access or transaction goals. Informally, a "menu tree" can be regarded as a series of vertices in a hierarchy or ordered pattern, arranged in rows of increasing numbers of vertices. More precisely, a "menu tree" can be represented as a "tree" in which (i) the vertices are all the options provided anywhere in the "menu tree", plus a first vertex, (ii) every vertex except the first vertex, i.e., every "option vertex", is associated with the verbal description (or such other means) by which a "menu" presents that option, (iii) an edge connects the first vertex to each vertex that the first "menu" presents to the user as an option, and (iv) each other vertex is similarly connected by edges to every other vertex that the corresponding "menu" presents to the user as an option. As the number of options increases, so does the length of paths from the first vertex to goal vertices.

In overview, in accordance with the teachings of our invention, the user can navigate the graph or tree in a way that allows them to skip from one vertex to another vertex that may be many rows down the graph or tree and/or where the vertices may not be connected together by an edge. This eliminates the necessity for making many choices.

Particular implementations make it possible to jump laterally from one vertex to another if the navigation enters a wrong branch of the tree or if the user changes his goal. The approach is accomplished through associating each vertex with a verbal description (or prompt), and matching words in users' requests and responses with these verbal descriptions to enable the selection of vertices that may not be directly connected to the user's current location in the graph or tree by an edge.

In some variants, we create a system with the unique ability to learn by incorporating previously unknown words, keyword or synonyms of keywords so that the system modifies itself to thereby increase the likelihood that a user will efficiently and quickly reach the goal.

For purposes of illustration, the invention will be described by way of example, first using a series of simple examples followed by a more complex example of a more detailed and commercially suitable example variant, in the context of a menu-type automated telephone voice response system for a publication, a hierarchical network of the type that is frequently encountered and easily understood that implements a combination of some of the features of the simple examples in order to illustrate how those features can be combined or overlayed.

It should be understood that the present invention is applicable to a wide range of different networks, which can be mathematically represented by graph structures consisting of vertices and edges and should not be considered to be limited to the particular application described. Representative examples of suitable applications for the invention include implementing an enhanced and more efficient "Find" function or file system browser for personal computer operating systems, a navigation system for television program listing, document management or retrieval systems, a "geographic information system" in an automobile that allows location of addresses or business(es) meeting certain criteria, or other devices that incorporate some hierarchical navigation aspect as part of its operation.

In order to more fully understand the invention, various independent aspects are now presented below by way of simple illustrative examples. In this manner the teachings of the invention can be understood in a way that makes it possible to use, overlay and/or combine those aspects in a beneficial manner in an implementation of the invention. Depending upon the

particular implementation of the invention, one or more of the aspects may be used together in various permutations and/or combinations, with the understanding that different permutations and/or combinations may be better suited for particular applications or have more or less benefits or advantages than others.

The underlying scenario common to all these basic examples is that there is a hierarchical arrangement to the possible choices that can be illustrated in a form of "tree" structure.

FIG. 1 is an example graph 100 representing a possible hierarchically arranged transaction processing or decisional system suitable for use with the invention. The individual boxes 102 - 120 are referred to as "nodes" and each represents a specific choice or option in the hierarchy. For purposes described in more detail below, each node is arbitrarily uniquely identified in some manner. In the example of FIG. 1, the individual nodes 102 - 120 are numbered 1 through 10 starting from the top node 102 in the hierarchy.

Each "node" is associated with exactly one verbal description, for example in the case of an airline system, a verbal description relating to some aspect of the reservation process. Each such description contains "key" words that are deemed to be of importance and other words that can be disregarded. For example, one node may have the associated verbal description "Would you like to make a reservation?" In this description, there is only one "key" word – "reservation" deemed important, so all of the other words in the description can be ignored.

A level in the hierarchy below that one may be used to obtain further narrowing information, for example, using the verbal description "Is the reservation for a domestic or international flight?" In this description, the terms "domestic" and "international" are "key" words. Similarly, the word "flight" could be a "key" word, for example, for a system that

involves not only airline travel but also rail and/or cruise travel or it could be an "ignored" or stop word for a purely airline related system because it has minimal meaning in that context. Again, the other words can be ignored as well.

The unique identification of each node allows the creation of a list of all the key words and their associated nodes so that, if a key word is duplicated in two or more nodes, it need only be listed once. For example, a hierarchical tree related to "pens" might have nodes for ball-point pens, fine point pens, medium point pens, fountain pens, felt-tip pens, quill pens, erasable pens, etc. By using this approach, one could list the keyword "point" once, but associate it with each of the nodes where that keyword appears by using the unique identifier for each node where the term appears.

In this manner the keywords are obtained from the collection of available descriptions found in the particular application in which the invention will be used. In addition, each particular node where the keyword appears is associated with the keyword. Thus, with respect to the pen application above, the keyword "point" might appear in nodes 2, 3, 6, 7, 13 and 15. Similarly, the keyword "erasable" might appear in nodes 3, 4, 5, 6 and 22. An index, as described more fully below, associating these keywords with the nodes containing them is then created, for example:

point: 2, 3, 6, 7, 13,15
erasable: 3, 4, 5, 22

By making use of these associations the "tree" can be negotiated by allowing presentation of relevant verbal descriptions for the nodes associated with a term, irrespective of where in the

hierarchy they are, thereby causing a "jump" to a particular node without necessarily traversing the tree in the rigid hierarchical manner.

Various examples will now be presented to illustrate certain concepts related to the invention. It should be understood that while these examples are presented in the context of things and likely experiences of ordinary people, the same approach can be applied to other forms of transaction processing including navigating through hierarchically nested data files in a computer system, pattern analysis or image processing, etc. the term "transaction" as used herein relating to traversal through a hierarchy to a goal, not mathematical calculation per se.

Moreover, the specific formats used and presented in these examples are purely for illustration purposes. It should be understood that that other techniques for interrelating data, such as hash tables, direct or indirect indexing, etc. can be substituted in a straightforward manner. Thus, for example, the relationship between the word and a node could be configured such that the location of the word in a list as the "n-th" item could be used as an index into another list containing the nodes correlated to the list. A similar approach could be used for the thesaurus, the important aspect relative to the invention being the relationship among certain words and the node(s) in which they occur and, where applicable, the relationship between certain words and "synonyms" for those words, not the data structure or its form or format whereby that information is kept or maintained.

Example 1

Example 1 illustrates, in simplified form, how an index is used to jump among nodes with reference to FIG. 2. In this example, the hierarchical tree 200 represents a portion of a more

complex tree specifically involving possible decision relating to fruit and a decision between two specific types of fruits, an apple and an orange.

In prior art hierarchical trees, navigation of this graph 200 would necessarily involve going through the "fruit" node 202 in order to reach the "apple" 204 or "orange" 206 nodes. As a result, assuming this simple tree was part of a larger tree for an on-line supermarket that prompted the user for what they wanted to purchase, the exchange would be both rigid and time consuming. For example, in response to a prompt "What do you want to purchase?" if the response was anything other than "fruit" traversal to the "fruit" node 202 could not occur. At the point in the tree that would lead to the "fruit" node 202, neither apple nor orange would be an acceptable response.

In accordance with the invention, assuming the only relevant keywords for that portion of the tree were "fruit", "apple" and "orange", an inverted index would be created that includes an association of "Fruit" with the top node 202, "Apple" with the bottom left node 204, and "Orange" with the bottom right node 206. As shown above, that association can be created using node identifiers, in this example, the node identifiers 1A01, 1A02 and 1A03 are arbitrarily assigned and used. Thus, the information can be stored in a file, for example, as follows:

Fruit, 1A01
Apple, 1A02
Orange, 1A03

Accordingly, to navigate the system 200, when a response to a verbal description is provided by a user, possible keywords are identified in the response and used to search the index and identify any node to which the response may be directed, irrespective of the hierarchy. Thus, a user response of "an orange" to a verbal description located above the "fruit" node 202 in

the hierarchy, for example, "What would you like to buy today?" would cause the system to identify "orange" as a key word from the response, search the index, and directly identify node 1A03 (206) as the node whose verbal description should be presented next, thereby avoiding the need to traverse intervening nodes, for example, through the "fruit" node (202) 1A01, at all. This illustrates an example of a simple jump according to the invention.

Example 2

Having illustrated a simple "node jump" a more complex (and likely) scenario can be shown. In this example, the Example 1 graph of FIG. 2 applies, but relevant portion of the index is as follows:

Fruit, 1A01
Apple, 1A02, 2F09
Orange, 1A03

As a result, there are two nodes relevant to the keyword "apple" one being the node 204 in the portion of the graph shown in FIG. 2 and one in the node uniquely identified as 2F09 located somewhere else in the hierarchy (not shown).

In this example, a user response containing the keyword "apple" would identify nodes with identifiers 1A02 and 2F09. In this case, and unlike the prior art, the verbal descriptions from both nodes would be presented to the user, likely in alternative fashion. Thus, if the user did not want an apple, they wanted apple cider, node 2F09 might be more appropriate because it is part of the "drinks" portion of the overall hierarchy.

Thus, presenting the user with the verbal description from both nodes would likely result in a jump to the portion of the graph nearer to node 2F09 since it is closer to the user's goal thereby speeding up the process and avoiding potentially confusing or frustrating the user.

Example 3

While the verbal descriptions associated with various nodes will generally be chosen to accurately represent the node, in accordance with certain variants of the invention, it is possible to create a situation where a user response takes them away from their ultimate desired goal. Nevertheless, by using the teachings of the present invention, the user can often still be brought to their goal quicker than possible with the prior art because the user need not rigidly trace through the hierarchy. This is accomplished by virtue of the "grouping" aspect inherent in some implementations of the invention.

This example illustrates the "grouping" aspect using a simplified graph 300 representing a portion of an airline reservation system as shown in FIG. 3.

In particular, the graph of FIG.3 can be thought of as part of a very simple interactive voice response ("IVR") system.

As described above, each node is uniquely identified, for example, by the numbers 1 through 7 and the identified terms "Reservation", "Domestic", "International", "Business Class", "Economy Class" are deemed the relevant keywords. Note, there is no requirement for a the "keyword" to be a single word, in some implementations, keywords could be single words, phrases of two or more words, or even some other form of information like a specific data pattern.

Again, an inverted index is created as described above associating those keywords with the nodes, in this case:

Reservation, 1
Domestic, 2
International, 3

Business Class, 4, 6
Economy Class, 5, 7

Assuming that the top node is assigned the number 1, its two child nodes (Domestic and International) are assigned the numbers 2 and 3, and the grandchild nodes (i.e. at the lowest level in the hierarchy) have been assigned numbers 4, 5, 6, and 7 taken from left to right each node can be uniquely located. Note that the last two entries in the inverted index are each associated with two nodes, 4 and 6 in the first case, and 5 and 7 in the second.

Using the above, the concept of grouping of nodes from different parts of the graph (i.e. nodes that are not siblings or nodes that do not have a common parent) can be explained.

Presume that the response to a verbal description presented as an initial query of "What do you want to do?" was "Make a business class reservation." In this case there are two keywords present, "reservation" and "business class".

Depending upon the particular implementation, as noted previously, the verbal descriptions associated with each identified node could be presented together or in sequence. Alternatively, and as is the case here, a set of rules can be established, for example, such that if an identified node is a sub-node of another identified node, only the verbal description of the sub-node(s) is provided because of inherent redundancy. Thus, since both "business class" nodes 310, 314 are sub-nodes of the "reservation" node 302, the verbal description associated with the "reservations" node can be suppressed if it can be determined that business class necessarily implies reservations.

In this example, a search of the inverted index would identify nodes 4 and 6 (310, 314) from different parts of the tree are associated with the keywords in the query, and thus the

system, in presenting the verbal descriptions from each, in effect, alters the tree structure and groups these nodes in the result. Thus, the combination of result nodes presented depends upon the user query or response, not that predetermined by the graph structure itself.

Of course, the goal would still not be reached because of the ambiguity caused by "Business Class" being under both "Domestic" and "International". However, that ambiguity can be handled by suitable wording of the following verbal descriptions and whether they are combined or provided sequentially or by other nodes.

Example 4

A persistent and further drawback present in the prior art is the inability to operate if any term other than the specific allowed terms are provided. Thus, in an IVR of the prior art, providing anything other than the recognized term(s) will likely result in meaningless repeat of the same inquiry by the IVR or an error.

Advantageously, the teachings of the present invention allow for construction of a more flexible system than available in the prior art. Specifically, we can incorporate a thesaurus to accommodate synonyms for the keywords.

Example 4 illustrates the addition of a simple thesaurus as an aspect of a system so that a synonym of a keyword may also be used by the system to jump to the desired nodes in the graph. Example 4 is discussed with reference to a portion 400 of an interactive television program listing system as shown in FIG. 4.

Such a system implementing the invention will allow a user to speak to or interact with a device to look for programs of his choice by time slot, genre, favorite actor or actress, etc.

This example, as with the other examples above, use an inverted index, in this case one where each node 402, 404, 406 is uniquely identified by a string of six characters, the portion of which corresponding to FIG. 4 is shown as follows.

Programs; acgyct
Sitcoms; ifgnxh
Films; vnymos

Since a common synonym for "Films" is "Movies" a thesaurus can be created associating the two. Depending upon the particular implementation, thesaurus terms to be equated to the keywords can be taken from a standard thesaurus or can be custom created for the particular application. In addition, the equating of terms can be done in any of a myriad of different ways, the exact implementation details of which however re irrelevant to the invention, but a few representative examples of which however are contained herein for purposes of illustration.

In one example case, the equating can be done on a purely word basis. For example, a file can be constructed such that one or more single word synonyms are directly associated with an index word, for example as follows:

Movies, Flicks – Films

Alternatively, the synonyms can be equated with the node identifier(s) corresponding to the index term, for example as follows:

Movies, Flicks – vnymos

In the former case, the system would still have to search the index after the thesaurus has provided the proper index term(s). In the latter case, the thesaurus provides a direct link to the respective node(s) so that re-searching is not required.

In the system of Example 4, a user who provides the input "Movies" would cause the processing to occur as follows.

The system would search the inverted index of keywords and fail to locate "Movies" as a keyword. As a result, it would search the thesaurus and find that the word "Movies" is a synonym that can be correlated with a keyword. At this point, depending upon the particular thesaurus, it would either return to the inverted index and search using the synonym keyword "Films" and return the result as the node 406 identified by "vnymos", or go directly to the node 406 identified by "vnymos" based upon the thesaurus entry.

Of course, it is possible (and likely) that in actual usage a synonym will be associated with more than one keyword. For example, "Comedies" may be associated with both the keywords "Sitcoms" and "Films", resulting in, for example, the following entry in a thesaurus:

Comedies – Sitcoms, Films

In this case, a search for "Comedies" would result in the system identifying that the synonym was associated with nodes 404, 406 for both "Sitcoms" and "Films", and it would return both terms or node identifiers corresponding to the two keywords as the result.

Example 5

Advantageously, the thesaurus concept can be extended further so that an initially unknown word (i.e. a word that is neither a keyword nor a thesaurus word) can be learned by the system and added to a thesaurus for future use.

This example is described with reference to FIG. 5 which is a portion 500 of a larger system graph as part of a very simple "geographic information system" found in some automobiles, kiosks and elsewhere today. Such a system enables a user to, among other things,

identify and get information about different locations in an environment. For example, information about particular types of restaurants in an area.

In this example, the inverted index for the portion 500 shown in FIG. 5 could look as follows:

Restaurants, 1
Pizza, 2
Burgers, 3
Chinese, 4

A user issues the following query to the system "fast food" in order to find a quick meal.

The system's search of both the index and thesaurus would result in the "term", in this case a phrase, not being found in either. In this case, it is an unknown phrase, and the system has to learn the "meaning" of the term.

To do this, the system first offers the verbal description from the top level node(s) 502 to the user – in this example, just "Restaurants". The user presumably provides a positive response. (Of course, in a real system, it is possible and likely there are more top level nodes than just one. In that case, the user would be offered two or more of these nodes, and would have to select "Restaurants" to match his intended request.)

Continuing on, once the user has responded affirmatively, the system moves down the tree and offers the verbal description from each of the child nodes: "Pizza" (504), "Burgers" (506), and "Chinese" (508). Presuming that the user picks "Pizza", the transaction interaction would look something like this:

User: Fast food

System: Restaurants?

User: Yes

System: Pizza, Burgers, or Chinese?

User: Pizza

At this point, the system has "learned" for the time being that it can equate "fast food" with "pizza" and can add "fast food" as a synonym to "pizza" in the thesaurus.

This user, who first used the unknown term "fast food", had to trace a path down the tree. However, now the system is able to associate "pizza" with "fast food" and create or add a thesaurus entry to reflect this association, for example as follows:

Fast food – Pizza

Thus, the system has learned a meaning of the initially unknown term "fast food" and has added it to the thesaurus for future use.

As a result, a subsequent uses of the same term "fast food" will enable the system to jump directly to the "pizza" node 504.

Example 6

This example illustrates how additional meanings for an existing thesaurus term or phrase can be learned by the system for future use, whether the existing thesaurus term or phrase was an original thesaurus term or one previously learned with continuing reference to FIG. 5.

At this point, the inverted index is unchanged as:

Restaurants, 1
Pizza, 2
Burgers, 3
Chinese, 4

Additionally, presume the following entry now exists in the thesaurus.

Fast food – Pizza

Suppose a new user now issues the query "fast food" as above, but with "Burgers" rather than "Pizza" in mind.

Based upon the thesaurus, the system would go directly to the "Pizza" node. However, the user will reject "Pizza", having "burgers" in mind. By rejecting the "Pizza" node 504 description, the user indicates that the "Pizza" node 504 is not of interest. The system is therefore configured with a further set of rules, in this case one in which the system goes up in the hierarchy to a higher node, the top node 502 in this portion of the example, and provides the verbal descriptions for the other nodes 502, 504, 506, 508 so as to cause a tracing down the tree. This can be illustrated by the following "dialog":

User: Fast food

System: Pizza?

User: No

System: Restaurants?

User: Yes

System: Pizza, Burgers, or Chinese?

User: Burgers

This time, although this user has had to trace through at least a portion of the path from a higher-level node 502 of the tree 500, the system has learned yet another meaning for "fast food". It now adds this meaning to the earlier entry in the thesaurus, for example as:

Fast food – Pizza, Burgers

It has now learned two meanings for future use. If a user were now to issue the query "Fast food", the system would respond with the verbal descriptions from the nodes 504, 506 corresponding to both Pizza and Burgers.

Thus, the system can keep learning new meanings of terms based on the intended meanings of users "deduced" from the interactions between users and the system.

Of course, the nature and extent to which the system will incorporate synonyms and/or keywords in a continual learning process will not only depend upon its construction and rules, but also on the quality of the original thesaurus and the quality of the initial inverted index. In addition, where in the tree the system jumps if the user rejects the initial meaning(s) offered by the system can be handled different ways in different implementations.

For example, the system can always jump to fixed ancestor(s) (either the top node or a parent or some ancestor(s) at an intermediate point) or a fixed level (e.g. halfway from the top). This approach has the advantage of being simple to implement, but it has the problem of inflexibility because it may be relatively efficient for certain graphs and associated verbal descriptions, but not for all. For example, if two or more nodes' verbal descriptions are offered and rejected, the relevant node selected would have to be common ancestor(s) of the offered nodes. In other words, with reference to Example 6 which is part of a larger tree, going up to the "Restaurants" node 502 would mean going to the parent of the "Pizza" node 504 rather than all the way to the top in the larger tree containing the portion 500 shown.

A more flexible alternative uses the information recorded in the thesaurus to find every synonym for "pizza" in the thesaurus and collect all the other keywords associated with those synonyms. Then the system would search the inverted index to identify all the nodes associated

with these other associated keywords and identify the most common ancestor of all of those nodes and go to it. By using the information in the thesaurus in this way the system makes use of known properties of the one meaning of "fast food", which is "Pizza", to construct an intelligent hypothesis about where the other meanings of "fast food" might lie in the graph. This allows the user to reach another meaning with the least effort and allows the system thereby to learn what the new meaning of "fast food" is more efficiently.

Example 7

Of course, just as it may be desirable to create implementations to add meanings to the thesaurus, it may be equally or more desirable to cause an existing meaning for a thesaurus word to be dropped, for example, due to relative lack of use. This process is described with continuing reference to FIG. 5 and the associated inverted index, particularly with respect to the thesaurus entry resulting from the most recent example.

Fast food – Pizza, Burgers

In this example, presume that there have been several uses of the query "fast food" and that the user(s) issuing these queries have almost always selected "Burgers" and almost never "Pizza".

In accordance with another implementation of the invention, the system is constructed to track the frequency of use of a particular term in the thesaurus. Depending upon the particular implementation, the tracking can be done for all entries in the thesaurus, for only those added as part of the "learning" process, or for some specified combination thereof.

In addition, some specified criterion is used to determine when, and which terms, if any, should be removed from the thesaurus. Depending upon the particular implementation the

criterion can be based upon usage relative to time, usage of a particular term relative to some other term(s), term usage relative to overall thesaurus usage, or simply elimination of all added terms not used since the last purge.

Thus, presuming that the system has kept track of the frequency of use of different meanings of "fast food", and that "Pizza" does not meet the criterion for a sufficiently high frequency, the meaning "Pizza" can be dropped as a synonym for "Fast food" and the entry (after purging) would look as follows:

Fast food – Burgers

Thus, a further enhanced implementation can be constructed so the system is dynamically updating the thesaurus, either adding meanings or dropping meanings for existing and/or initially unknown words.

Example 8

A further advantage to the invention is that, in some implementations, it can be configured so that, when there are multiple relevant nodes to be presented, an associated ranking can be used to determine the type, method or order of presentation. For example, the ranking can be based upon the frequency of use of particular nodes, which is tracked in these implementations, so that the most frequently selected or used nodes are presented first, more prominently, or in a particular manner.

For example, this can be illustrated by continuing from Example 7, where the thesaurus entry was as follows:

Fast food – Pizza, Burgers

Under the assumption that the system has been tracking the frequency of usage of the "Pizza" node and the "Burgers" node and each has been accessed an identical number of times. When a user enters the query "Fast food", as above, the system presents the user with both the "Pizza" node 504 and the "Burgers" node 506, but because it tracks usage and the usage is the same, it presents them in the order they are listed, i.e. "Pizza" and then "Burgers". However, at this point, the user's selection will cause one entry to have a greater frequency of usage relative to the other entry, for example a selection of "Burgers" will make it have a higher frequency of usage and, accordingly, a higher ranking for the next instance of use.

Thus, the next time the system will be presenting both the "Pizza" and "Burgers" nodes to a user, the "Burgers" node 506 will have the higher frequency of usage and, accordingly, will be presented first, or more prominently, or in some other specified manner because of its ranking. If the frequency reverses with use so that the "Pizza" node 504 outranks "Burgers" node 506, then the "Pizza" node 504 will supplant the "Burgers" node 506.

Example 9

A further variant of Example 8 allows the node rankings to be used to prune the nodes themselves. In this variant, a criterion can be specified, typically zero usage over a long specified period of time, that is used to remove an entire node. This is advantageously made possible because of the system's ability to "jump" among nodes. Thus, it may occur that a node within the tree is never accessed, but a child node of that node is. In some variants therefore, when this state exists for a sufficiently long period of time, the system is constructed to delete that node. It should be understood that, if handled properly, this process will not even affect the "learning" process because, even if no user action ever directly causes the node to be presented,

if the learning process causes the node to be presented the node's access frequency will be non-zero and it will not be "pruned".

In addition, by tracking access frequency on a node basis, a qualitative evaluation of the hierarchical system can be made and visualized. This makes it possible to review the overall hierarchy after some period of time and periodically optimize it based upon the result instead of relying purely upon the dynamic optimization that inherently and naturally flows from use of the teachings of the invention.

Having now described various component aspects of different variants implementing the invention, by way of the above examples, it should be understood that the "jumps" can occur from any node to any node, i.e. vertically and/or laterally and to another node that is higher, lower or on the same "level" as the node from which the jump is made. All manner of vertical and lateral jumps from multiple nodes to multiple nodes are possible.

In addition, it should be understood that in some applications (like document retrieval systems) the verbal description from the identified node may be the one issued whereas, in others (like an IVR system), the verbal descriptions for the children of the identified nodes may be what is presented. Nevertheless, in both cases, the process as described above by way of example will be the same or directly analogous.

Having described the various aspects individually a more commercially suitable example, employing a combination of the above examples, can now be presented with reference to FIG. 6 which illustrates a simplified example of an "interactive voice response unit" (IVR) hierarchy 600 that might be used in the airline industry. Of course, a real menu tree used in an IVR may have any number of nodes from several, up to a thousand, or more. For example, a tree with 4

branches from each node and which has 5 levels uniformly would have 1365 nodes. As shown in FIG. 6, the tree 600 is a hierarchical tree and consists of the following nodes and branches:

Initial start (node a0) 602

domestic flight arrival information (node a1) 604

domestic reservations (node a2) 606

international flight arrival information (node a3) 608

international reservations (node a4) 610

The node 604 identified by a1 is a service node with pre-recorded information.

The node 606 has two child node a 2, first/business class (node a5) and economy (node a6).

The node 608 identified by a3 is service node with pre-recorded information.

The node identified as a4 has three child nodes identified as first class (node a7), business class (node a8), and economy (node a9).

The nodes 612, 614, 616, 618, 620 identified as a5, a6, a7, a8, a9 are all service nodes (i.e. terminal nodes) where a respective customer service representative will interact with the caller.

Of course, a real system may also have a choice at the top level or at each level for a live operator and may even have a choice to go back to the previous menu.

Even for such a simple example, in a traditional interactive voice response system, the caller would have to listen to several choices and then traverse a path down to a service node. Someone interested in business class reservations on a domestic flight would have to traverse the path (a0, a2, a5) for example. This involves listening to multiple choices at each level of the tree (e.g. first a prompt at a0, then four prompts offering a1, a2, a3, and a4 at the next level, at which the caller would choose a2, and finally two prompts offering a5 and a6, at which level the caller

would choose a5 and then wait for the operator) and then making a choice by pressing an appropriate number on the telephone dial pad or alternatively saying the appropriate number. In certain cases, he may make a mistake: he may choose international reservations when he is interested in domestic reservations or something similar (simply by pressing the wrong number on his touch-tone telephone or saying the wrong number). If he does, then he has no choice but to disconnect the phone and redial the number (or if the system has a backtracking option, then he can backtrack, but even here he has wasted valuable time).

In contrast, in accordance with a system implementing the invention, the caller would be able to say what he was looking for (e.g. "I want to make a domestic business class reservation") and the system would identify and respond with the appropriate node 612 (e.g. a5 in this case or the relevant customer service representative directly). In other words, it would enable the caller to skip to the correct node(s) without having to trace through the entire path. If the user makes a mistake, he could ask for something different wherever he finds himself in the tree, and skip laterally or vertically to his preferred choice.

The system implementing the invention can further include an option that the entire transaction (e.g. the making of the reservation) would be carried out through natural language interactions with the system without the intervention of a human customer service representative. In other words, all the details of his domestic reservation are obtained by the system and the system updates a database accordingly and issues whatever commands are required (e.g. the mailing of a ticket) to be carried out by some human representative later.

While it is true that some more advanced interactive voice response systems available today allow for natural language interactions, they are highly constrained natural language

interactions with relatively little or no intervention by a human operator. However, unlike with systems using the invention, those systems still require direct path traversal through the hierarchy (i.e. jumping to non-connected nodes is not contemplated or possible, let alone allowed). Moreover, such systems still typically use a limited list of keywords, which the caller is required to use to correctly traverse to the next connected node.

In contrast, variants of a system implemented in accordance with the invention can incorporate an automatically generated or updated thesaurus, which greatly expands the range of words or terms a caller can use. In these variants, the user is not restricted to parroting the highly constrained script as required by other interactive voice response systems, nor is the user limited to traversal to a connected node. In these more complex implementations of the invention, a system can be constructed that is able to learn new words or terms that it may not have understood the first time. For example, if a user asks for "coach class" and the system does not have the word "coach" or the phrase "coach class" in its keyword list or in its current thesaurus, then on this first occasion, it offers the user a traditional path down the conventional tree. But it tracks what the user did, what node of the tree the user went to, and on this basis, it learns a new response to "coach class". The next time a caller (either the same person or a different person) uses the words "coach class" the system does not offer the traditional path as it did the first time, but instead it offers a new set of nodes based on what it learned the first time. Thus, in such implementations, the thesaurus is a dynamically changing entity, continually updating itself by learning new words and terms and learning new "meanings" (i.e. new actions or responses) for existing terms.

Implementations according to the invention can also allow novel groupings of nodes to be presented to the caller based on his query. If he asks for "economy class" without specifying whether he wants an international or domestic reservation, then the system would offer him the nodes a6 and a9 (appropriately phrased in natural language), and allow him to further choose whether he wants international or domestic reservations, something current systems do not offer. In other words, the system can pick out the relevant responses from different branches of the tree and pool them together and offer them to the caller.

This functioning of the system, by which it is able to skip around laterally or vertically in the tree, is enabled by the associating of natural language (i.e. human language) verbal descriptions with each node, and then using these as an initial basis for the navigation, augmented, in some variants, by a dynamically changing thesaurus that greatly expands its range and comprehension.

Thus, based upon a conceptual understanding of the above examples, further details of the process will now be presented.

The flowcharts of FIGS. 7 through 14 are illustrative of a functional example of the general method of a more complex variant the invention as would be implemented in software according to the flowcharts in this case for a newspaper subscription application. It should be understood that particular details are provided in the description below merely for completeness or because they are necessary or helpful for forming an understanding of the particular implementation. They are not to be considered essential for implementing the invention. Similarly, details unrelated to or unnecessary for understanding the invention have been omitted to avoid confusion.

An example implementation is described and contains two programs, a preparatory program, illustrated in FIGS. 7-10 and a transaction or query processing program, illustrated in FIGS. 11-14. In addition, a particular software implementation fairly corresponding to the flowcharts of FIGS. 7 - 14 appears in the Appendix A that follows. The program contained therein, is written in the "C" programming language for execution on any personal computer having a processor, memory , input-output, etc. capabilities to run the particular application in its intended environment.

Broadly, the first program process of FIGS. 7-10 constructs an inverted index and an application-specific thesaurus and the second program process of FIGS. 11-14 uses those constructs in a transaction processing system to interact with a user.

In the preparatory program of FIGS. 7A, 7B and 8-10, the shorthand names of files that the program uses and the contents of the corresponding files are as follows. Notably, both the process parts shown in FIG. 7A and 7B as well as the process part shown in FIG. 8 are indicated as start points. This is because they are each independent of each other in that any of the three could start before any other or two or more could be run concurrently. Thus, it should not be presumed that they are mutually exclusive or any one is per se required for the invention. Moreover, it should be understood that any one or more could have been undertaken at a different time, by a different entity, or for a different application. Whether one or more of the portions shown in FIG. 7A, FIG. 7B or FIG. 8 are the starting points, the starting point for actual operational processing will be the same.

The file named 'p' contains a list of prompts or verbal descriptions in a hierarchical relationship (i.e. they can be visualized or arranged in a tree-type graph).

The file named 'w' contains documents that are related to the prompts or verbal descriptions in 'p'. For example, 'w' could contain a training manual for customer service personnel or a website document that is likely to contain material that is related to the queries customers may have. This file is used to create a thesaurus.

The file named 'f' contains forms that are used to elicit relevant information from customers. They have fields like 'name', for example, which would be used by the system to ask and record a caller's name.

The file named 'x' contains an index associating the forms in 'f' with terminal prompts or descriptions in 'p'. Once a terminal prompt is reached in the process, the corresponding form from the file 'x' is activated, and the system proceeds to elicit information from the user.

The file named 's' contains a list of application-specific stop words, many of which are high-occurrence and/or generally uninformative words like 'a', 'an', 'the' or 'from' or words with a high-occurrence in for the particular application such that they have little meaning, for example, 'fly' in an airline reservation system, 'street' in a navigation system, 'file' in a computer search tool. These are eliminated from 'p' and 'w' and 'f' before processing, because they don't carry any useful information for the application.

The file 't.cfg' contains the thesaurus and inverted index that will be constructed by the program. Of course, in alternative variants, the thesaurus could be a separate file from the inverted index file and either or both could be made up of multiple files.

The file 'l.cfg' is a file that is used to store newly learned words. As with the t.cfg' file, the 'l.cfg' file need not be a separate file, it could be part of 't.cfg', or part of a separate thesaurus and/or inverted index file. Similarly, the 'l.cfg' file could be made up of several files.

With reference to FIGS. 7A, 7B and 8 through 10, the processes as carried out by the first program are as follows. It bears noting that, although the process and its components are presented by way of example in a particular order, unless a specific process component is expressly stated to necessarily have to occur at a particular time or after some other particular process component, or two process components must necessarily occur in sequence because one relies upon completion of the other before it can start, no order should be implied or considered required since the order in different implementations may be different and may vary based upon the particular programmer, programming language and/or computer involved.

The files p, w, f, x, and s are each read and processed as follows. It should be understood that the order of processing of file 'p' relative to file 'f' or their respective sub-processing components, although shown sequentially, could be done in a myriad of ways including doing each of the reading extracting and storing concurrently or as a common operation (i.e. reading for both is done before extracting for both, etc.).

Specifically, keywords are extracted from p ___ and from f ___. These are initially just all the words or terms contained in the prompts in p. The keywords are stored, for example, in a temporary file.

Similarly, thesaurus words are extracted from w. These are initially just all the words or terms in w. They are also stored, for example, in a temporary file.

Stop words are loaded from s (902) and stop words and duplicate words are eliminated from keywords and thesaurus words stored in the temporary files.

The thesaurus is constructed in accordance with FIGS. 9 and 10 described in overview as follows:

a. Increment the file of thesaurus words with keywords from p and f remaining after elimination of stop words.

b. Create a matrix of thesaurus words as row words (or words listed along the rows of the matrix) against keywords as column words (or words listed along the columns of the matrix).

c. Count the number of co-occurrences of each row word with each column word of the matrix in the documents contained in w and fill in that number in the corresponding matrix cell. (For example, a co-occurrence of a pair of words may be defined as that pair occurring in the same paragraph. If w is made up of a hundred paragraphs, then take each pair of row word and column word and count the number of times this pair occurs within the space of each of the hundred paragraphs in w. For each pair, the pair may co-occur zero or more times in a paragraph and add up the number of co-occurrences in all the paragraphs in w.)

This process yields a matrix filled with nonnegative integers in each cell. It is then possible to consider each row of numbers as a vector associated with the corresponding row word. When viewed geometrically, these vectors, one for each row word, form angles with each other in a multi-dimensional space. As a result, we can calculate the cosine of each such angle by computing scalar products for the angles. Thus, we compute the cosines of the angles formed by the vectors associated with each pair of row words.

The cosine values for all pairs of row words and column words are calculated and stored, for example, in a new matrix.

For each row word, the top 'n' cosine values are identified as are the corresponding keywords. For example, in an airline system context, if there are two row words 'coach' and 'economy', where 'economy' is also a keyword (originally from p and/or f), and if the cosine value of this pair or words is among the top 'n' cosines for the word 'coach', then 'economy' is identified as a synonymous keyword for coach.

A new file can then be created, formatted for example, by listing thesaurus words on the left (e.g. coach), and against each thesaurus word, its associated keywords (e.g. economy). This is referred to as an inverted index (i.e. the thesaurus) of row words and their keyword synonyms. Essentially, this file will now contain words like 'coach' coupled with its particular alternative meanings, one of which may be 'economy'. The user interactive transaction processing program, the second program, will later use this thesaurus file when a caller uses a word like 'coach' in his query to determine the relevant keywords (like 'economy'). This will enable the program to find the relevant prompt with which to respond to the user.

Optionally, to provide the system with a set of prompts or verbal descriptions with which to respond to a user, another inverted index is created using the files p, f, and x. This index will contain a list of keywords from p and/or f associated with the prompts in which they occur. Thus, when a user uses a synonym like 'coach' in a query, the second program will look up the thesaurus, find the keywords corresponding to it (e.g. 'economy'), and then look up the inverted index to find the prompts corresponding to 'economy' and other corresponding keywords.

Once both the inverted index and thesaurus files have been created, the file t.cfg can be created from them for use by the second program.

One example of the program flow for a fairly generic transaction processing program implementing one variant of the invention is illustrated in the flowcharts of FIGS. 11 through 14. This example is configured to incorporate a collection of several of previously described simple aspects. To demonstrate the functions of this program and how this program operates, for context we use an example interaction that a calling customer might have with this example system.

Following the example is the Appendix contains that program code essentially implementing a variant of the invention largely corresponding to that of FIGS. 7 through 14.

The particular example we use for purposes of illustration is for an automated telephone system for a newspaper, like the New York Times. For simplicity, every item in the flowchart is not traced through since, an understanding of the process with respect to one path will be sufficient to allow an understanding of the other paths.

The example begins with "I want to subscribe" uttered by the caller to the system. We will assume that the first three words of the query (i.e. "I", "want", and "to") are stop words and the last word (i.e. "subscribe") is neither a keyword nor a thesaurus word.

The process as carried out by the second program are as follows:

The files t.cfg, l.cfg, f, x, and s are read (1102).

The keywords, thesaurus words, prompts from t.cfg. are loaded (1104), as are the learned words from l.cfg. Initially, l.cfg will be empty as the program has not yet learned any new words. The forms and index of forms against prompts from f and x respectively are loaded, as are stop words from s.

The program opens the interaction with a greeting (1106) and an elicitation of the first query from the caller (1108). This might be: "Thank you for calling the New Herald. How may we help you?"

The caller then utters his or her statement: "I want to subscribe".

The stop words in the statement are first eliminated, leaving behind just the word "subscribe".

The statement is then processed in the following way:

The keywords and the thesaurus words remaining in the query are identified (1202, 1204) by comparing with the list in t.cfg and l.cfg. As we have assumed that "subscribe" is neither, we have none.

The prompts that best match the identified keywords and thesaurus words are selected (1206). As there are no such words identified, there are no prompts selected.

The program arrives at a decision in the flowchart: are any nodes selected? (1208). Since the answer is in the negative, the program will follow the branch and select the top level node (1218). (Note: These top level prompts are the ones at the top level of the menu tree.) This completes the prompt selection process. The process then proceeds to the second part of the query process.

The process proceeds with another decision: has a single leaf prompt been selected? (1210). Since the top level prompts are selected (of which there are more than one and also none is a leaf prompt), a negative answer is the result.

These prompts or verbal descriptions are issued to the user (caller) and elicit another response. Assume that the offered verbal descriptions are as follows:

System: Are you calling about subscriptions?

System: Is there a problem with your paper or delivery?

System: Would you like information about the New York Times website?

System: Are you calling about advertisements?

System: Are you calling about something else?

Assume further that the caller responds as follows:

User: I am calling about subscriptions.

As a result, the program returns to selecting verbal descriptions by identifying the keywords and the thesaurus words remaining in the query by comparing with the list in t.cfg and l.cfg (1202, 1204). "Subscriptions" is now synonymous with a keyword and it is identified.

The program will again select verbal description(s) that best match the identified keywords and thesaurus words (1206).

For example, assume these are:

System: Would you like to order a subscription?

System: Would you like to give a gift subscription?

System: Would you like to change your address or change any other information?

The program then arrives at a decision branch (1208) in the flowchart: are any nodes selected? Since the answer is affirmative, it follows that branch and exits the prompt selection process and returns to the query process.

This begins with another decision box: is a single leaf node selected? (1210). The answer is no, since three prompts have been selected.

Next, these verbal descriptions are issued to the caller and the system will await his response (1220). We assume the caller responds as follows:

User: I want to order a subscription

The program will again return through a loop to the prompt selection process (1202, 1204, 1206) where the program will identify the keywords and the thesaurus words remaining in the query by comparing with the list in t.cfg and l.cfg. "Order" and "subscription" are now identified.

Verbal descriptions are selected that best match the identified keywords and thesaurus words. Now assume this is just the prompt "Would you like to order a subscription?" from the three descriptions above.

The program will then arrive at a decision branch (1208) in the flowchart: are any nodes selected? Since the answer is affirmative, it follows that branch and exits the prompt selection process and returns to the query process to again arrive at a decision: has a single leaf node been selected? (1210). This time the answer is yes, a single prompt has been reached, which is also a leaf prompt, since it is at the bottom of the menu tree.

This is followed by another decision: any verbal description corresponding to the node? (1212). The program checks t.cfg and finds the answer is no.

The branch then leads to yet another decision (1214): is a form for verbal description available? The answer by checking the index x is the yes branch. This leads to the portion of the flowchart of FIG. 13.

The form is processed in the following way:

The first part is a decision: is it a response form? (1302). The answer is no.

The system then issues questions to the caller based on the form and accepts information back (1304). The questions are of the form "Please tell us your name", "Where do you live?", "Do you want an annual or half-yearly subscription?" etc. The caller provides the information to the system.

It repeats the information the caller has given the system and asks if the information is correct. Let us assume the user confirms that the information is correct.

The system then calls an external routine to store the information in a database. The routine returns another form to the system (1306) and returns in a loop to the question: is it a response form? (1302). Since the form contains questions about the payment, based on the type and period of subscription selected by the caller, the answer will be negative.

The system then issues these questions to the caller and the caller provides the required information (1304).

The system then repeats the information and gets a confirmation from the caller.

The information is passed to another routine (mentioned in the form) to update the database. This routine (1306) then returns a response form and again returns in the loop to the question: is it a response form? (1302). This time the answer is yes. The system then issues a response (1308) thanking the caller for the subscription, and exits this process returning to FIG. 11.

The system now exits the query process as well since the caller's query has been completely processed and the corresponding actions taken by the system. It now returns to the main part of the program.

The next process in the main part of the program is a question: is there an unknown word in the caller's query? (1112). The answer is yes, since the word "subscribe" in the initial query was not known to the system. This invokes the portion of the flowchart of FIG. 14.

The system therefore has to learn this previously unknown word:

The learning process begins with a decision: is the word already in l.cfg? (1402). The answer is no, since l.cfg is initially empty and the word has not been encountered before.

The corresponding "NO" branch is followed and the word is added to the list of learned words (initially empty) with keywords from the final single leaf prompt that was selected (1404).

The system then records these changes in l.cfg (1408) and returns to the main part of the program in FIG. 11. It has now learned the meaning of the initially unknown word "subscribe".

Next, the program asks the caller if he wishes to continue (1114) (i.e. are there any further queries). We assume the answer is no and the system thanks the user and exits.

Now, having described the example traversal of one path through the second program with reference to the flowchart, an example dialogue for the path traversal that has taken place is presented so the complete transaction can now be understood.

Dialogue:

Caller: I want to subscribe

System: Are you calling about subscriptions?

System: Is there a problem with your paper or delivery?

System: Would you like information about the New York Times website?

System: Are you calling about advertisements?

System: Are you calling about something else?

Caller: I am calling about subscriptions

System: Would you like to order a subscription?

System: Would you like to give a gift subscription?

System: Would you like to change your address or change any other information?

Caller: I want to order a subscription

System: Please tell us your name

Caller: Bertrand Russell

System: Where do you live?

The dialogue continues in this way with the system eliciting the required information from the caller.

Having demonstrated traversal in a system where the system was constructed to learn when an unknown word is used, what happens the second time a caller uses the same word "subscribe" in a query after it has been learned by the system can now be demonstrated. This demonstrates the power of including the optional feature of learning in the program.

In this case, the dialogue that occurs when a new caller uses the word "subscribe" following the above is now presented.

Dialogue:

Caller: I want to subscribe

System: Please tell us your name

Caller: J. L. Austin

System: Where do you live?

Thereafter, the process continues. Notably, the system has now learned the correct response to the query "I want to subscribe".

Other Variants

Having described several simple and more complex examples that make it possible to use the invention, other variants can now be presented. Examples of such optional functions that can be incorporated into other variants, individually or collectively, include:

a) creating the thesaurus by providing access to a collection of multiple documents and determining synonymy based on sufficient similarity of meaning with the keywords as measured by the frequency of co-occurrence of the keywords in the collection of documents;

b) identifying words in the user's response by recording the response for future learning;

c) parsing out of a response all non-stop word unknown terms or keywords;

d) identifying synonyms for all non-stop terms in the user's response;

e) cycling between user and system responses until the user reaches a vertex (i.e. verbal description) that enables him to carry out his goal and updating the thesaurus when the goal vertex is reached by associating the recorded previously unknown words in the user's response with the keywords that are associated with the verbal description reached by the user;

f) recording, when the goal vertex is reached, the pairs of synonyms in the user's responses and the keywords that are associated with the verbal description reached by the user;

g) removing associations between keywords and their synonyms from the thesaurus that have not been accessed more than a specified amount of times within a specified period (this can be based upon a parameter set in the system by the system's administrator or can occur as part of program maintenance or updates);

h) selecting the verbal descriptions that best meet the user's goal as indicated by the keywords and synonyms in the user's response by identifying the keywords in the user's response and/or the keywords corresponding to synonyms in the user's response and computing a degree of match between each verbal description and the identified keywords, in accordance with conventional linguistic processing techniques;

i) computing the degree of match between verbal descriptions and identified keywords by utilizing the pairs of synonyms in user's response and the keywords associated with the verbal descriptions reached by users as previously recorded;

j) responding to the user on the basis of verbal descriptions selected by presenting the user with verbal descriptions that best match the user's previous response;

k) for "best match" variants, in the event that even the best matches have a low degree of match, the best "n" verbal descriptions are presented to the user ("n" being a number representing a predetermined system parameter);

l) for "best match" variants, in the event that the best matches have a low degree of match, the user is automatically connected to a human operator, when or if a human operator is available;

m) for "best match" variants in the event that the best matches have a low degree of match, the best "n" verbal descriptions are presented to the user, along with an option of being connected to a human operator when or if a human operator is available;

n) presenting the user with those verbal descriptions that best match the user's previous response in order to elicit any information from the user that may be required to accomplish the user's goal;

o) recording information elicited from a user in a database for future use;

p) selecting multiple verticies in the graph structure that are not connected to a previously selected vertex, based upon parameters associated with nodes correlated to keywords and synonyms in a user's response;

q) selecting a vertex in the graph structure in the same row as the previously selected vertex based upon the keywords and synonyms in the user's response; and/or

r) updating the thesaurus by adding words from a user's response that are not in the thesaurus.

Finally, it is to be understood that various variants of the invention including representative embodiments have been presented to assist in understanding the invention. It should be understood that they are not to be considered limitations on the invention as defined by the claims, or limitations on equivalents to the claims. For instance, some of these variants are mutually contradictory, in that they cannot be simultaneously present in a single embodiment. Similarly, some advantages are applicable to one aspect of the invention, and inapplicable to others. Thus, no particular features or advantages should be considered dispositive in determining equivalence.

It should therefore be understood that the above description is only representative of illustrative embodiments. For the convenience of the reader, the above description has focused on a representative sample of all possible embodiments, a sample that teaches the principles of the invention. The description has not attempted to exhaustively enumerate all possible combinations or variations, for example, those arising out of the use of particular hardware or software, or the vast number of different types of applications in which the invention can be used. That alternate embodiments may not have been presented for a specific portion of the invention, or that further undescribed alternate embodiments may be available for a portion of the invention, is not to be considered a disclaimer of those alternate embodiments. One of ordinary skill will appreciate that many of those undescribed embodiments incorporate the minimum essential aspects of the invention and others incorporate one or more equivalent aspects.

APPENDIX A

FILE IDENTIFICATION

Main Source Files
main.c, process.c, arraylib.c, stemlib.c, dialog.c, interactive.c, formlib.c
Header Files
globalvar.h, process.h, arraylib.h, forms.h
Make Files
Makefile
Parameter Files
t.ini, d.ini
Data Files
p, w, s, f, x, a
Configuration Files
t.cfg, l.cfg
Shell Script Files
acct_info, add_acct, chg_acct, get_pymt, updt_pymt, susp_deli, updt_acct, prefer

MAIN SOURCE CODE (in C)
main.c: Main Program to process p and w to create the thesaurus

```
/*********************************************************************

  SOURCE CODE DOCUMENTATION

*********************************************************************/
#include <stdio.h>
#include <string.h>
#include "process.h"
#include "arraylib.h"
#include "forms.h"

int numColumn = 0,  numRow = 0 , numIndex = 0, numMenu;
int topValues = 5 ;
char **rowTerms, **columnTerms, **prompts, **stopWords;
double **matrix, **cosine;
float phoneThreshold = 0.02, webThreshold = 0.0006;
int **indexList, **menuList, **thesaurus, **promptKeys;
int numStopWord = 0;
int numForms, numPF;
struct form **formlist;
char ***Fprompts, *wdoc, *pdoc, *sdoc, *fdoc, *xdoc, *cfg;

main(int argc, char *argv[]) {

int i, j;

/*********************************************************************

  PRINT THESAURUS PROGRAM INFO

*********************************************************************/

 /* if (argc != 5) {
   printf("Usage Instructions: t p w f x\n");
   printf("Parses w for matrix row terms and p for matrix column terms.\n");
   printf("*** Exiting, goodbye.\n");
   exit(1);
 } */

 if (argc != 2) {
   printf("Usage Instructions: t <ini-file>\n");
```

```
  printf("*** Exiting, goodbye.\n");
  exit(1);
 }
/*****************************************************************

  OPEN INPUT FILES

*****************************************************************/
 readini(argv[1]);
 loadStopWords(sdoc);
 numPF = loadFormsList(xdoc);
 numForms = loadForms(fdoc);

/*****************************************************************

  PREPARATION FOR PHONEDOC PARSING

*****************************************************************/
 printf("\nReading files ....\n");
 numColumn = processFile(pdoc, &columnTerms, phoneThreshold);
 // printf("The document contains %d relevant terms.\n\n", numColumn);

 // This routine will add the keywords from the Forms into ColumnTerms.
 numColumn = addFormKeys(&columnTerms, numColumn);

  // printf("The document contains %d relevant terms.\n\n", numColumn);

/*****************************************************************

  PREPARATION FOR WEBDOC PARSING

*****************************************************************/

 numRow = processFile(wdoc, &rowTerms, webThreshold);

/*****************************************************************

  MERGE COLUMNTERMS & FINALTERMS INTO ROWTERMS

*****************************************************************/

 numRow = mergeArray(&rowTerms, columnTerms, numRow, numColumn);
 sortArray(rowTerms, numRow);
```

```
  // printf("The document contains %d relevant terms.\n\n", numRow);
/*****************************************************************

   MATRIX CONSTRUCTIONS

*****************************************************************/

  printf("loading prompts ...\n", numIndex);
  numIndex = loadPrompts(pdoc);
  printf("processing words ...\n", numIndex);
  createMatrix(wdoc);
  numRow = eraseZeroes();
  calcCosine();
  fillIndex();
  // This function will add leaf prompts to the index keywords from Forms.
  // appendIndex(argv[3]);
  createThesaurus();
  // printf("created thesaurus .\n\n", numIndex);
  printf("saving data ...\n");
  saveData(cfg);
  printf("done.\n");
}


readini(char * filenm)
{
        FILE * fp;
        char buf[80], key[80], value[80], comment[80];
        int cnt;
        if ((fp=fopen(filenm,"r"))==NULL)
        {
                perror(filenm);
                exit(1);
        }
        while (fgets(buf,79,fp)!=NULL)
        {
        sscanf(buf,"%s %s %s",key,value, comment);
        if (!strcmp(key, "pdoc"))
                        pdoc=strdup(value);
        if (!strcmp(key, "wdoc"))
                        wdoc=strdup(value);
        if (!strcmp(key, "sdoc"))
                        sdoc=strdup(value);
```

```
        if (!strcmp(key, "fdoc"))
                fdoc=strdup(value);
        if (!strcmp(key, "xdoc"))
                xdoc=strdup(value);
        if (!strcmp(key, "cfg"))
                cfg=strdup(value);
        if (!strcmp(key, "pt"))
                sscanf(buf,"%s %f %s",key,&phoneThreshold,value);
                //phoneThreshold=(float)atof(value);
        if (!strcmp(key, "wt"))
                sscanf(buf,"%s %f %s",key,&webThreshold,value);
        if (!strcmp(key, "tv"))
                topValues=atoi(value);

        }
}
```

process.c: This program contains various functions called from Main

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "globalvar.h"
#include "arraylib.h"
#include "forms.h"
#define min(x, y)  (x < y) ? x : y

int *rowcount, *colcount;

int processFile(char *filename, char ***cArray, float threshold) {
FILE * fp;
char tmpWord[50], paraFlag;
int i, numWords = 0, wordLen = 0, totWords = 0;
float *freqArray;

fp = fileOpen(filename,"r");
*cArray = NULL;
freqArray = NULL;

while((wordLen = fetchWord(fp, tmpWord)) != 0) {
        totWords++;
        if (! inArray(stopWords, tmpWord, numStopWord)) { // ignore stopwords
```

```
                stemWord(tmpWord);
                if (i = inArray(*cArray, tmpWord, numWords)) {
                        freqArray[i - 1]++;
                        }
                else {
                        addWord(cArray, tmpWord, ++numWords);
                        freqArray = (float *) realloc(freqArray, numWords * sizeof(float));
                        freqArray[numWords - 1] = 1;
                        }
                }
        }
 // printf("Totwords = %d, numWords = %d\n",totWords, numWords);
for ( i = 0; i < numWords; i++)
        {
       if ((float)(freqArray[i] / totWords) >= threshold)
          (*cArray)[i] = NULL;
        }

numWords = removeNulls((*cArray), numWords);
sortArray((*cArray), numWords);

return numWords;
}

/*****************************************************************************
Prompts Processing
*****************************************************************************/

int loadPrompts(char *filename)
{
 char buffer[256];
 int i = 0, j, len, nc;
 int level[10], tabs, m = 0;
 FILE * fp;

 for (i = 0; i < 10; i++)
        level[i] = 0;

 fp = fileOpen(filename,"r");
 prompts = NULL;
 menuList = NULL;
 j = i = 0;
 while (fgets(buffer, 256, fp) != NULL)
```

```
        {
        tabs = allTrim(buffer);
        if ((len = strlen(buffer)) == 0)
                continue;
        if ((j = inArray(prompts, buffer, i)) == 0)
                        j = i;
        else
                j--;
        level[tabs + 1] = j + 1;
        menuList = (int **)realloc(menuList, ++m * sizeof(int *));
        menuList[m - 1] = (int *)malloc(3 * sizeof(int));
        menuList[m - 1][0] = level[tabs];
        menuList[m - 1][1] = level[tabs + 1];
        menuList[m - 1][2] = 0;
        if (j == i)
                addWord(&prompts, buffer, ++i);
        }
        numMenu = m;
for (j = 0; j < numMenu; j++)
        {
        for(m = 0; m < numMenu; m++)
                if (menuList[j][1] == menuList[m][0])
                        break;
        if (m != numMenu)  /* Leaf Node */
                continue;
        nc = 0;
        for (m = 0; m < numMenu; m++)
                if (menuList[m][0] == menuList[j][0])
                        nc++;
        if (nc != 1)
                {
                menuList[j][2] = 99;
                continue;
                }
        len = strlen(prompts[menuList[j][1] - 1]);
        if (prompts[menuList[j][1] - 1][len - 1] == '?')
                menuList[j][2] = 99;
        else
                {
                for (m = 0; m < numMenu; m++)
                        if (menuList[m][1] == menuList[j][0])
                                menuList[m][2] = menuList[j][1];
                menuList[j][2] = 100;
```

```
                }
        }
return i;
}


void fillIndex()
{
int i, j, k;

indexList = (int **)malloc(numColumn * sizeof(int *));
for (i = 0; i < numColumn; i++) {
        indexList[i] = (int *)malloc(numIndex * sizeof(int));
        for (j = 0; j < numIndex; j++)
                indexList[i][j] = 0;
        }

for (i = 0; i < numIndex; i++)
                updateThesaurus(prompts[i], i + 1);
updateFrmForms();

}

updateThesaurus( char *str, int pmpt)
{
char tmpstr[256];
char *sarray[50];
int i, j, k, wrds;
int iflag = 0, dflag = 0;

strcpy(tmpstr, str);
wrds = readValues(tmpstr, sarray);
stemArray(sarray, wrds);
for (i = 0; i < wrds; i++)
        {
        for (j = 0; j < numColumn; j++)
                {
                if (!strcmp(columnTerms[j], sarray[i]))
                        {
                        iflag = 1;
                        dflag = 0;
                        for (k = 0; indexList[j][k] && k < numIndex; k++)
                                if (indexList[j][k] == pmpt)
```

```
                                        dflag = 1;
                    if (k < numIndex && dflag == 0)
                            indexList[j][k] = pmpt;
                    break;
                    }
            }
        }
if (iflag == 0)
        {
        printf("** warning the following prompt does not contain index word\n");
        printf("\t%s\n", str);
        }
}


/*****************************************************************
 Create Matrix here
*****************************************************************/
void createMatrix(char * filename) {
 int i, j, nwp = 0, k = 0;
 int minv;
 FILE * fp;

 /* allocate memory for matrix */
 matrix = (double **)malloc(numRow * sizeof(double *));
 for (i = 0; i < numRow; ++i) {
        matrix[i] = (double *)malloc(numColumn * sizeof(double));
        for (j = 0; j < numColumn; j++)
                matrix[i][j] = 0;
        }

 /* allocate memory for rowcount and column count */
 rowcount = (int *)malloc(numRow * sizeof(int));
 colcount = (int *)malloc(numColumn * sizeof(int));

 /* Go to start of document */

 fp = fileOpen(filename,"r");

 while (!feof(fp))
        {
        /* initialize rowcount array */
        for (j = 0; j < numRow; j++)
```

```
            rowcount[j] = 0;
        /* initialize columncount array */
        for (j = 0; j < numColumn; j++)
            colcount[j] = 0;

        nwp = readPara(fp);
        if (feof(fp))
            break;
        if (nwp == 0)
            continue;

        /* add co-occurance of rowword & colword to the matrix */
        for (j = 0; j < numRow; j++)
            for (k = 0; k < numColumn; k++) {
                minv = min(rowcount[j], colcount[k]);
                matrix[j][k] += minv;
                }
        }
}


int readPara(FILE *fp )
{
int i, j, k, wcount = 0, m = 0;
int nextpara, currpara, wordLen;
char tmpword[50];

currpara = ftell(fp);
wcount = wordsInPara(fp);
if (feof(fp)) {
        if (wcount == 0)
            return 0;
        }

nextpara =  ftell(fp);
fseek(fp, currpara, 0);
for (i = 0; i < wcount; ++i) {
        wordLen = fetchWord(fp, tmpword);
        if (inArray(stopWords,tmpword,numStopWord))
            continue;
        stemWord(tmpword);

            /* count the occurance of each word from the row in para */
```

```
                for (j = 0; j < numRow; j++)
                        if (!strcmp(rowTerms[j], tmpword)) {
                                rowcount[j]++;
                                break;
                                }
                /* count the occurance of each word from the column in para */
                for (j = 0; j < numColumn; j++)
                        if (!strcmp(columnTerms[j], tmpword)) {
                                colcount[j]++;
                                break;
                                }

 }
 fseek(fp, nextpara, 0);
 return wcount;
}

int wordsInPara (FILE *fp)
{
 int c, count = 0;
 int state;
 const int out = 0, in = 1;

 state = out;
 while ((c = getc(fp)) != EOF) {
        if (!isalpha(c)) {
                if (c == '\n' || c == EOF)
                        break;
                state = out;
        }
        else
        if (state == out) {
                state = in;
                count++;
        }
 }
 return count;
}

/*************************************************************
Calculate Cosine Function
*************************************************************/
void calcCosine()
```

```
{
int i, j, k, sum;

/* memory allocation for the cosine matrix */

cosine = (double **)malloc(numRow * sizeof(double *));
for (i = 0; i < numRow; ++i) {
        cosine[i] = (double *)malloc(((numRow) * sizeof(double)));
        for (j = 0; j < numRow; j++)
                cosine[i][j] = 0;
        }

/*Normalization*/

for (i = 0; i < numRow; ++i)
        {
        sum = 0;
        for (k = 0; k < numColumn; ++k)
                sum += matrix[i][k] * matrix[i][k];
        if (sum != 0)
                {
                for (j = 0; j < numColumn; ++j)
                        matrix[i][j] = matrix[i][j] / sqrt(sum);
                }
        }

 /*Cosines*/

for (i = 0; i < numRow; ++i)
        {
        for (k = i + 1; k < numRow; ++k)
                {
                cosine[i][k] = 0;
                for (j = 0; j < numColumn; ++j)
                        cosine[i][k] += matrix[i][j] * matrix[k][j];
                }
        }
}

/*********************************************************************
eraseZeroes : removes the row with all zero column in the matrix
*********************************************************************/
int eraseZeroes() {
```

```
int j, k;
int cond;

/* Free and nullify the rowTerms and matrix row for all zeroes */
for (j = 0; j < numRow; ++j) {
 cond = 1;
 for (k = 0; k < numColumn; ++k) {
  if (matrix[j][k] != 0) {
      cond = 0;
      break;
  }
 }
 if (cond == 1) {
      rowTerms[j] = NULL;
      matrix[j] = NULL;
   }
}

/* Push NULL rows at the end of arrays */
for (j = 0; j < numRow; j++)
 {
 if (rowTerms[j] == NULL)
      {
      for (k = j + 1; k < numRow; k++)
            if (rowTerms[k] != NULL)
                  break;
      if (k < numRow)
       {
       rowTerms[j] = rowTerms[k];
       matrix[j] = matrix[k];
       rowTerms[k] = NULL;
       matrix[k] = NULL;
       }
   }
 }

/* count new numRow */
for (j = 0; (rowTerms[j] != NULL) && j < numRow; j++);

return j;

}
```

```
/*************************************************************************
createThesaurus: Function to Create Thesaurus of rowTerms by taking the
index words matching the top 5 cosine values.
*************************************************************************/
void createThesaurus()
{
int i, j, k, l;
int m, numword;
double *tmpcos, prevcosine = 0;
int *colnum;

tmpcos = (double *)malloc(numRow * sizeof(double));
colnum = (int *)malloc(numRow * sizeof(int));

thesaurus = (int **)malloc(numRow * sizeof(int *));
for (i = 0; i < numRow; i++) {
        thesaurus[i] = (int *)malloc(numColumn * sizeof(int));
        for (j = 0; j < numColumn; j++)
                thesaurus[i][j] = 0;
        }

/* initialization of thesaurus */

for (i = 0; i < numRow; i++) {
        for ( j = 0; j < numRow; j++) {
                if (i > j)
                        tmpcos[j] = cosine[j][i];
                else
                        if (i < j)
                                tmpcos[j] = cosine[i][j];
                        else
                                tmpcos[j] = 0;
                colnum[j] = j;
                }
        floatSort(colnum, tmpcos, numRow);

        numword = prevcosine = 0;

        /* count top 'topValues' of cosine */
        for (m = 0; m < numColumn; m++) {
                if (prevcosine != tmpcos[m])
                        numword++;
```

```
                prevcosine = tmpcos[m];
                if (numword == topValues + 1)
                        break;
                }
        --m;
        /* m = total num of syn */

        for (j = k = 0; k <= m; k++) {
                if ((l = inArray(columnTerms,rowTerms[colnum[k]], numColumn)) != 0)
                        if (tmpcos[k] != 0) {
                                thesaurus[i][j] = l;
                                j++;
                                }
                }

        }
}

/*************************************************************************
floatSort : Sorts the array of cosine values and corresponding index of
index words in reverse order.
*************************************************************************/
floatSort(int *colnum, double *tmpcos, int numRow)
{
int i, j, k;
double f;
for (i = numRow - 1; i > 0; i--)
        for (j = 0; j < i; j++) {
                if (tmpcos[j] < tmpcos[j + 1]) {
                        f = tmpcos[j], k = colnum[j];
                        tmpcos[j] = tmpcos[j + 1], colnum[j] = colnum[j + 1];
                        tmpcos[j + 1] = f, colnum[j + 1] = k;
                        }
                }
}

void saveData(char *filenm)
{
int i, j, k, l;
FILE *fp;

fp = fileOpen(filenm, "w");
```

```
printArray(fp, "PROMPTS", prompts, NULL, numIndex, 0); // Write Prompts to the file

/********** Write Menu-Tree to the file **********/
// printArray(fp, "MENUTREE", NULL, menuList, numMenu, 2);
fprintf(fp, "[%s]\n", "MENUTREE");
for (i = 0; i < numMenu; i++)
        fprintf(fp, "%d,%d,%d\n", menuList[i][0], menuList[i][1],menuList[i][2]);
fprintf(fp, "\n");

printArray(fp, "INDEX", columnTerms, indexList, numColumn, numIndex); // Write Index to
the file
printArray(fp, "THESAURUS", rowTerms, thesaurus, numRow, numColumn); // Write
Thesaurus to the file

fclose(fp);
printf("Data saved in %s\n",filenm);
}

printArray(FILE *fp, char *head, char **cArray, int **iArray, int cNum, int iNum)
{
int i,j;
fprintf(fp, "[%s]\n", head);
for (i = 0; i < cNum; i++)
        {
        fprintf(fp, "%s ", cArray[i]);
        for (j = 0; j < iNum && iArray[i][j] != 0; j++)
                fprintf(fp, "%d,", iArray[i][j]);
        fprintf(fp, "\n");
        }
fprintf(fp, "\n");
}



int  addFormKeys(char ***cArray, int count)
{
char **wordList, *tmparray[20];
int i, j, k, words;
int l, tmpcount;

wordList = NULL;
words = 0;
for(i = 0; i < numForms; i++)
```

```
        for (j = 0; j < formlist[i]->numFields; j++)
                {
                if (!strcmp("MChoice", formlist[i]->fields[j]->Type))
                        for(k = 0; k < formlist[i]->fields[j]->numChoice; k++)
                                {
                                tmpcount = createArray(formlist[i]->fields[j]->Choice[k],
tmparray);
                                for(l = 0; l < tmpcount; l++)
                                        addWord(&wordList, tmparray[l], ++words);
                                }
                }

i = mergeArray(cArray, wordList, count, words);
sortArray((*cArray), i);
return i;

}



updateFrmForms()
{
int i, j, k, l;
int m, n, x, tmpcount;
int pmpt;
char *tmpstr, *tmparray[20];

for (i = 0; i < numPF; i++)
        {
        pmpt = inArray(prompts, Fprompts[i][1], numIndex);
        if (pmpt == 0)
                {
                printf("Unknown prompt encountered for form %s\n",Fprompts[i][0]);
                exit(1);
                }
        for(j = 0; j < numForms; j++)
                if (!strcmp(Fprompts[i][0], formlist[j]->name))
                        break;
        if (j == numForms)
                continue;
        for(k = 0; k < formlist[j]->numFields; k++)
                {
                if (strcmp(formlist[j]->fields[k]->Type,"MChoice"))
```

```
                continue;
            for(l = 0; l < formlist[j]->fields[k]->numChoice; l++)
                {
                tmpcount = createArray(formlist[j]->fields[k]->Choice[l], tmparray);
                for(m = 0; m < tmpcount; m++)
                        {
                        n = inArray(columnTerms, tmparray[m], numColumn);
                        n--;
                        for (x = 0; indexList[n][x] && x < numIndex; x++)
                                if (indexList[n][x] == pmpt)
                                        break;
                        if (x < numIndex && indexList[n][x] == 0)
                                indexList[n][x] = pmpt;
                        }
                }
            }
        }
}
```

arraylib.c: This program contains general purpose functions

```
#include <stdio.h>
#include <string.h>
#include "globalvar.h"
#include "forms.h"

FILE * fileOpen(char *, char *);

int fetchWord(FILE *f, char * wrd) {
int i = 0, c;
wrd[0] = 0;
if (feof(f))
        return 0;
while(!isalpha(c = fgetc(f)))
        if (c == EOF)
                return 0;
do {
        wrd[i++] = tolower(c);
        } while(isalpha(c = fgetc(f)));
wrd[i] = 0;
return i;
}
```

```
int inArray(char **array, char *word, int length)
{
int i;

for (i = 0; i < length; i++)
        if (array[i] != NULL && !strcmp(array[i], word))
                        return i + 1;
return 0;
}

int removeNulls(char **strarray, int numWords)
{
int i, j;

for (i = 0; i < numWords; i++)
        {
        if (strarray[i] == NULL)
                {
                for (j = i + 1; j < numWords; j++)
                        if (strarray[j] != NULL)
                                {
                                strarray[i] = strarray[j];
                                strarray[j] = NULL;
                                break;
                                }
                }
        }

/* get count of filtered words */
for (j = 0; (strarray[j] != NULL) && (j < numWords); j++);
return j;
}

int mergeArray(char ***Array1, char **Array2, int numArray1, int numArray2) {
int i;

for (i = 0; i < numArray2; i++)
        if (! inArray((*Array1), Array2[i], numArray1))
                addWord(Array1, Array2[i], ++numArray1);

return numArray1;
}
```

```
int readValues(char *str, char **array)
{
int i , j = 0, c;
int state;
const int out = 0, in = 1;

state = out;
for (i = 0; (c = str[i]) != 0; i++)
        {
        if (!isalnum(c))   /* alfa-numeric to read numbers also  */
                {
                state = out;
                str[i] = 0;     /* word is over end it with null */
                }
        else
                {
                str[i] = tolower(c);
                if (state == out)
                        {
                        state = in;
                        array[j++] = str + i; /* word started, store the ptr.*/
                }
                }
        }
return j;
}

void sortArray(char *allwords[], int numwords) {
 int i = 0;
 int j = 0;
 char *tmp;

 for (i = 0; i < numwords; ++i)
   for (j = i + 1; j < numwords; ++j)
     if (strcmp(allwords[i], allwords[j]) > 0) {
         tmp = allwords[i];
         allwords[i] = allwords[j];
         allwords[j] = tmp;
     }
}

loadStopWords( char * filename) {
```

```
FILE * fp;
char tmpWord[50];
int wordLen = 0;

numStopWord = 0;
fp = fileOpen(filename,"r");
stopWords = NULL;
while((wordLen = fetchWord(fp, tmpWord)) != 0)
                addWord(&stopWords, tmpWord, ++numStopWord);
}

FILE * fileOpen(char *filename, char *mode)
{
FILE * fp;

if ((fp = fopen(filename, mode)) == NULL) {
                perror(filename);
                exit(1);
                }
return fp;
}

addWord(char ***cArray, char * word, int c)
{
*cArray = (char **) realloc(*cArray, c * sizeof(char *));
(*cArray)[c - 1] = strdup(word);
}

int removeZeros(int *intArray, int numInt)
{
int i, j;

for (i = 0; i < numInt; i++)
        {
        if (intArray[i] == 0)
                {
                for (j = i + 1; j < numInt; j++)
                        if (intArray[j] != 0)
                                {
                                intArray[i] = intArray[j];
                                intArray[j] = 0;
                                break;
                                }
```

```
                }
        }

/* get count of filtered integers */
for (j = 0; (intArray[j] != 0) && (j < numInt); j++);
return j;
}

/*****************************************************************88
Newly added functions ( for further reducing the code )
*****************************************************************/
int breakStr(char * str, char **strarray)
{
char c, *tmpstr;
int i, j = 0;
int state;
const int out = 0, in = 1;
/* Seperate the sentence into individual words */
tmpstr = strdup(str);
state = out;
for (i = 0; (c = tmpstr[i]) != 0; i++)
        {
        if (!isalpha(c))
                {
                state = out;
                tmpstr[i] = 0;
                }
        else
                {
                tmpstr[i] = tolower(c);
                if (state == out)
                        {
                        state = in;
                        strarray[j++] = tmpstr + i;
                }
                }
        }
return j;
}

/* remove stopWords */
filterStopWords(char ** strarray, int numWords)
{
```

```
int i;

for (i = 0; i < numWords; i++)
        if (inArray(stopWords, strarray[i], numStopWord))
                        strarray[i] = NULL;
}

/* remove duplicates */
filterDuplicates(char ** strarray, int numWords)
{
int i;
for (i = 0; i < numWords; i++)
        if (strarray[i] != NULL && inArray(strarray, strarray[i], i))
                        strarray[i] = NULL;
}

int loadFormsList( char *filename)
{

char buf[256];
FILE *fp;
int len, i;

fp = fileOpen(filename, "r");

Fprompts = NULL;
numPF = 0;
while (fgets(buf,255,fp) != NULL)
        {
        len = strlen(buf);
        for (i = 0; i < len; i++)
                if (buf[i] == ':')
                        {
                        buf[i] = 0;
                        break;
                        }
        if (i == len)
                {
                fprintf(stderr, "Error in Prompt list\n");
                exit(0);
                }
        allTrim(buf);
        allTrim(buf + i + 1);
```

```
        Fprompts = (char ***)realloc(Fprompts, (++numPF)*sizeof(char **));
        Fprompts[numPF-1] = (char **)malloc(2 * sizeof(char *));
        Fprompts[numPF-1][0] = strdup(buf);
        Fprompts[numPF-1][1] = strdup(buf + i + 1);
        }
fclose(fp);
return numPF;
}

int loadForms(char * filename)
{
int i, j, formcount = 0;
FILE *fp;
char buf[80], **namelist = NULL;
formlist = NULL;
numForms = 0;

fp = fileOpen(filename, "r");
while(fgets(buf,79,fp) != NULL)
        {
        if (buf[0] == '[')
                {
                for(i = 0; buf[i]; i++)
                        if (buf[i] == '[' || buf[i] == ']')
                                buf[i] = ' ';
                allTrim(buf);
                addWord(&namelist, buf, ++formcount);
                }
        }

for ( i = 0; i < formcount; i++)
        {
        formlist = (struct form **)realloc(formlist, (++numForms) * sizeof(struct form *));
        formlist[numForms - 1] = (struct form*)malloc(sizeof(struct form));
        loadForm(fp , formlist[numForms - 1], namelist[i]);
        }
fclose(fp);
return numForms;
}

int allTrim (char * str)
{
        int i, j, sf, tabs;
```

```
        for (i = tabs = 0; isspace(str[i]) && str[i]; i++)
                tabs += (str[i] == '\t')? 1: 0;
        for ( j = sf = 0; str[i]; i++, j++)
                str[j] = iscntrl(str[i])? ' ': str[i];
        for(str[j--] = 0; isspace(str[j]) && j > 0; str[j--] = 0);
        return tabs;
}

int createArray (char * str, char ** array)
{
int count;
count = breakStr(str, array);
return processArray(array, count, 1);
}

int processArray(char ** array, int count, int sflag)
{
if (sflag)
        filterStopWords(array, count);
stemArray(array, count);
filterDuplicates(array, count);
return removeNulls(array, count);
}
```

stemlib.c: This program contains functions related to stemming algorithim

```
/* This is the Porter stemming algorithm, coded up in ANSI C by the author.
It may be be regarded as cononical, in that it follows the algorithm presented
in Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14, no. 3,
pp 130-137, only differing from it at the points maked --DEPARTURE-- below.

  See also http://www.tartarus.org/~martin/PorterStemmer

  The algorithm as described in the paper could be exactly replicated
  by adjusting the points of DEPARTURE, but this is barely necessary,
  because (a) the points of DEPARTURE are definitely improvements, and
  (b) no encoding of the Porter stemmer I have seen is anything like
  as exact as this version, even with the points of DEPARTURE!

  You can compile it on Unix with 'gcc -O3 -o stem stem.c' after which
  'stem' takes a list of inputs and sends the stemmed equivalent to
```

```
  stdout.

  The algorithm as encoded here is particularly fast.

  Release 1
*/

#include <string.h>  /* for memmove */

#define TRUE 1
#define FALSE 0

/* The main part of the stemming algorithm starts here. b is a buffer
  holding a word to be stemmed. The letters are in b[k0], b[k0+1] ...
  ending at b[k]. In fact k0 = 0 in this demo program. k is readjusted
  downwards as the stemming progresses. Zero termination is not in fact
  used in the algorithm.

  Note that only lower case sequences are stemmed. Forcing to lower case
  should be done before stem(...) is called.
*/

static char * b;      /* buffer for word to be stemmed */
static int k,k0,j;    /* j is a general offset into the string */

/* cons(i) is TRUE <=> b[i] is a consonant. */

int cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return FALSE;
      case 'y': return (i==k0) ? TRUE : !cons(i-1);
      default: return TRUE;
   }
}

/* m() measures the number of consonant sequences between k0 and j. if c is
  a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
  presence,

     <c><v>       gives 0
     <c>vc<v>     gives 1
     <c>vcvc<v>   gives 2
     <c>vcvcvc<v> gives 3
```

```
    ....
*/

int m()
{  int n = 0;
   int i = k0;
   while(TRUE)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(TRUE)
   {  while(TRUE)
      {  if (i > j) return n;
           if (cons(i)) break;
           i++;
      }
      i++;
      n++;
      while(TRUE)
      {  if (i > j) return n;
        if (! cons(i)) break;
        i++;
      }
      i++;
   }
}

/* vowelinstem() is TRUE <=> k0,...j contains a vowel */

int vowelinstem()
{  int i; for (i = k0; i <= j; i++) if (! cons(i)) return TRUE;
   return FALSE;
}

/* doublec(j) is TRUE <=> j,(j-1) contain a double consonant. */

int doublec(int j)
{  if (j < k0+1) return FALSE;
   if (b[j] != b[j-1]) return FALSE;
   return cons(j);
}
```

```
/* cvc(i) is TRUE <=> i-2,i-1,i has the form consonant - vowel - consonant
   and also if the second c is not w,x or y. this is used when trying to
   restore an e at the end of a short word. e.g.

     cav(e), lov(e), hop(e), crim(e), but
     snow, box, tray.

*/

int cvc(int i)
{  if (i < k0+2 || !cons(i) || cons(i-1) || !cons(i-2)) return FALSE;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return FALSE;
   }
   return TRUE;
}

/* ends(s) is TRUE <=> k0,...k ends with the string s. */

int ends(char * s)
{  int length = s[0];
   if (s[length] != b[k]) return FALSE; /* tiny speed-up */
   if (length > k-k0+1) return FALSE;
   if (memcmp(b+k-length+1,s+1,length) != 0) return FALSE;
   j = k-length;
   return TRUE;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

void setto(char * s)
{  int length = s[0];
   memmove(b+j+1,s+1,length);
   k = j+length;
}

/* r(s) is used further down. */

void r(char * s) { if (m() > 0) setto(s); }

/* step1ab() gets rid of plurals and -ed or -ing. e.g.
```

```
    caresses -> caress
    ponies   -> poni
    ties     -> ti
    caress   -> caress
    cats     -> cat

    feed     -> feed
    agreed   -> agree
    disabled -> disable

    matting  -> mat
    mating   -> mate
    meeting  -> meet
    milling  -> mill
    messing  -> mess

    meetings -> meet

*/

void step1ab()
{  if (b[k] == 's')
   {  if (ends("\04" "sses")) k -= 2; else
      if (ends("\03" "ies")) setto("\01" "i"); else
      if (b[k-1] != 's') k--;
   }
   if (ends("\03" "eed")) { if (m() > 0) k--; } else
   if ((ends("\02" "ed") || ends("\03" "ing")) && vowelinstem())
   {  k = j;
      if (ends("\02" "at")) setto("\03" "ate"); else
      if (ends("\02" "bl")) setto("\03" "ble"); else
      if (ends("\02" "iz")) setto("\03" "ize"); else
      if (doublec(k))
      {  k--;
         {  int ch = b[k];
            if (ch == 'l' || ch == 's' || ch == 'z') k++;
         }
      }
      else if (m() == 1 && cvc(k)) setto("\01" "e");
   }
}

/* step1c() turns terminal y to i when there is another vowel in the stem. */
```

```
void step1c() { if (ends("\01" "y") && vowelinstem()) b[k] = 'i'; }


/* step2() maps double suffices to single ones. so -ization ( = -ize plus
   -ation) maps to -ize etc. note that the string before the suffix must give
   m() > 0. */

void step2() { switch (b[k-1])
{
   case 'a': if (ends("\07" "ational")) { r("\03" "ate"); break; }
             if (ends("\06" "tional")) { r("\04" "tion"); break; }
             break;
   case 'c': if (ends("\04" "enci")) { r("\04" "ence"); break; }
             if (ends("\04" "anci")) { r("\04" "ance"); break; }
             break;
   case 'e': if (ends("\04" "izer")) { r("\03" "ize"); break; }
             break;
   case 'l': if (ends("\03" "bli")) { r("\03" "ble"); break; } /*-DEPARTURE-*/

 /* To match the published algorithm, replace this line with
   case 'l': if (ends("\04" "abli")) { r("\04" "able"); break; } */

             if (ends("\04" "alli")) { r("\02" "al"); break; }
             if (ends("\05" "entli")) { r("\03" "ent"); break; }
             if (ends("\03" "eli")) { r("\01" "e"); break; }
             if (ends("\05" "ousli")) { r("\03" "ous"); break; }
             break;
   case 'o': if (ends("\07" "ization")) { r("\03" "ize"); break; }
             if (ends("\05" "ation")) { r("\03" "ate"); break; }
             if (ends("\04" "ator")) { r("\03" "ate"); break; }
             break;
   case 's': if (ends("\05" "alism")) { r("\02" "al"); break; }
             if (ends("\07" "iveness")) { r("\03" "ive"); break; }
             if (ends("\07" "fulness")) { r("\03" "ful"); break; }
             if (ends("\07" "ousness")) { r("\03" "ous"); break; }
             break;
   case 't': if (ends("\05" "aliti")) { r("\02" "al"); break; }
             if (ends("\05" "iviti")) { r("\03" "ive"); break; }
             if (ends("\06" "biliti")) { r("\03" "ble"); break; }
             break;
   case 'g': if (ends("\04" "logi")) { r("\03" "log"); break; } /*-DEPARTURE-*/
```

```
/* To match the published algorithm, delete this line */

} }

/* step3() deals with -ic-, -full, -ness etc. similar strategy to step2. */

void step3() { switch (b[k])
{
   case 'e': if (ends("\05" "icate")) { r("\02" "ic"); break; }
             if (ends("\05" "ative")) { r("\00" ""); break; }
             if (ends("\05" "alize")) { r("\02" "al"); break; }
             break;
   case 'i': if (ends("\05" "iciti")) { r("\02" "ic"); break; }
             break;
   case 'l': if (ends("\04" "ical")) { r("\02" "ic"); break; }
             if (ends("\03" "ful")) { r("\00" ""); break; }
             break;
   case 's': if (ends("\04" "ness")) { r("\00" ""); break; }
             break;
} }

/* step4() takes off -ant, -ence etc., in context <c>vcvc<v>. */

void step4()
{  switch (b[k-1])
   {  case 'a': if (ends("\02" "al")) break; return;
      case 'c': if (ends("\04" "ance")) break;
                if (ends("\04" "ence")) break; return;
      case 'e': if (ends("\02" "er")) break; return;
      case 'i': if (ends("\02" "ic")) break; return;
      case 'l': if (ends("\04" "able")) break;
                if (ends("\04" "ible")) break; return;
      case 'n': if (ends("\03" "ant")) break;
                if (ends("\05" "ement")) break;
                if (ends("\04" "ment")) break;
                if (ends("\03" "ent")) break; return;
      case 'o': if (ends("\03" "ion") && (b[j] == 's' || b[j] == 't')) break;
                if (ends("\02" "ou")) break; return;
                /* takes care of -ous */
      case 's': if (ends("\03" "ism")) break; return;
      case 't': if (ends("\03" "ate")) break;
                if (ends("\03" "iti")) break; return;
      case 'u': if (ends("\03" "ous")) break; return;
```

```
      case 'v': if (ends("\03" "ive")) break; return;
      case 'z': if (ends("\03" "ize")) break; return;
      default: return;
    }
    if (m() > 1) k = j;
}

/* step5() removes a final -e if m() > 1, and changes -ll to -l if
   m() > 1. */

void step5()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/* In stem(p,i,j), p is a char pointer, and the string to be stemmed is from
   p[i] to p[j] inclusive. Typically i is zero and j is the offset to the last
   character of a string, (p[j+1] == '\0'). The stemmer adjusts the
   characters p[i] ... p[j] and returns the new end-point of the string, k.
   Stemming never increases word length, so i <= k <= j. To turn the stemmer
   into a module, declare 'stem' as extern, and delete the remainder of this
   file.
*/

int stem(char * p, int i, int j)
{  b = p; k = j; k0 = i; /* copy the parameters into statics */
   if (k <= k0+1) return k; /*-DEPARTURE-*/

   /* With this line, strings of length 1 or 2 don't go through the
      stemming process, although no mention is made of this in the
      published algorithm. Remove the line to match the published
      algorithm. */

   step1ab(); step1c(); step2(); step3(); step4(); step5();
   return k;
}

/*--------------------stemmer definition ends here------------------------*/
```

```
stemArray(char **list, int arrayLen)
{
int i;
for (i = 0; i < arrayLen; i++)
        if (list[i] != NULL)
                stemWord(list[i]);
}

stemWord( char * s)
{
s[stem(s,0, strlen(s) - 1) + 1] = 0;
}
```

dialog.c: This is main program of dialog module

```
/*****************************************************************************
dialog.c : The main function for the interactive dialog program. loads all
the global arrays and variables before calling the interacitve function.

Arguments are:
1. The Configuration file for Thesaurus. contains Prompts, index,
  basic thesaurus etc.
2. The Learning Thesaurus. - used to store learnt words and to refer to them.
*****************************************************************************/
#include <stdio.h>
#include <string.h>
#include "arraylib.h"

int numColumn,  numRow, numIndex, numMenu;
int startPoint, eofFlag, topValues;
char **rowTerms, **columnTerms, **prompts, **stopWords;
float **matrix, **cosine, phoneThreshold, webThreshold;
int **indexList, **menuList, **thesaurus;
int numStopWord, numOrgRow;
int numForms, numPF;
struct form **formlist;
char ***Fprompts, *formfile;
int **scoring, numScore = 0;
char *cfg, *lcfg, *fdoc, *xdoc, *sdoc;
int minPromptCount = 1, timeout = 30;

void Interactive(char *);
```