Type signature

From Wikipedia, the free encyclopedia

In computer science, a **type signature** or **type annotation** defines the inputs and outputs for a function, subroutine or method. A type signature includes the number of arguments, the types of arguments and the order of the arguments contained by a function. A type signature is typically used during overload resolution for chosing the correct definition of a function to be called among many overloaded forms.

Contents

- 1 Examples
  - 1.1 C/C++
  - 1.2 Erlang
  - 1.3 Haskell
  - 1.4 Java
- 2 Signature
- 3 Method signature
- 4 Examples
  - 4.1 C/C++
  - 4.2 C#
  - 4.3 Java
  - 4.4 Objective-C
- 5 References

Examples

C/C++

In C and C++, the type signature is declared by what is commonly known as a function prototype. In C/C++, a function declaration reflects its use; for example, a function pointer that would be invoked as:

```
char c;
double d;
int retVal = (*fPtr)(c, d);
```

has the signature:

```
(int) (char, double);
```

Erlang

In Erlang, type signatures may be optionally declared, as:

```
-spec(function_name(type1(), type2(), ...) -> out_type()).
```

For example:

```
-spec(is_even(number()) -> boolean()).
```

Haskell

A type signature in the Haskell programming language is generally written in the following format:

```
functionName :: arg1Type -> arg2Type -> ... -> argNType
```

Notice that the type of the result can be regarded as everything past the first supplied argument. This is a consequence of currying, which is made possible by Haskell's support for first-class functions; this function requires two inputs where one argument supplied and the function is "curried" to produce a function for the argument not supplied. Thus calling `f x`, where `f :: a -> b -> c`, yields a new function `f2 :: b -> c` that can be called `f2 b` to produce `c`.

The actual type specifications can consist of an actual type, such as `Integer`, or a general type variable that is used in parametric polymorphic functions, such as `a`, or `b`, or `anyType`. So we can write something like: `functionName :: a -> a -> ... -> a`

Since Haskell supports higher-order functions, functions can be passed as arguments. This is written as: `functionName :: (a -> a) -> a`

This function takes in a function with type signature `a -> a` and returns data of type `a` out.

Java

In the Java virtual machine, *internal type signatures* are used to identify methods and classes at the level of the virtual machine code.

Example: The method `String String.substring(int, int)` is represented in bytecode as `Ljava/lang/String/substring(II)Ljava/lang/String;`. The signature of `main()` method looks like this:

```
public static void main(String[] args)
```

And in the disassembled bytecode, it takes the form of `Lsome/package/Main/main: ([Ljava/lang/String;)V`

The method signature for the `main()` method contains three modifiers:

- `public` indicates that the `main()` method can be called by any object.
- `static` indicates that the `main()` method is a class method.
- `void` indicates that the `main()` method has no return value.

Signature

A function signature consists of the function prototype. It specifies the general information about a function like the name, scope and parameters. Many programming languages use name mangling in order to pass along more semantic information from the compilers to the linkers. In addition to mangling, there is an excess of information in a function signature (stored internally to most compilers) which is not readily available, but may be accessed.[1]

Understanding the notion of a function signature is an important concept for all computer science studies.

- Modern object orientation techniques make use of interfaces, which are essentially templates made from function signatures.
- C/C++ uses function overloading with various signatures.

The practice of multiple inheritance requires consideration of the function signatures to avoid unpredictable results.

Computer science theory, and the concept of polymorphism in particular, make much use of the concept of function signature.

In the C programming language signature is roughly equivalent to its prototype definition.

The term "signature" may carry other meanings in computer science. For example:

- File signatures can identify or verify the content of a file.
- Database signatures can identify or verify the schema or a version of a database.
- In the ML family of programming languages, "signature" is used as a keyword referring to a construct of the module system that plays the role of an interface.

Method signature

In computer programming, especially object-oriented programming, a method is commonly identified by its unique **method signature**, which usually includes the method name, and the number, types and order of its parameters.[2] A method signature is the smallest type of a method.

Examples

C/C++

In C/C++, the method signature is the method name and the number and type of its parameters, but it is possible to have a last parameter that consists of an array of values:

```
int printf(const char*, ... );
```

Manipulation of these parameters can be done by using the routines in the standard library header `<stdarg.h>`.

C#

Similar to the C syntax, C# sees as the method signature its name and the number and type of its parameters, where the last parameter may be an array of values:[3]

```
void Add(out int sum, params int[] value);
[...]
Add(out sum, 3, 5, 7, 11, -1);  // sum == 25
```

Java

In the Java programming language, a method signature is the method name and the number and type of its parameters. Return types and thrown exceptions are not considered to be a part of the method signature.

```
methodName(parameters) {...};
```

For example, the following two methods have distinct signatures:

```
doSomething(int y);
doSomething(String y);
```

The following three methods do have the same signatures and are considered the same, as only the return value differs. The name of the parameter is not part of the method signature and is ignored by the compiler for checking method uniqueness.

```
int doSomething(int y)
String doSomething(int x)
int doSomething(int z) throws java.lang.Exception
```

Objective-C

In the Objective-C programming language, method signatures for an object are declared in the interface header file. For example,

```
- (id)initWithInt:(int)value;
```

defines a method `initWithInt` that returns a general object (an `id`) and takes one integer argument. Objective-C only requires a type in a signature to be explicit when the type is not `id`; this signature is equivalent:

```
- initWithInt:(int)value;
```

References

1. "C++ Reference: Programming terms". Retrieved 3 December 2013.
2. Paul Leahy. "Method Signature". http://www.about.com/: About.com Guide. Retrieved 2011-05-31. "A method signature is part of the method declaration. It is the combination of the method name and the parameter list."
3. Mössenböck, Hanspeter (2002-03-25). "Advanced C#: Variable Number of Parameters" (PDF). http://ssw.jku.at/Teaching/Lectures/CSharp/Tutorial/: Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik. p. 52. Retrieved 2011-08-03.

Categories: Type theory | Subroutines
