Day 2 C++

- C++ gives flexibility to write a program with or without a class and its member functions definitions.
- A simple C++ program will include comments, namespace, headers, main(), and input/output statements

Comments

- It is useful for documentation
- C++ structure supports two comment styles
    - Single line comment

/ / An example to show single line comment

It can also be written as

/ / An example to demonstrate

/ / single line comment

    - Multiline comment

/* An example to demonstrate multiline comment */

for(int i = 0; i<10; //loop runs 10 times i++) --- This is wrong... i++ also gets commented

for(int i = 0; i<10; /*loop runs 10 times */ i++)

Headers

- Usually, a program includes different programming elements such as built-in functions, classes, keywords, constants, operators and more which are already defined in the standard C++ library
- For using such pre-defined elements in a program, an appropriate header must be included in the program. Moreover, the standard headers comprise information such as prototype, definition and return type of library functions, the data type of constants and more.
- Consequently, programmers do not have to declare (or define) the predefined programming elements explicitly. Further, standard headers specify in the program through the preprocessor directive” #include.

Namespace

- Namespace allows the grouping of different entities such as classes, objects, functions and a variety of C++ tokens, etc., under a single name.
- The C++ Standards Committee has rearranged the entities of the standard library under a namespace known as std.
    - **using namespace std;**

      cout<<“Hello World”;
    - **std: :cout<<“Hello World”;**
- The using statement informs the compiler that you want to use the std namespace
- The purpose of namespace is to avoid name collision it is done by localising the names of identifiers
- It creates a declarative region and defines the scope
- Anything defined within a namespace is in the scope of that namespace
- Here std is the namespace in which entire standard C++ library is declared

Main Function

- The main () is a startup function that starts the execution of a c++ program. All C++ statements which are to be executed are written within main ( ). The compiler executes all the instructions written within the opening and closing curly braces’ {}’ that enclose the body of the main ( ).
- As soon as all the instructions in main () execute, the control passes out of main ( ), and terminates the whole program and return a value to the operating system.
- By default, main () in C++ returns an int value to the operating system. Thus, main () must end with the return 0 statement. A return value zero denotes success and a non-zero value denotes failure or error.
- Every program should have one and only one main function, otherwise the compiler will not be able to locate the begining of the program

Tokens in C++

- Token is a generic word for keywords, Data types, Variables, Constants and Identifiers

  int main() {

```
int a =2;
double const b = 4;
float c = 1.5;
char d = 'A';
```

…..

Keywords, Constants, Datatype

- Keywords have fixed meaning that cannot be changed.
- Keywords cannot be used as variable names
- There are 32 keywords in C eg. Auto, break, case, char, enum, extern, etc
- There are additional 30 reserved words in C++

  new, catch, namespace, bool, class, friend etc
- Constants are fixed values
- They do not change during the execution of program
- There are two types of constants
  - Numeric constants
  - Character constants
- Constant can be defined by the use of keyword 'const'
- Another way is by using '#define' preprocessor directive

Data Type

- Data type is a finite set of values along with a set of rules

  in the above eg. int, double, float, char are data types
- a, c, d are variables
- Variable is a data name
- It may be used to store a data value
- The values can change when a program runs
- Before using a variable it must be declared
- We should try to give meaningful names to variables

Identifiers

- Identifiers are user defined names

- An identifier consists of letters, digits and underscore
- Both uppercase and lowercase letters are permitted
- First character must be an alphabet or underscore
- Identifiers are case sensitive
- Variable name, function name, structure name etc are known as identifiers

Operators

- Operator is a symbol that we use for performing mathematical or logical manipulations
    - Arithmetic operators
    - Increment and Decrement operators
    - Relational operators
    - Logical operators
    - Bitwise operators
    - Assignment operators
    - Misc. operators

Typecasting

- Typecasting is used to make one type variable to act like another type
- Typecasting is done by enclosing the data type you want within parenthesis
- (float) -- this is the cast that needs to be put in front of he variable you want to cast
- (float)varname
- This cast is valid for one single operation

Structure of object oriented program

```
#include <iostream>
using namespace std;
class square
{
   int x;   // data member
   public:
   int area(int);  //member function
};
```

```
int square :: area(int a){      //:: is a scope resolution operator. It specifies that area is not global function but it is member function of class
   x = a;
   return x*x;
}
main(){
   square sqr;     // sqr is the object of class square
   cout << "Area of square is " << sqr.area(4) << endl;     //here we are calling function area using the object sqr and dot operator
   return 0;
}
```

Class

- Class is a user-defined data type
- It is a template of an object..
    - Eg. Animal is a class and dog is the object of class animal
- Class is created using a keyword class
- It holds data and functions
- Class links the code and data
- The data and the functions are called as the members of the class

Object

- Objects are variables
- They are the copy (instances) of a class
- Each of them has property and behaviour
- Properties are defined through data elements
- Behaviour is defined through member functions called methods
- The class itself is just a template that is not allocated any memory

When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

- To use the data and methods defined in the class, we have to create an object of that class

Class_name object_name;

- We use (.) dot operator for accessing data and methods of an object

Eg. Obj.number()

- The public data members are also accessed in the same way, however the private data members are not allowed to be accessed directly by the object.

Member Functions in classes

- There are 2 ways to define a member function:
    - Inside class definition
    - Outside class definition
    - To define a member function outside the class definition we have to use the **scope resolution:: operator** along with the class name and function name.

Constructors

- Constructors are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition. There are 3 types of constructors:
    - Default Constructor
    - Parameterized Constructor
    - Copy Constructor – it creates a new object, which is an exact copy of the existing object. The compiler provides a default Copy Constructor to all the classes.
        - Class-name (class-name &){}

Destructors

- Destructor is another special member function that is called by the compiler when the scope of the object ends.

Syntax of a class

Class class-name

{

public/private/protected:

Data members

Member functions

};

Access specifier

- Public Specifier
    - The public specifier allows the data to be accessed outside the class
    - A public member can be used anywhere in the program
- Private Specifier
    - The members declared private cannot be accessed outside the class
    - Private members can be used only by the members of the class
- Protected Specifier
    - Protected members cannot be accessed from outside the class
    - They can be accessed by a derived class

Scope Resolution Operator

- It is used to access hidden data
- To access the variable or function with the same name we use :: operator
- Suppose the local variable and global variable has same name, the local variable gets the priority
- We can access the global variable using :: operator

Encapsulation, Data Abstraction

- So far we have seen data and function combined together in a class
- Class is a single unit in which the data and function using them are grouped, this mechanism is called as **encapsulation**
- We have also seen that the data of the class is a private member, which cannot be accessed outside the class.
- The private data is hidden and cannot be accessed outside the class, this mechanism is called **data abstraction**
- The interface is seen but the implementation is hidden

Static Members

- Static variables are initialized to zero before the first object is created
- Only one copy of the static variable exists for the whole program
- All the objects will share that variable

- It will remain in the memory till the end of the program
- A Static Function may be called by itself without depending on any object
- To access the static function we use,

Classname :: staticfunction();

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

class stat1 {
   int x;  // private member
public:
  static int sum;

  stat1(){    //Constructor
    x = sum++;
  }
  static void statdis(){
    cout << "Result is : " << sum << "\n";
  }
  void number(){
    cout << "Number is : " <<  x << "\n";
  }
};
int stat1 :: sum;  //to declare the static variable globally we use :: //now the storage is allocated to the variable sum and is initialized to zero
int main(){
   stat1 obj1, obj2, obj3;
   obj1.number();
   obj2.number();
   obj3.number();
   stat1::statdis();
   cout << " Now static var sum is" << obj1.sum << "\n";
   return 0;
}
```

Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when entered into the block or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access this variable only within that block.
- Initialization of Local Variable is Mandatory.

Instance Variables

Instance variables are non-static variables and are declared in a class outside any method, constructor, or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
- Initialization of Instance Variable is not Mandatory.
- Instance Variable can be accessed only by creating objects.

Static Variables

- Static variables are also known as Class variables.
  These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of Static Variable is not Mandatory. Its default value is 0
- If we access the static variable like the Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name with the class name automatically.
- If we access the static variable without the class name, the Compiler will automatically append the class name.