Chapter 2 : Introduction to C++

Data Types, Tokens & Keywords, Identified & Constants

Data Types : All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store.

Primitive Data Types : These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool etc.

Primitive data types available in C++ are :

1. Integer
2. Character
3. Boolean
4. Floating Point
5. Double Floating Point
6. Valueless or Void
7. Wide Character

• **Derived Data Types** : The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

1. Function
2. Array
3. Pointer
4. Reference

• **Abstract or User-Defined Data Types** : These data types are defined by user itself. Like, defining a class in C++ or a structure.

C++ provides the following user-defined datatypes:

1. Class
2. Structure
3. Union
4. Enumeration
5. Typedef defined Datatype

• **Tokens** : A token is the smallest element of a program that is meaningful to the compiler.

Tokens can be classified as follows :

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

- **Keywords** : Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords.
- **Identifiers** : Identifiers are used as the general terminology for the naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords.

- **Constants** : Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals.
- **Strings** : Strings are nothing but an array of characters ended with a null character (‘\0’). This null character indicates the end of the string. Strings are always enclosed in double-quotes.
- **Special Symbols** : The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () {}, ; * = #
- **Operators** : Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act upon are called operands.

Introduction to Reference Variables

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting '&' in the declaration.

```
#include<iostream>
using namespace std;
int main()
{
  int x = 10;
  // ref is a reference to x.
  int& ref = x;
  // Value of x is now changed to 20
```

```
  ref = 20;
cout << "x = " << x << endl ;
  // Value of x is now changed to 30
  x = 30;
  cout << "ref = " << ref << endl ;
  return 0;
}
```

Introduction to Classes and Objects

The classes are the most important feature of C++ that leads to Object Oriented Programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.

For example : Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all posses this behavior and characteristics.

Similarly, class is just a blue print, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior.

Classes and Objects Specifiers

In C++, accessing the data members and functions in the class depends on the access given to that particular data member or function using an access specifier.

C++ supports the following access specifiers :

1. **Private**

This is the default access specifier for a class in C++. This means that if no access specifier is specified for the members in a class, then it is considered private.

When a member is private, it cannot be accessed outside the class. Not even using the object and the dot operator. The private data members can only be accessed using the member functions of the class.

2. **Public**

A data member or function that is defined as public in the class is accessible to everyone outside the class. These members can be accessed using the object and the dot operator.

3. **Protected**

A protected member of a class is accessible to the class itself and the child classes of that class.

This access specifier is especially used in case of inheritance and we will discuss this in detail while discussing the inheritance topic.

Let us take the following example to better understand these access specifiers.

```
#include <iostream>
#include <string>
using namespace std;
class ABC{
int var1 = 10;
public:
string name;
void display()
   {
      cout<<"var1 ="<<var1<<endl;
      cout<<"name ="<<name<<endl;
```

```
  }
};
int main();
   {
      ABC abc;
      //abc.var1 = 20;
      abc.name = "sth";
      abc.display();
}
```

Output :

```
var1 =10
name =sth
```

In this program, we have two data members out of which var1 of type int is private (access specifier not specified. Default is private). Another member is the string name, which is declared as public. We have yet another function display which displays the value of both these members.

In the main function, we declare an object abc of class ABC. Then we set values to data members and also the call function display using object ‘abc’.

However, when the compiler encounters the line abc.var1 = 20; it will generate an error that “var1 is private variable”.

This is because we cannot access private data members of a class outside the class. Thus there is an error. But we can access it inside the function and therefore when we output the value of var1 in the display function; it does not throw any error.

Hence the output of the program displays the initial value with which var1 is declared.

So far, we have seen the details about classes, object, and access specifiers, now let us take up a complete example of a sample class student. This class has data members: student_id, student_name, and student_age. It also has member functions to read student info and display student info.

In order to make things easy for the readers, we have declared all members of the class as public.

Defining data members and member functions

Variables declared within a class preceded by a data type which define the state of an object are called data members, and functions which define the behaviour of an object of that class are called member functions.

Eg-

```
class human{
private: string name; // data member
int age; // data member
public: void run(){ }; // member function
void eat(){ }; // member function
};
```

Instantiating and Using Classes

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the object constructor.

Note : The phrase "instantiating a class" means the same thing as "creating an object." When you create an object, you are creating an "instance" of a class, therefore "instantiating" a class.

The new operator requires a single, postfix argument: a call to a constructor. The name of the constructor provides the name of the class to instantiate.The new operator returns a reference to the object it created. This reference is usually assigned to a variable of the appropriate type,

like :

Point originOne = new Point(23, 94);

The reference returned by the new operator does not have to be assigned to a variable. It can also be used directly in an expression.

For example :

int height = new Rectangle().height;

Array of Objects

- Like array of other user-defined data types, an array of type class can also be created.
- The array of type class contains the objects of the class as its individual elements.
- Thus, an array of a class type is also known as an array of objects.
- An array of objects is declared in the same way as an array of any built-in data type.

Syntax :

class_name array_name [size] ;

Example :

```
#include <iostream>
class MyClass {
  int x;
public:
  void setX(int i) { x = i; }
  int getX() { return x; }
};
void main()
{
  MyClass obs[4];
```

```
 int i;
  for(i=0; i < 4; i++)
    obs[i].setX(i);
  for(i=0; i < 4; i++)
    cout << "obs[" << i << "].getX(): " << obs[i].getX() << "\n";
  getch();
}
```

Output :

```
obs[0].getX(): 0
obs[1].getX(): 1
obs[2].getX(): 2
obs[3].getX(): 3
```

Managing console I/O

Every program takes some data as input and generates processed data as output from the familiar input process output cycle. Like this way c++ programming language provide this type facility. We have already use these operator these are the cin (<<) and cout (<<)operator.

C++ support rich set of input and output operators and function since these function use the advanced features of c++ as like classes derived classes and virtual function.

C++ uses the concept of stream and stream classes to implementation i/o operation with the console and disk files.

C++ stream classes

What is Stream?

- A stream is an abstraction. It is a sequence of bytes.
- It represents a device on which input and output operations are performed.
- It can be represented as a source or destination of characters of indefinite length.
- It is generally associated to a physical source or destination of characters like a disk file, keyboard or console.
- C++ provides standard iostream library to operate with streams.
- The iostream is an object-oriented library which provides Input/Output functionality using streams.

- The fstream.h header file contains a declaration of ifstream, ofstream and fstream classes.
- The iostream.h file contains istream, ostream and iostream classes and included in the program while doing disk I/O operations.
- The filebuf class contains input and output operations with files. The streambuf class does not organize streams for input and output operations, only derived classes of streambuf performs I/O operations.
- These derived classes arranges a space for keeping input data and for sending output data.
- The istream and ostream invokes the filebuf functions to perform the insertion or extraction on the streams.