Sep 10, 2015

COMPUTER PROGRAMMING

LAB 5

ABDUL AZIZ

FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Lab Instructor	**Mr. Abdul Aziz**
Course	**Computer Programming Lab**
Duration	**2hrs**

Objectives:

In this lab, following topics will be covered

- ❖ Introduction Functions
- ❖ Function Terminologies
- ❖ Difference between call by value and call by reference
- ❖ Understanding Function Overloading
- ❖ Inheritance
- ❖ Multiple Inheritance
- ❖ Constant member function
- ❖ Static member function

1. Introduction to Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- **Function Body:** The function body contains a collection of statements that define what the function does.

2. Function Definitions (Terminologies)

Function - A subprogram in C++.

Function call (function invocation) - A program's request for the service of a particular function.

Function definition - A function declaration that includes the body of the function.

Function prototype - A function declaration without the body of the function.

Function value type - The data type of the result value returned by the function.

Void function - A function that does not return a value**.**

Call by value – A method used by a function that uses a parameter that receives a copy of the value of the corresponding argument.

Call by reference - Is a method used by a function that uses a parameter that receives a location (memory address) of the corresponding argument.

/* Example */

```
#include <iostream>
using namespace std;
                                // function declaration
int max(int num1, int num2);

 int main ()

{                       // local variable declaration:
  int a = 100;
  int b = 200;
  int ret;
                        // calling a function to get max value.

  ret = max(a, b);
  cout << "Max value is : " << ret << endl;
  return 0;

}
```

```
                                // function returning the max between two numbers
int max(int num1, int num2)
{                                // local variable declaration
  int result;
   if (num1 > num2)
     result = num1;
  else
     result = num2;
   return result;

}
```

3. Function call by value

In the call-by-value method, the value of an argument is copied into the formal parameter of the function. With this, changes made to the parameters of the function will not affect the arguments used to call it. Take a look at the following program, which adds ten to a number and uses the call-by-value method. Note that the argument is not affected:

```
#include <iostream>
using namespace std;

int ChangeIt(int value);    // Function Prototype
int main()
{
 int number = 20;
cout << "Value of argument *before* function call: " << number << endl;

 cout << "The Number returned from the function: " << ChangeIt(number)
<<
endl;

 cout << "Value of argument *after* function call: " << number << endl;

 return 0;

}

int ChangeIt(int value)

{
  //Description: This function changes a value through the use of
  //            call-by-value
  //Precondition: value is given as an input.
  //Post-condition: ChangeIt returns value + 10

 value += 10;
 return value;

}
```

4. Function call by reference

The call-by reference method of passing an argument does not pass a value to the function. Instead, this method passes the address of a value to a function. Therefore, when this method of passing an argument is used, changes made to the parameters of a function will affect the argument used to call the function. The way we achieve this method of argument passing is by using the reference operator (&). We will now look at a sample program that adds ten to a number. Note that the argument is affected, the original value of the variable is changed:

```
#include <iostream>
using namespace std;

int ChangeIt(int &value);    // Function Prototype
int main()
{
 int number = 20;
cout << "Value of argument *before* function call: " << number << endl;

 cout << "The Number returned from the function: " << ChangeIt(number)
<<
endl;

 cout << "Value of argument *after* function call: " << number << endl;

 return 0;

}

int ChangeIt(int &value)

{
  //Description: This function changes a value through the use of
  //            call-by-value
  //Precondition: value is given as an input.
  //Post-condition: ChangeIt returns value + 10

 value += 10;
 return value;

}
```

5. Function Overloading

Function overloading means two or more functions can have the same name but either the number of arguments or the data type of arguments has to be different.

/* Example */

```
#include <iostream>
using namespace std;
class printData
{
  public:
    void print(int i)
      {
      cout << "Printing int: " << i << endl;
    }

    void print(double  f)
      {
      cout << "Printing float: " << f << endl;
    }
    void print(char* c) {
      cout << "Printing character: " << c << endl;
    }
};

int main(void)
{
  printData pd;
                                // Call print to print integer
  pd.print(5);
                                // Call print to print float
  pd.print(500.263);
                               // Call print to print character
  pd.print("Hello C++");
  return 0;
}
```

6. Multiple Inheritance

To use multiple inheritance, simply specify each base class (just like in single inheritance), separated by a comma. **Multiple inheritance** enables a derived class to inherit members from more than one parent.

For example: A class*Rectangle* is derived from base classes *Area* and *Circle*.

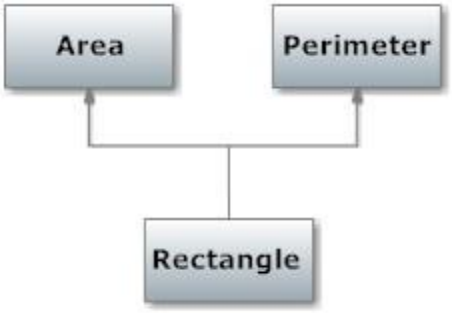

Figure: Multiple Inheritance Example

```
#include <iostream>
using namespace std;
class Area
{
  public:
    float area_calc(float l,float b)
    {
       return l*b;
    }
};

class Perimeter
{
  public:
    float peri_calc(float l,float b)
    {
       return 2*(l+b);
    }
};

/* Rectangle class is derived from classes Area and Perimeter. */
class Rectangle : private Area, private Perimeter
{
    private:
       float length, breadth;
    public:
      Rectangle() : length(0.0), breadth(0.0) { }
      void get_data( )
      {
         cout<<"Enter length: ";
         cin>>length;
         cout<<"Enter breadth: ";
         cin>>breadth;
      }

      float area_calc()
      {
      /* Calls area_calc() of class Area and returns it. */

         return Area::area_calc(length,breadth);
```

```
    }

    float peri_calc()
    {
    /* Calls peri_calc() function of class Perimeter and returns it. */

       return Perimeter::peri_calc(length,breadth);
    }
};


int main()
{
   Rectangle r;
   r.get_data();
   cout<<"Area = "<<r.area_calc();
   cout<<"\nPerimeter = "<<r.peri_calc();
   return 0;
}
```

Exercise

1. Imagine a publishing company that markets both book and audiocassette versions of its works. Create a class publication that stores the title (string) and price (float) of a publication. From this class derive two classes **book**, which adds a page_count(int) and **tape**, which adds a playing time in minutes(float).Each of these three classes should have a function getdata() to getdata and putdata() to display its data.

   Write a main() program to test the book and tape classes by creating instances of them, asking the user to fill in data with getdata(), and then displaying the data with putdata().

2. Write a program that computes and displays the charges for a patient's hospital stay. First, the program should ask if the patient was admitted as an in-patient or an out-patient. If the patient was an in-patient the following data should be entered:
   - The number of days spent in the hospital
   - The daily rate
   - Charges for hospital services (lab tests, etc.)
   - Hospital medication charges.

   If the patient was an out-patient the following data should be entered:
   - Charges for hospital services (lab tests, etc.)
   - Hospital medication charges.

   The program should use two overloaded functions to calculate the total charges. One of the functions should accept arguments for the in-patient data, while the other function accepts arguments for out-patient data. Both functions should return the total charges.