Functions and Arrays

Parameter-Passing Mechanisms

The general form of a C++ function heading is:

```
ReturnType Name ( ParameterDeclarationList )
```

where *ParameterDeclarationList* is an optional sequence of one or more *Parameter-Declarations* separated by commas, each of which has the form:

```
Type ParameterName
```

where *Type* is a valid type, and *ParameterName* is a valid identifier.

Terminology

1. A parameter whose *Type* **is not** followed by an ampersand (`&`) is called a **value parameter**. A value parameter is a variable, local to the function, such that when the function is called, it receives a copy of the value of the corresponding argument.
2. A parameter whose *Type* **is** followed by an ampersand (&) is called a **reference parameter**. A reference parameter is an alias (e.g., another name) for its corresponding argument.

Header Files and Functions

- Header files contain numerous frequently used functions that programmers can use without having to write codes for them.
- Programmers can also write their own declarations and functions and store them in header files which they can include in any program that may require them (these are called user-defined header file that contains user defined functions).

Example #1

```
// program showing function definition, declaration, call and
// the use of the return statement
#include <iostream>
using namespace std;
 float  avg (float, float);

int main ()
{
       // prototypes for the function avg() that main() is going to call
      float  y1, y2, avgy;
            y1=5.0;
       y2=7.0;
       avgy = avg(y1, y2);
       // calling the function avg() i.e. control passes
       // to avg() and the return value is assigned to avgy
       cout<<"\ny1 = "<<y1<<"\ny2 = "<<y2;
       cout<<"\nThe average is= "<<avgy<<endl;
       return 0;
}
```

Output:

Example #2

```
// calculating an area of triangle no need prototype
#include <iostream>
using namespace std;

 float triangle_area (float base, float height)
{
      float area;
      area = (0.5 * base * height);
      return area;}

int main()
{
      float b, h, a;
      b = 4;
      h = 6;
      a = triangle_area(b, h);
      cout<<"Area = (0.5*base*height)"<<endl;
      cout<<"where, base = 4, height = 6"<<endl;
      // compiler will substitute the inline function code.
      cout<<"Area =  "<<a<<endl;
      return 0;}
```

output:

Example #3

// Demonstrating local and global variables

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

// a function prototype
void  myFunction ();
// a global scope variables
int  x = 5, y = 7;

int  main()
{
cout<<"x = 5, y = 7, global scope\n";
cout<<"\nx within main: "<<x<<"\n";
 cout<<"y within main: "<<y<<"\n\n";
cout<<"Then function call....\n";
myFunction();
cout<< "Back from myFunction...\n\n";
cout<< "x within main again: "<<x<<"\n";
cout<< "y within main again: "<<y<<"\n\n";
return 0;
}

void  myFunction()
{
// a local scope variable
int  y = 10;
cout<<"\ny = 10, local scope\n"<<"\n";
cout<<"x within myFunction: "<<x<<"\n";
cout<<"y within myFunction: "<<y<<"\n\n";
}
```

Output:

Example #4

```
// demonstrates the use of default parameter values
#include <iostream>
using namespace std;

// a function prototype, width = 25 and height = 1, are default values
int AreaOfCube(int length, int width = 25, int height = 1);

int main()
{           // assigning new values
int length = 100;
int width = 50;
int height = 2;
int area;
area = AreaOfCube(length, width, height);
cout<<"First time function call, area = "<<area<<"\n";
area = AreaOfCube(length, width);
// height = 1, default value
cout<<"Second time function call, area = "<<area<<"\n";
area = AreaOfCube(length);
// width = 25, height = 1, default values
cout<<"Third time function call, area = "<<area<<"\n";
return 0;
}

AreaOfCube(int length, int width, int height)
{           return (length * width * height);      }
```

Output:

Example #5

Call by value

```
#include<iostream>
using namespace std;
// call by value
void change(int,int);
int main()
{
    int a,b;
    cout<<"Enter values for a and b \n";
    cin>>a>>b;
    change(a,b);
    cout<<"\n The values of a and b after executing the function  :";
    cout<<a<<" "<<b;
    return 0 ;}


void change(int c, int d)
{

    c=c*10;
    d=d+8;
    cout<<"\n The values of a and b inside the function : "<<c<<" "<<d;}
```

Output:

Call by Reference

Repeat the previous example using call by reference

```
#include<iostream>
using namespace std;
// call by refrence
void change(int &,int &);
int main()
{
    int a,b;
    cout<<"Enter values for a and b \n";
    cin>>a>>b;
    change(a,b);
    cout<<"\n The values of a and b after executing the function  :";
    cout<<a<<" "<<b;
    return 0 ;
}


void change(int &c, int &d)
{      c=c*10;
    d=d+8;
    cout<<"\n The values of a and b inside the function : "<<c<<" "<<d;
```

Output: