FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

CL217 – OBJECT ORIENTED PROGRAMMING LAB

LAB MANUAL # 05

INSTRUCTOR: MUHAMMAD ABDULLAH

DEPARTMENT OF COMPUTER SCIENCE

SEMESTER SPRING 2021

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Functions:

Functions are like building blocks. They let you divide complicated programs into manageable pieces. They have other advantages, too:

1. While working on one function, you can focus on just that part of the program and construct it, debug it, and perfect it.
2. Different people can work on different functions simultaneously.
3. If a function is needed in more than one place in a program or in different programs, you can write it once and use it many times.
4. Using functions greatly enhances the program's readability because it reduces the complexity of the function main .

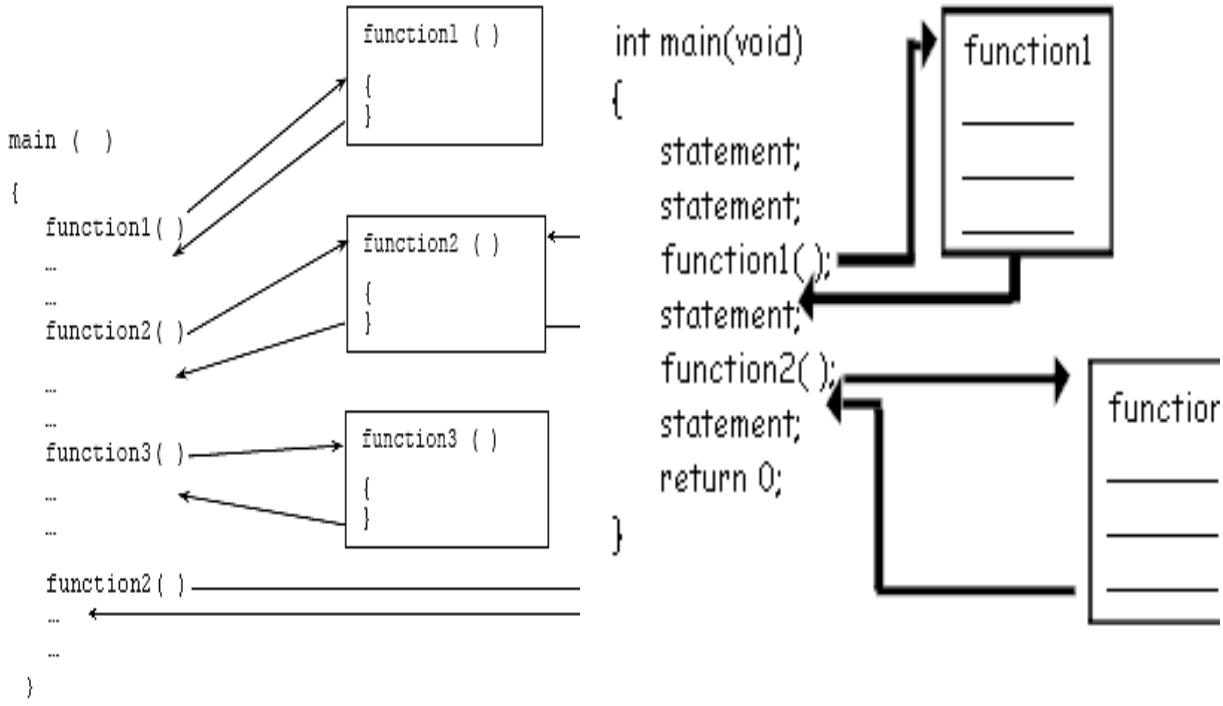


Figure 1: Functions

```
#include <iostream>
#include <cctype>
#include <iomanip>
using namespace std;
double getNumber();
char identifyInput();
double getKilos (double);
double getMeters (double);
int main()
{
```

```
double numberIn = 0.0, kilos = 0.0, meters = 0.0, answer = 0.0 ;
char choice;
numberIn = getNumber();
choice = identifyInput();
if (choice =='P')
{
answer = getKilos (numberIn);
cout << numberIn << " pounds are equivalent to " << answer << "
Kilograms.\n";
}
else
{
answer = getMeters (numberIn);
cout << numberIn << " yards are equivalent to " << answer << "
Meters.\n";
}
system("pause");
return 0;
}
double getNumber()
{
double input;
do
{
cout << "Enter a number greater than 0 that represents a measurement
in either pounds or yards: ";
cin >> input;
}while (input <= 0);
return input;
}
char identifyInput()
{
char option;
do
{
cout << "Enter a P if the number you input was pounds and Y if the
number input was yards: ";
cin >> option;
option = toupper(option);
}while(!(option == 'P' || option == 'Y'));
return option;
}
double getKilos (double lbs)
{
double result;
result = lbs * 0.45359237;
```

```
return result;
}
double getMeters ( double yds)
{
double result;
result = yds * 0.9144;
return result;
}
```

Output

```
/* Sample Run-I:
Enter a number greater than 0 that represents a measurement in either pounds or yards: 512
Enter a P if the number you input was pounds and Y if the number input was yards: P
512.00 pounds are equivalent to 232.24 Kilograms.

Sample Run-II:
Enter a number greater than 0 that represents a measurement in either pounds or yards: 512
Enter a P if the number you input was pounds and Y if the number input was yards: Y
512.00 yards are equivalent to 468.17 Meters.
Press any key to continue . . .
*/
```

Functions Overloading

Creating several functions with the same name but different formal parameters.
Two functions are said to have different formal parameter lists if both functions have:

- A different number of formal parameters or
- If the number of formal parameters is the same, then the data type of the formal parameters, in the order you list them, must differ in at least one position.

If a function's name is overloaded, then all of the functions in the set have the same name. Therefore, all of the functions in the set have different signatures if they have different formal parameter lists. Thus, the following function headings correctly overload the function functionXYZ:

```
void functionXYZ()
void functionXYZ(int x, double y)
void functionXYZ(double one, int y)
void functionXYZ(int x, double y, char ch)
```

Consider the following function headings to overload the function functionABC:

```
void functionABC(int x, double y)
int functionABC(int x, double y)
```

Both of these function headings have the same name and same formal parameter list. Therefore, these function headings to overload the function functionABC are incorrect. In this case, the compiler will generate a syntax error.

```
#include <iostream>
#include <conio.h>
using namespace std;
void repchar();
void repchar(char);
void repchar(char, int);
void main()
{
repchar();
repchar('=');
repchar('+', 30);
system("pause");
}
void repchar()
{
for(int j=0; j<45; j++)
cout << '*';
cout << endl;
}
void repchar(char ch)
{
for(int j=0; j<45; j++)
cout << ch;
cout << endl;
}
void repchar(char ch, int n)
{
for(int j=0; j<n; j++)
cout << ch;
cout << endl;
}
```

Output

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
```

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

void sum(int x, int y)  // formal arguments
    {
    cout<<"sum of int is: "<<(x+y)<<endl;
    }
void sum(double x, double y)  // formal arguments
    {
    cout<<"sum of double is: "<<(x+y)<<endl;
    }
void sum(int x, double y)  // formal arguments
    {
    cout<<"sum of int & double is: "<<(x+y)<<endl;
    }
void sum(double x, int y)  // formal arguments
    {
    cout<<"sum of double & int is: "<<(x+y)<<endl;
    }

int main() {
    sum(3,5);
    sum(3.3,5.6);
    sum(3,5.4);
    sum(3.6,5);

  return 0;
}
```

Output

Output:
sum of int is: 8
sum of double is: 8.9
sum of int & double is: 8.4
sum of double & int is: 8.6

Functions with Default Parameters

- The default parameter is a way to set default values for function parameters a value is no passed in (i.e. it is undefined).
- In C++ programming, we can provide default values for function parameters.

- If a function with default arguments is called without passing arguments, then the default parameters are used.
- However, if arguments are passed while calling the function, the default arguments are ignored.

```
#include <iostream>
#include <conio.h>
using namespace std;
void repchar(char='*', int=45);
int main()
{
repchar();
repchar('=');
repchar('+', 30);
system("pause");
}
void repchar(char ch, int n)
{
for(int j=0; j<n; j++)
cout << ch;
cout << endl;
}
```

Output

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
Press any key to continue . . .
```

How to pass arguments to functions

We can pass arguments to functions by two ways:

1. Passing by values.
2. Passing by references.

1. Pass by Values

Passing arguments in such a way where the function creates copies of the arguments passed to it , it is called passing by value.

When an argument is passed by value to a function, a new variable of the data type of the argument is created and the data is copied into it. The function accesses the value in the newly created variable and the data in the original variable in the calling function is not changed.

```
#include <iostream>
#include<iostream>

using namespace std;
void sum(int x, int y)

{

    int sum =x+y;

    cout<<"Result is: "<<sum;

}
int main()

{

   int a=5;

    int b=6;

    sum(a,b);

    return 0;

}
```

Output:

Result is: 11

2. Pass by Reference

The data can also be passed to a function by reference of a variable name and that contains data.

The reference provides the second name (or **alias**) for a variable name.

Alias: Two variables refer to the same thing or entity. Alias are the alternate name for referring to the same thing.

When a variable is passed by reference to a function, no new copy of the variable is created. Only the address of the variable is passed to the function.

The original variable is accessed in the function with reference to its second name or alias. Both variables use the same memory location.

Thus, any change in the reference variable also changes the value in the original variable.

The reference parameters are indicated by an ampersand (&) sign after the data type both in the function prototype and in the function definition.

```
#include <iostream>

using namespace std;


void swapNums(int &x, int &y) {
  int z = x;

  x = y;

  y = z;
  }


int main() {

  int firstNum = 10;

  int secondNum = 20;
  cout << "Before swap: " << "\n";

  cout << firstNum << secondNum << "\n";
  swapNums(firstNum,  secondNum);
  cout << "After swap: " << "\n";

  cout << firstNum << secondNum << "\n";
  return 0;

}
```

Output:

```
Before swap:
10 20

After swap:
20 10
```

The ampersand sign (&) is also used with the data type of **"x"** and **"y"** variables. The **x** and **y** are the aliases of "firstNum" and "secondNum" variables respectively.

The memory location of **"x"** and "firstNum" is the same and similarly, memory location of **"y"** and "secondNum" is same.

How to pass 1D array to Function

```
#include<iostream>

using namespace std;
void arrayIterationFunction(int test[])

{

    for (int i = 0; i <6 ; i++)

    {

        cout<<"myNum["<<i<<"]="<<test[i]<<endl;

    }

}
int main()

{

    int myNum[]= {1,3,4,5,6,7};

    arrayIterationFunction(myNum);
}
```

Output:

myNum[0]=1
myNum[1]=3
myNum[2]=4
myNum[3]=5
myNum[4]=6
myNum[5]=7

```
#include<iostream>

using namespace std;
int array_size;    // initialization at the top

void arrayIterationFunction(int test[])

{

   for (size_t i = 0; i <array_size; i++) //loop for insertion in array

    {

        cout<<"Enter value at test["<<i<<"]=";

        cin>>test[i];

    }

    for (int i = 0; i <array_size; i++) //loop to show value of array

    {

        cout<<"myNum["<<i<<"]="<<test[i]<<endl;

    }

}

int main()

{

    cout<<"Enter Array Size: ";

    cin>>array_size;

    int myNum[array_size];
    arrayIterationFunction(myNum);
}
```

Output:

Enter Array Size: 4
Enter value at test[0]=3
Enter value at test[1]=2
Enter value at test[2]=1
Enter value at test[3]=6

myNum[0]=3
myNum[1]=2
myNum[2]=1
myNum[3]=6

Find Maximum Value in 1 D Array

```
#include <iostream>

using namespace std;
int main() {


int array_size= 5;

int myNum[array_size]= {11,10,33,4,5};

int max =myNum[0];


for (int i = 0; i <array_size; i++)

{

    if (myNum[i] > max)

    {

        max=myNum[i];

    }

}

cout<<"Maximum number is: "<<max;


  return 0;

}
```

Output:

```
Maximum number is: 33
```

Find Minimum Value in 1 D Array

```
#include <iostream>

using namespace std;
int main() {


int array_size= 5;

int myNum[array_size]= {11,10,33,4,5};

int min =myNum[0];


for (int i = 0; i <array_size; i++)

{

    if (myNum[i] < min)

    {

        min=myNum[i];

    }


}

cout<<"Manimum number is: "<<min;
  return 0;

}
```

Output:

```
Manimum number is: 4
```

Pointers

Pointers are the most powerful feature of C and C++. These are used to create and manipulate data structures such as linked lists, queues, stacks, trees etc. The virtual functions also require the use of pointers. These are used in advanced programming techniques. To understand the use of pointers, the knowledge of memory locations, memory addresses and storage of variables in memory is required.

Memory addresses & Variables

Computer memory is divided into various locations. Each location consists of 1 byte. Each byte has a unique address.

When a program is executed, it is loaded into the memory from the disk. It occupies a certain range of these memory locations. Similarly, each variable defined in the program occupies certain memory locations. For example, an int type variable occupies two bytes and float type variable occupies four bytes.

When a variable is created in the memory, three properties are associated with it. These are:

- Type of the variable
- Name of the variable
- Memory address assigned to the variable.

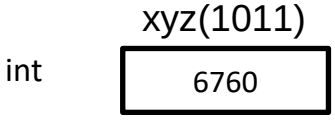
For example an integer type variable xyz is declared as shown below

int xyz =6760;

int represents the data type of the variable.

xyz represents the name of the variable.

When variable is declared, a memory location is assigned to it. Suppose, the memory address assigned to the above variable xyz is 1011. The attribute or properties of this variable can be shown as below:



The box indicates the storage location in the memory for the variable xyz. The value of the variable is accessed by referencing its name. Thus, to print the contents of variable xyz on the computer screen, the statement is written as:

```
cout<<xyz
```

The memory address where the contents of a specific variable are stored can also be accessed. The **address operator (&)** is used with the variable name to access its memory address. The address operatory (&) is used before the variable name.

For example, to print the memory address of the variable xyz, the statement is written as:

```
cout<<&xyz
```

The memory address is printed in hexadecimal format.

Pointer Variables

The variables that is used to hold the memory address of another variable is called a pointer variable or simply pointer.

The data type of the variable (whose address a pointer is to hold) and the pointer variable must be the same. A pointer variable is declared by placing an asterisk (*) after data type or before the variable name in the data type statement.

For example, if a pointer variable **"p"** is to hold memory address of an integer variable, it is declared as:

```
int* p;
```

Similarly, if a pointer variable **"rep"** is to hold memory address of a floating-point variable, it is declared as:

```
float* rep;
```

The above statements indicate that both "**p"** and **"rep"** variable are pointer variables and they can hold memory address of integer and floating-point variable respectively.

Although the asterisk is written after the data type, is usually more convenient to place the asterisk before the pointer variable. i.e. **float *rep;**

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

int main() {

int a, b;
int *x, *y;

a =  33;
b =  66;
x = &a;
y = &b;

cout<<"Memory address of variable a= "<<x<<endl;
cout<<"Memory address of variable b= "<<y<<endl;

  return 0;
}
```

Output:

```
Memory address of variable a= 0x7bfe0c
Memory address of variable b= 0x7bfe08
```

A pointer variable can also be used to access data of memory location to which it points.

In the above program, **x** and **y** are two pointer variables. They hold memory addresses of variables **a** and **b**. To access the contents of the memory addresses of a pointer variable, an asterisk (*) is used before the pointer variable.

For example, to access the contents of **a** and **b** through pointer variable **x** and **y**, an asterisk is used before the pointer variable. For example,

```
cout<<"Value in memory address x = "<<*x<<endl;
cout<<"Value in memory address y = "<<*y<<endl;
```