C++ Functions

User Defined Functions

In order to make your programming task more structured and to promote code reuse, it is common to write your own subroutines, called functions. It is common to include these functions in the same source file as the main() function. Let's start with a function that returns the larger of two integers.

```
int larger ( int i , int j ){
        return i > j ? i : j ;
}
```

We would want to use this function in a program like this:

```
cout << "The larger of " << i << " and " << j << " is : " << larger ( i , j ) ;
```

When the compiler reaches this call to the larger function, it must verify that there does indeed exist a function named larger that takes two integer parameters. For this reason, we must include a function prototype in or source file at a point before the function call appears. The function prototype consists of the function signature only:

```
int larger (int , int ) ;
```

Although not required, it is common to include names for the individual parameters, like this:

```
int larger ( int i , int j ) ;
```

Function Overloading

As an example of code reuse, we can use our larger function to construct a new function that returns the larger of three integers. This is also an example of ***function overloading***, in which two different functions have the same name, but differ in the number or type of parameters. This new larger function looks like this:

```
int larger ( int i , int j , int k){
        return larger ( i , larger ( j , k) ) ;
}
```

Programming Example: Larger.cpp

Scope Rules

There are two type of identifiers in C++: local identifiers and global identifiers

1. Local Identifiers - Declared inside a block {} or a function. These identifiers are not accessible outside of that block or function in which they are declared.

2. Global Identifiers - Declared outside all blocks and functions. These identifiers are accessible from anywhere in the program. It is a good idea to avoid using global identifiers since they lead to a high degree of coupling.

Programming Example: Global1.cpp

Whenever a variable is created, it is placed in a particular scope.

It is illegal to have two variables with the same name in the same scope.

However, two variables that are in different scopes can have the same name.

When you refer to a variable name in your program, the following rules are used to decide which variable should be accessed:

1. Global variables can be accessed from inside a function provided both of the following are true:

   (a) The global identifier is declared before the function is implemented. The variable declaration can come before or after the function prototype, but it must come before the function implementation.

   (b) The global variable is not hidden. The following things can hide a global identifier

      i. A function parameter with the same name as the global identifier
      ii. A function variable with the same name as the global identifier

   Programming Example: Global2.cpp

2. Inside a nested block, an identifier can be accessed
   (a) Only from within that block, from the point of declaration forward
   (b) From within more deeply nested blocks, provided it is not hidden inside those blocks

   Programming Example: Scope.cpp

Default Arguments

You can specify default values for the argument of your functions. If a function call is made with some of the arguments omitted, then the default values are used. The arguments with default values must be the right most arguments in the function signature.

Programming Example: DefaultArgs.cpp

Function Overloading

C++ allows you two write multiple functions with the same name. You just need to make sure that either the *number* of parameters is different in each function, or that the *type* of the parameters is different in each function.

Suppose your program is using **int**, **long**, **float** and **double** variables. And suppose you want a function to compute the square of each of these variables. Without function overloading you would need to have the following functions:

```
int squareInt(int x);
long squareLong(long x);
float squareFloat(float x);
double squareDouble(double x);
```

Function overloading allows you to use the same name for each of these four functions (since the type of the parameters is different). You can now have these four functions:

```
int square(int x);
long square(long x);
float square(float x);
double square(double x);
```

Programming Example: Overloading.cpp

Reference Parameters

When an argument is passed to a function, C++ can pass:

1. The *value* of the argument – Called "Pass by Value"
2. The *memory address* of the argument – Call "Pass by Reference"

See the Memory Address

To see the memory address in which a variable is stored, you use the & operator.

Programming Example: References1.cpp

Changing the Actual Parameter

When Pass by Value is used

The value of the actual parameter *cannot* be changed by the function.

When Pass by Reference is used

The value of the actual parameter *can* be changed by the function.

A function knows to use Pass by Reference from the **& in its signature**.

The function calls for Pass by Value and for Pass by Reference are **identical**.

The bodies of the two functions are **identical**.

Programming Example: References2.cpp

Return Multiple Values

Reference parameters can be used to return more than one value from a function.

Programming Example: References3.cpp

Prevent Changes with "const"

When a large object needs to be passed to a function, Pass by Reference can greatly improve performance.

Changes to the object can be prevented by using const.

Programming Example: References4.cpp

Creating an Alias

References can also be used to create an alias for a variable. In other words, two variables can be tied to the same memory address.
Reference variables must be initialized when they are declared. Once created, a reference variable cannot be changed.

Programming Example: References5.cpp

Function Templates

Function templates are used for generic programming.

The function is compiled multiple times, once for each different type that is actually used with that function.

Programming Example: LargestTemplate.h and LargestTemplate.cpp