## CHAPTER ONE

### Functions

A function is block of code which is used to perform a particular task, for example let's say you are writing a large C++ program and in that program you want to do a particular task several number of times, like displaying value from 1 to 10, in order to do that you have to write few lines of code and you need to repeat these lines every time you display values. Another way of doing this is that you write these lines inside a function and call that function every time you want to display values. This would make you code simple, readable and reusable.

*Syntax of Function*

```
return_type function_name (parameter_list)
{
  //C++ Statements
}
```

Let's take a simple example to understand this concept.

**A simple function example**

```
#include <iostream>
using namespace std;
/* This function adds two integer values
 * and returns the result
 */int
sum(int num1, int num2){
  int num3 = num1+num2;
return num3;
}
int main(){
  //Calling the function
  cout<<sum(1,99);
  return 0;
}
```

**The same program can be written like this:** Well, I am writing this program to let you understand an important term regarding functions, which is function declaration. Let's see the program first and then at the end of it we will discuss function declaration, definition and calling of function.

```
#include <iostream>
using namespace std;
//Function declaration
int sum(int,int);
```

```
//Main function
int main(){
  //Calling the function
  cout<<sum(1,99);
  return 0;
}
/* Function is defined after the main method
 */
int sum(int num1, int num2){
  int num3 = num1+num2;
  return num3;
}
```

**Function Declaration:** You have seen that I have written the same program in two ways, in the first program I didn't have any function declaration and in the second program I have function declaration at the beginning of the program. The thing is that when you define the function before the main() function in your program then you don't need to do function declaration but if you are writing your function after the main() function like we did in the second program then you need to declare the function first, else you will get compilation error.

**syntax of function declaration:**

```
return_type function_name(parameter_list);
```

**Note:** While providing parameter list you can avoid the parameter names, just like I did in the above example. I have given `int sum(int,int);` instead of `int sum(int num1,int num2);`.

**Function definition:** Writing the full body of function is known as defining a function.

**syntax of function definition:**

```
return_type function_name(parameter_list) {
   //Statements inside function
}
```

**Calling function:** We can call the function like this:

```
function_name(parameters);
```

Now that we understood the **working of function**, let's see the types of function in C++

**Types of function**

We have two types of function in C++:

1) Built-in functions

2) User-defined functions

1) Build-it functions

Built-in functions are also known as library functions. We need not to declare and define these functions as they are already

written in the C++ libraries such as iostream, cmath etc. We can directly call them when we need.

*Example: C++ built-in function example*

Here we are using built-in function pow(x,y) which is x to the power y. This function is declared in cmath header file so we have included the file in our pr ogram using #include directive.

```
#include <iostream>
#include <cmath>
using namespace std;
int main(){
   cout<<pow(2,5);
   return 0;
}
```

## 2) User-defined functions

We have already seen user-defined functions, the example we have given at the beginning of this tutorial is an example of user-defined function. The functions that we declare and write in our programs are user-defined functions. Lets see another example of user-defined functions.

*User-defined functions*

```
#include <iostream>
#include <cmath>
using namespace std;
//Declaring the function sum
int sum(int,int);
int main(){
  int x, y;
  cout<<"enter first number: ";
  cin>> x;
  cout<<"enter second number: ";
  cin>>y;
  cout<<"Sum of these two :"<<sum(x,y);
  return 0;
}
//Defining the function sum
int sum(int a, int b) {
  int c = a+b;
  return c;
}
```

Default Arguments in C++ Functions

The default arguments are used when you provide no arguments or only few arguments while calling a function. The default arguments are used during compilation of program. For example, let's say you have a user-defined function `sum` declared like this: `int sum(int a=10, int b=20)`, now while calling this function you do not provide any arguments, simply called sum(); then in this case the result would be 30, compiler used the default values 10 and 20 declared in function signature. If you pass only one argument like this: sum(80) then the result would be 100, using the passed argument 80 as first value and 20 taken from the default argument.

**Example: Default arguments in C++**

```
#include <iostream>
using namespace std;
int sum(int a, int b=10, int c=20);

int main(){
  cout<<sum(1)<<endl;
  cout<<sum(1, 2)<<endl;
  cout<<sum(1, 2, 3)<<endl;
  return 0;
}
int sum(int a, int b, int c){
  int z;
  z = a+b+c;
  return z;
}
```

**Rules of default arguments**

As you have seen in the above example that I have assigned the default values for only two arguments b and c during function declaration. It is up to you to assign default values to all arguments or only selected arguments but remember the following rule while assigning default values to only some of the arguments:

**If you assign default value to an argument, the subsequent arguments must have default values assigned to them, else you will get compilation error.**

**For example:** Let's see some valid and invalid cases.

**Valid:** Following function declarations are valid –

```
int sum(int a=10, int b=20, int c=30);
int sum(int a, int b=20, int c=30);
int sum(int a, int b, int c=30);
```

**Invalid:** Following function declarations are invalid –

```
int sum(int a=10, int b, int c=30);
int sum(int a, int b=20, int c);
int sum(int a=10, int b=20, int c);
```

## Array

An array is a collection of similar items stored in contiguous memory locations. In programming, ometimes a simple variable is not enough to hold all the data. For example, let's say we want to store the marks of 500 students, having 500 different variables for this task is not feasible, we can define an array with size 500 that can hold the marks of all students.

**Declaring an array in C++**

There are couple of ways to declare an array.

Method 1:

```
int arr[5]; =>arr[0] = 10;, arr[1] = 20;, arr[2] = 30;, arr[3] = 40;, arr[4] = 50;
```

Method 2:

```
int arr[] = {10, 20, 30, 40, 50};
```

Method 3:

```
int arr[5] = {10, 20, 30, 40, 50};
```

**How to access Array Elements?**

Array index starts with 0, which means the first array element is at index 0, second is at index 1 and so on. We can use this information to display the array elements. See the code below:

```
#include <iostream>
using namespace std;
int main(){
  int arr[] = {11, 22, 33, 44, 55};
  cout<<arr[0]<<endl;
  cout<<arr[1]<<endl;
  cout<<arr[2]<<endl;
  cout<<arr[3]<<endl;
  cout<<arr[4]<<endl;
  return 0;
}
```

Although this code worked fine, displaying all the elements of array like this is not recommended. When you want to access a particular array element then this is fine but if you want to display all the elements then you should use a loop like this:

```
#include <iostream>
using namespace std;
int main(){
  int arr[] = {11, 22, 33, 44, 55};
  int n=0;
  while(n<=4){
    cout<<arr[n]<<endl;
    n++;
  }
  return 0;
}
```

Multidimensional Array

Multidimensional arrays are also known as **array of arrays**. The data in multidimensional array is stored in a tabular form

**A two dimensional array:**

```
int arr[2][3];
```

This array has total 2*3 = 6 elements

**A three dimensional array:**

```
int arr[2][2][2];
```

This array has total 2*2*2 = 8 elements.

**Two dimensional array**

Let's see how to declare, initialize and access Two Dimensional Array elements.

**How to declare a two dimensional array?**

```
int myarray[2][3];
```

**Initialization:**

We can initialize the array in many ways:

**Method 1:**

```
int arr[2][3] = {10, 11 ,12 ,20 ,21 , 22};
```

**Method 2:**

This way of initializing is preferred as you can visualize the rows and columns here.

```
int arr[2][3] = {{10, 11 ,12} , {20 ,21 , 22}};
```

**Accessing array elements:**

arr[0][0] – first element

arr[0][1] – second element

arr[0][2] – third element

arr[1][0] – fourth element

arr[1][1] – fifth element

arr[1][2] – sixth element

Example

```
#include <iostream>
using namespace std;
int main(){
  int arr[2][3] = {{11, 22, 33}, {44, 55, 66}};
  for(int i=0; i<2;i++){
    for(int j=0; j<3; j++){
      cout<<"arr["<<i<<"]["<<j<<"]: "<<arr[i][j];
    }
  }
  return 0;
}
```

**Output:**

arr[0][0]: 11,arr[0][1]: 22,arr[0][2]: 33,arr[1][0]: 44arr[1][1]: 55
arr[1][2]: 66

## Three dimensional array

Let's see how to declare, initialize and access Three Dimensional Array elements.

**Declaring a three dimensional array:**

```
int myarray[2][3][2];
```

**Initialization:**

We can initialize the array in many ways:

**Method 1:**

```
int arr[2][3][2] = {1, -1 ,2 ,-2 , 3 , -3, 4, -4, 5, -5, 6, -6};
```

**Method 2:**

This way of initializing is preferred as you can visualize the rows and columns here.

```
int arr[2][3][2] = {
    {{1,-1}, {2, -2}, {3, -3}},
    {{4, -4}, {5, -5}, {6, -6}}
}
```

**Three dimensional array example**

```
#include <iostream>
using namespace std;
int main(){
  // initializing the array
  int arr[2][3][2] = {
    { {1,-1}, {2,-2}, {3,-3} },
    { {4,-4}, {5,-5}, {6,-6} }
  };  // displaying array values
  for (int x = 0; x < 2; x++) {
    for (int y = 0; y < 3; y++) {
      for (int z = 0; z < 2; z++) {
        cout<<arr[x][y][z]<<" "; } }}
  return 0;}
```

# Strings

Strings are words that are made up of characters, hence they are known as sequence of characters. We have two ways to create and use strings:

1) By creating char arrays and treat them as string

2) By creating string object

**Array of Characters – Also known as C Strings**

**Example 1:**

A simple example where we have initialized the char array during declaration.

```
#include <iostream>
using namespace std;
int main(){
  char book[50] = "A Song of Ice and Fire";
  cout<<book;
  return 0;
}
```

**Output:**

A Song of Ice and Fire

**Example 2: Getting user input as string**

we can be considered as inefficient method of reading user input, why? Because when we read the user input string using cin then only the first word of the string is stored in char array and rest get ignored. The cin function considers the space in the string as delimiter and ignore the part after it.

```
#include <iostream>
using namespace std;
int main(){
  char book[50];
  cout<<"Enter your favorite book name:";
  //reading user input
  cin>>book;
  cout<<"You entered: "<<book;
  return 0;
}
```

You can see that only the “The” got captured in the book and remaining part after space got ignored. How to deal with this then? Well, for this we can use cin.get function, which reads the complete line entered by user.

**Example 3: Correct way of capturing user input string using cin.get**

```
#include <iostream>
using namespace std;
int main(){
```

```
  char book[50];
  cout<<"Enter your favorite book name:";

  //reading user input
  cin.get(book, 50);
  cout<<"You entered: "<<book;
  return 0;
}
```

## Structures

Structure is a compound data type that contains different variables of different types. For example, you want to store Student details like student name, student roll num, student age. You have two ways to do it, one way is to create different variables for each data, but the downfall of this approach is that if you want to store the details of multiple students, and in that case it is not feasible to create separate set of variables for each student.

The second and best way of doing it by creating a structure like this:

```
struct Student
{    char stuName[30];
   int stuRollNo;
   int stuAge;
};
```

Now these three members combined will act like a separate variable and you can create structure variable like this:

structure_name variable_name

So if you want to hold the information of two students using this structure then you can do it like this:

```
Student s1, s2;
```

Then you can access the members of Student structure like this:

```
//Assigning name to first student
s1.stuName = "Ajeet";
//Assigning age to the second student
s2.stuAddr = 22;
```

Similarly I can set and get the values of other data members of the structure for every student. Let's see a complete example to put this up all together:

**Structure**

```
#include <iostream>
using namespace std;
struct Student{
  char stuName[30];
```

```
  int stuRollNo;
  int stuAge;
};
int main(){
  Student s;
  cout<<"Enter Student Name: ";
  cin.getline(s.stuName, 30);
  cout<<"ENter Student Roll No: ";
  cin>>s.stuRollNo;
  cout<<"Enter Student Age: ";
  cin>>s.stuAge;
  cout<<"Student Record:"<<endl;
  cout<<"Name: "<<s.stuName<<endl;
  cout<<"Roll No: "<<s.stuRollNo<<endl;
  cout<<"Age: "<<s.stuAge;
  return 0;
}
```

## Pointers

Pointer is a variable in C++ that holds the address of another variable. They have data type just like variables, for example an integer type pointer can hold the address of an integer variable and a character type pointer can hold the address of char variable.

*Syntax of pointer*

data_type *pointer_name;

**How to declare a pointer?**

his pointer p can hold the address of an integer

variable, here p is a pointer and var is just a

simple integer variable

int *p, var

**Assignment**

As I mentioned above, an integer type pointer can hold the address of another int variable. Here we have an integer variable var and pointer p holds the address of var. To assign the address of variable to pointer we use **ampersand symbol** (&).

p = &var;

**How to use it?**

cout<<&var;

cout<<p;

cout<<*p;

**Example of Pointer**

Let’s take a simple example to understand what we discussed above.

```
#include <iostream>
using namespace std;
int main(){
  //Pointer declaration
  int *p, var=101;
  //Assignment
  p = &var;
  cout<<"Address of var: "<<&var<<endl;
  cout<<"Address of var: "<<p<<endl;
  cout<<"Address of p: "<<&p<<endl;
  cout<<"Value of var: "<<*p;
  return 0;
}
```

**Pointer and arrays**

While handling arrays with pointers you need to take care few things. First and very important point to note regarding arrays is that the array name alone represents the base address of array so while assigning the address of array to pointer don’t use ampersand sign(&). Do it like this:

**Correct:** Because arr represent the address of array.

```
p = arr;
```

**Incorrect:**

```
p = &arr;
```

Example: Traversing the array using Pointers

```
#include <iostream>
using namespace std;
int main(){
  //Pointer declaration
  int *p;
  //Array declaration
  int arr[]={1, 2, 3, 4, 5, 6};
  //Assignment
  p = arr;
  for(int i=0; i<6;i++){
    cout<<*p<<endl;
    //++ moves the pointer to next int position
    p++;   }
  return 0;}
```