```
//*************************************************//
//                                                 //
//        IRA.h                                    //
//           Head file for IRA                     //
//*************************************************//
//
//   Author:   Hui Jin
//   Modification:    Mar 10, 2000 First version
//   This file defines the class of Irregular Repeat Accumulate code.
//   the code essentially is a concatenate of LDGM code and Accumulate
code.


#ifndef  _IRA_H
#define  _IRA_H

#include <iostream.h>
#include <stdlib.h>

#include "ldpc.h"
#include "node.h"
#include "random.hh"

class   IRAcode
{
   private:
     RandomGenerator  *Rand;    //random generator, used to simulate
noise.

     int    infobits_len;                 //length of information
bits
     int    codebits_len;                 //length of code bits
     int    checkbits_len;                //length of check bits
     double rate;                         //related by rate of this
code.

     int    LEdge_num;            //edges number on the left. LDGM
part.
     int    REdge_num;            //edges number on the right, Acc.
part.

     int   *Interleaver;     // the permutation of LDGM code. Length
is
                             //  LEdge_num.
     double *y_r;       //received codebits. Length codebits_len.
     double sigma;      //sigma, noise standard deviation, channel.


     degree_sequence *pLV;       //LDGM variable degree profile.
     degree_sequence *pLC;       //LDGM check degree profile.
```

```
                                   // passed by calling function.

     double *   LMessage;     // the message updated between left
                               //  variable and check nodes.
                               //  Length should be LEdge_num.
     double *   RMessage;     // The message update between check
nodes and
                               //  variable nodes on right.

     int * Lbelief;          //the belief from left variables, through
edges.
     int * Rbelief;          //the belief from right varibles, through
edges.

     var_node *RVarNode;    //for LDPC codes, only L exists.
     var_node *LVarNode;
     check_node * CheckNode; //check nodes.

     void getYrAWGN();




 //initialize Left variable nodes.
     void initLVNode(int nv[]);
 //initialize right variable nodes.
     void initRVNode();

 //initialize check nodes.
     void initCNode();
     void initMessage();
     void initM0();

//debug function.


   public:
     IRAcode() { }
     IRAcode(int info_len,  int check_len, int _LEdgenum, int nv[],
degree_sequence * _pLV, degree_sequence * _pLC, \
       double _sigma, RandomGenerator *_Rand, int *_Interleaver);


     ~IRAcode() {
                delete [] LMessage;
                delete [] RMessage;
                delete [] Lbelief;
                delete [] Rbelief;
                delete [] RVarNode;
                delete [] LVarNode;
```

```
                delete [] CheckNode;
                delete [] Interleaver;
                delete [] y_r;
                 }

     void update_message();//update message by message passing.

     int   wrong_bits();   // total wrong bits after belief
propagation is stoped.
     bool  state_check();  //1 if every parity check is satisfied.
                           //0 otherwise
     void iteration(int num);    // do message passing in num
iterations.
     int  smartIteration();      // do iterations in a smart way.


//debug function.
     void displayConnection();
     void displayMessage();
};

/*
this function gets the received sequence after AWGN channel. Given
sigma, and
assuming all zero is transmitted.
*/

inline void IRAcode::getYrAWGN() {
    int codeseq[codebits_len];
    for(int i=0; i<codebits_len; i++)
      codeseq[i]=0;

     Rand->binAWGNchannel(codeseq, codebits_len, sigma, y_r);
}


/*
This function initializes all the message to zero.
*/

inline void IRAcode::initMessage()
{
   int i;
   for(i=0; i<LEdge_num; i++)
       {
         LMessage[i]=0;
         Lbelief[i]=1;
        }
   for(i=0; i<REdge_num; i++)
       {
```

```
        RMessage[i]=0;
        Rbelief[i]=1;
        }
}


/*
this function displays the connection between nodes.
*/

inline void IRAcode::displayConnection()
{
   int i;
   for(i=0; i<infobits_len; i++)
         LVarNode[i].display_edgeid();
   for(i=0; i<checkbits_len; i++)
         {
         RVarNode[i].display_edgeid();
         CheckNode[i].display_edgeid();
         }

}

/*
this function displays the message on all edges.
*/
inline void IRAcode::displayMessage()
{
   int i;
   for(i=0; i<LEdge_num; i++)
      cout << i << " " << LMessage[i] << "   " << endl;
   for(i=0; i<REdge_num; i++)
       cout << i << " " << RMessage[i] << "   " << endl;
}


#endif  _IRA_H
```