COURSE CODE & NAME DCA1203 - Object Oriented Programming – C++

1.)

a.) **Question :- What is identifier and how they are created in C++ and specify their naming Conventions.**

Answer :- C++ Identifiers. C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers. In short, we can say that the C++ identifiers represent ...

⇨ In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

- **Constants**
- **Variables**
- **Functions**
- **Labels**
- **Defined data types**

Some naming rules are common in both C and C++. They are as follows:

- Only alphabetic characters, digits, and underscores are allowed.
- The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. After the first letter, we can use letters, digits, or underscores.
- In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
- A declared keyword cannot be used as a variable name.

For example, suppose we have two identifiers, named as 'FirstName', and 'Firstname'. Both the identifiers will be different as the letter 'N' in the first case in uppercase while lowercase in second. Therefore, it proves that identifiers are case-sensitive.

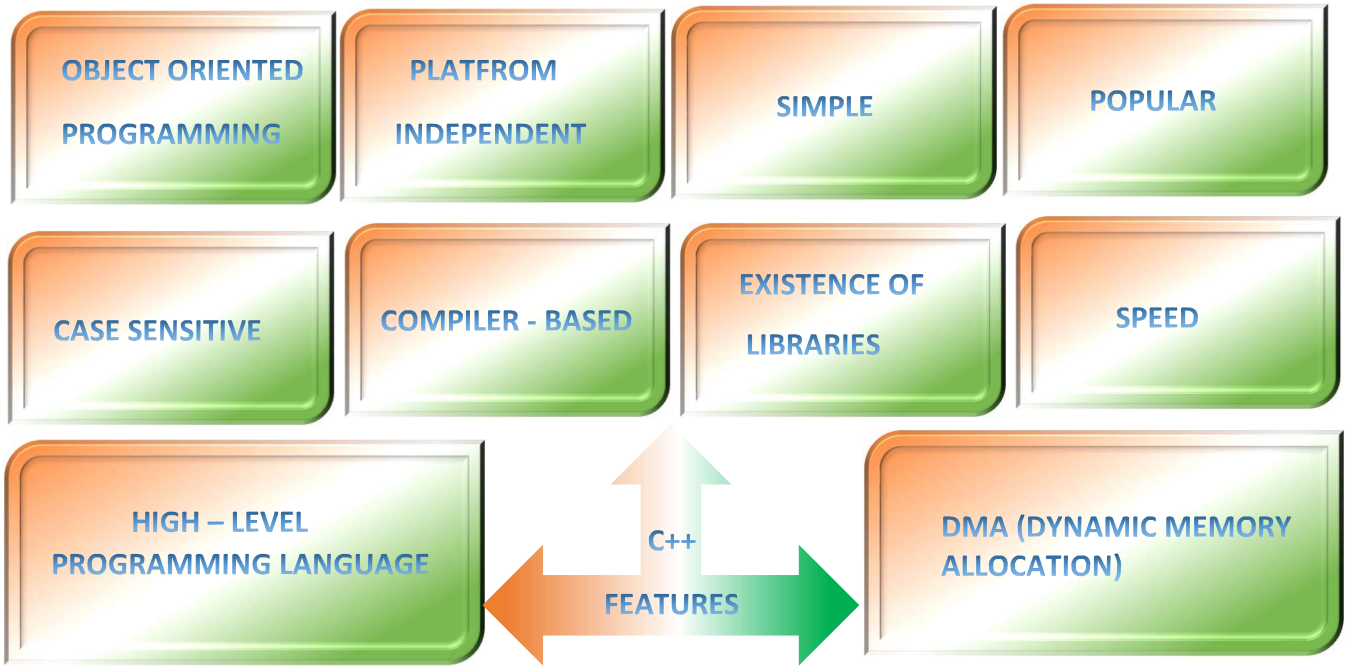
1.)

b.) **Question :- Describe the object-oriented programming features of C++?**

Answer :- C++ is quite similar to C programming. In fact, *C++ supports all the features offered with C* with the addition to various other important features like object-oriented programming, operator overloading, exception and error handling, the namespace feature, and many more. We can say C++ is the advanced version of C programming. Features of C++ give multiple reasons to upgrade our skills from C to C++.

Features of C++

Here are some of the remarkable features of C++ language:



1. OOP (Object-Oriented Programming)

C++ is an object-oriented language, unlike C which is a procedural language. This is one of the most important features of C++. It employs the use of objects while programming. These objects help you implement real-time problems based on data abstraction, data encapsulation, data hiding, and polymorphism. We have briefly discussed all the 5 main concepts of object-oriented programming.

2. Platform or Machine Independent/ Portable

Although C++ is not platform-independent as compiled programs on one operating system won't run on another operating system , But in another term, portability refers to using the same piece of code in varied environments.Let us understand this C++ feature with the help of an example. Suppose you write a piece of code to find the name, age, and salary of an

employee in Microsoft Windows and for some apparent reason you want to switch your operating system to LINUX. This code will work in a similar fashion as it did in Windows.

3. Simple

When we start off with a new language, we expect to understand in-depth. The simple context of C++ gives an appeal to programmers, who are eager to learn a new programming language.
If you are already familiar with C, then you don't need to worry about facing any trouble while working in C++. The syntax of C++ is almost similar to that of C. After all C++ is referred to as "C with classes".

4. Popular

After learning C, it is the base language for many other popular programming languages which supports the feature of object-oriented programming. Bjarne Stroustrup found Similar 67, the first object-oriented language ever, lacking simulations and decided to develop C++.

5. Case sensitive

Just like C, it is pretty clear that the C++ programming language treats the uppercase and lowercase characters in a different manner. For instance, the meaning of the keyword **'cout'** changes if we write it as **'Cout'** or **"COUT"**. Other programming languages like HTML and MySQL are not case sensitive.

6. Compiler-Based

Unlike Java and Python that are interpreter-based, C++ is a compiler-based language and hence it a relatively much faster than Python and Java.

7. Existence of Libraries

The C++ programming language offers a library full of in-built functions that make things easy for the programmer. These functions can be accessed by including suitable header files.

8. Speed

As discussed earlier, C++ is compiler-based hence it is much faster than other programming languages like Python and Java that are interpreter-based.

9. High-level programming language

It is important to note that C++ is a high-level programming language, unlike C which is a mid-level programming language. It makes it easier for the user to work in C++ as a high-level language as we can closely associate it with the human-comprehensible language, that is, English.

DMA (Dynamic Memory Allocation)

Since C++ supports the use of pointers, it allows us to allocate memory dynamically. We may even use constructors and destructors while working with classes and objects in C++.

2.)

Question :- What is Operator overloading? Write a C++ program illustrating overloading ++ operator?

Answer :- An operator is a symbol that indicates to the compiler to perform a particular operation. For example, there are various types of operators in C++, such as Arithmetic Operators, Logical Operators, Relational Operators, Assignment Operators, Bitwise Operators, and more. The C++ language allows programmers to give special meanings to operators. This means that you can redefine the operator for user-defined data types in C++. Binary Operator Overloading Example Program

```
#include<iostream.h>

#include<conio.h>

class complex {

   int a, b;

public:

   void getvalue() {

      cout << "Enter the value of Complex Numbers a,b:";

      cin >> a>>b;

   }

   complex operator+(complex ob) {

      complex t;

      t.a = a + ob.a;

      t.b = b + ob.b;

      return (t);

   }
```

```
    complex operator-(complex ob) {
        complex t;
        t.a = a - ob.a;
        t.b = b - ob.b;
        return (t);
    }
    void display() {
        cout << a << "+" << b << "i" << "\n";
    }
};
void main() {
    clrscr();
    complex obj1, obj2, result, result1;
    obj1.getvalue();
    obj2.getvalue();
    result = obj1 + obj2;
    result1 = obj1 - obj2;
    cout << "Input Values:\n";
    obj1.display();
    obj2.display();
    cout << "Result:";
    result.display();
    result1.display();
    getch();
```

}

2.)

A.) Question :- Illustrate types of Inheritance?

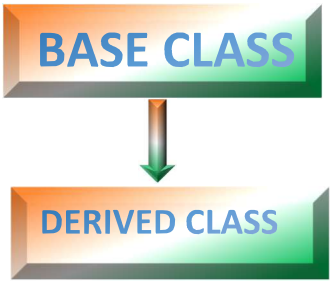
Answer :- Different Types of Inheritance

Inheritance is the process of creating a new Class, called the **Derived Class** , from the existing class, called the **Base Class** . The Inheritance has many advantages, the most important of them being the reusability of code. Rather than developing **new Objects** from scratch, new code can be based on the work of other developers, adding only the new features that are needed. The reuse of **existing classes** saves time and effort.

However, inheritance may be implemented in different combinations in **Object-Oriented Programming languages** as illustrated in figure and they include:

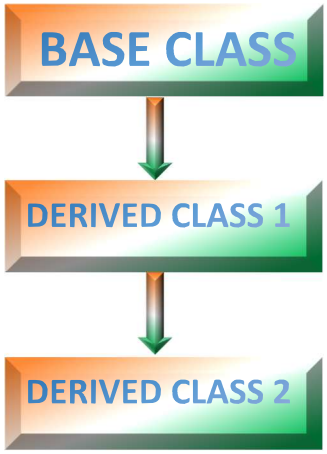
- Single Inheritance
- Multi Level Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance
- Multipath inheritance
- Multiple Inheritance

➔**Single Inheritance**



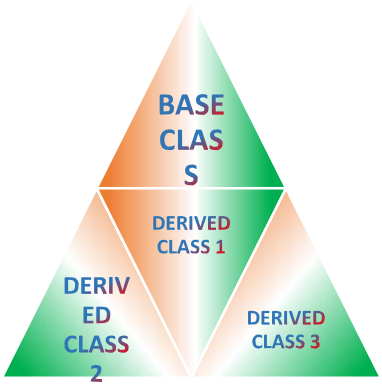
When a **Derived Class** to inherit properties and behaviour from a single **Base Class** , it is called as single inheritance.

➔**Multi Level Inheritance**



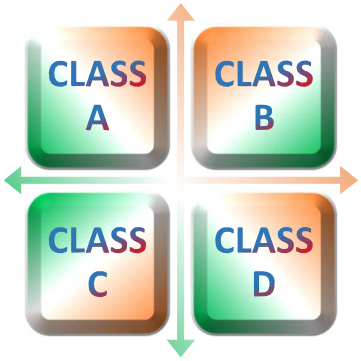
A **derived class** is created from another derived class is called **Multi Level Inheritance** .

⇨ **Hierarchical Inheritance**



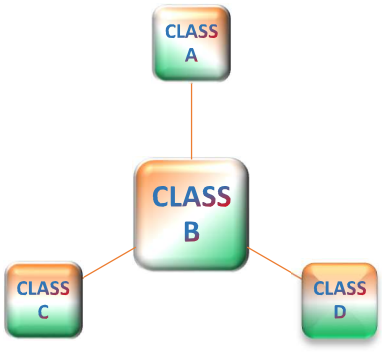
More than one **derived classes** are created from a single base class, is called **Hierarchical Inheritance** .

➔**Hybrid Inheritance**



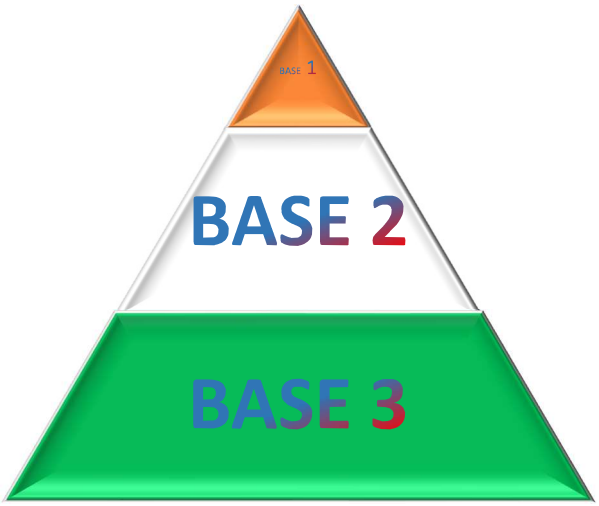
Any combination of above **three inheritance** (single, hierarchical and multi level) is called as **hybrid inheritance** .

➔**Multipath inheritance**



Multiple inheritance is a method of inheritance in which one **derived class** can inherit properties of base class in different paths. This inheritance is not supported in .NET Languages such as C#.

➔**Multiple Inheritance**



Multiple inheritances allows programmers to create classes that combine aspects of multiple classes and their corresponding hierarchies. In .Net Framework, the classes are only allowed to inherit from a single parent class, which is called single inheritance.

2.)

B.) Question :- What is a constructor? Write the syntax of declaring the constructor?

Answer :- Object-oriented programming works by structuring programs as objects. Objects are entities that represent something from real life, such as a dog or a person. Much like dogs or people, the objects representing them have their defining attributes and behaviours.
Below, we'll focus on the constructor, a type of method specific to the OOP paradigm. Although we make use of C++ programming to illustrate constructors, the principles underlying our examples are common to most OOP languages.

C++ Constructors

Constructors are methods that are automatically executed every time you create an object. The purpose of a constructor is to construct an object and assign values to the object's members. A constructor takes the same name as the class to which it belongs, and does not return any values. Let's take a look at a C++ constructor example:

```
#include <iostream>
#include <string>
class Dog {
  public:
    std::string name;
    bool drools;
    void barks() {
    std::cout << "Woof Woof\n";
      };
```

```
    Dog(std::string dog_name, bool dog_drools){ // this is the constructor
        name = dog_name;
        drools = dog_drools;
    }
};
```

The above is an example of a **parameterized** constructor method. As their name suggests, parameterized constructors accept parameters that they use to initialize an object's member variables. In our example, the constructor receives std::string dog_name and bool dog_drools. The values of these parameters are used to initialize the class attributes name and drools.

Let's see how this works in a concrete example:

```
int main(int argc, const char * argv[]) {
    Dog spot("Spot", true); // we're calling the constructor with 2 parameters
    std::cout << "Name: " << spot.name << std::endl;
    return 0;
}
```

Running this program produces the following output:

```
Name: Spot
```

In the program above, we create an object spot of the class Dog. In the line where we create the object, the constructor accepts the values "Spot" and "true" and assigns them to spot.name and spot. Drools.

The way our constructor above is defined requires us to always provide exactly two arguments when we create an object. In case we do not provide any arguments to the constructor, the compiler will print an error because the class does not have instructions for constructing unparameterized objects. However, there's an easy fix for this.

Default C++ Constructor

CPP constructors can also be **unparameterized**; that is, they can create objects without being given arguments. Such constructors are called default constructors. The "default" in "default constructor" refers to the fact that the constructor assigns default arguments

Default C++ Constructor with Default Arguments

Like all functions and methods, a constructor can also have default arguments. These are the values that the constructor uses to initialize member values if the user does not provide any custom arguments.

The following class definition extends our existing constructor to accept default arguments:

```
class Dog {
    public:
        std::string name;
        bool drools;
        void barks() {
```

```
        std::cout << "Woof Woof\n";
            };
    Dog(std::string dog_name = "No name", bool dog_drools = false){
        name = dog_name;
        drools = dog_drools;
    }
};
```

Note the changes in the highlighted line: dog_name and dog_drools now take up “No name” and “false” as their default values. Now we can still provide custom arguments, but also create an object without them. Let’s make sure that this works as intended by creating another Dog object, this time with an empty constructor:

```
int main(int argc, const char * argv[]) {
    Dog spot("Spot", true);
    Dog lulu;    // we’re calling the default constructor here
    std::cout << "Name: " << spot.name << std::endl;
    std::cout << "Name: " << lulu.name << std::endl;
    return 0;
}
```

Output:

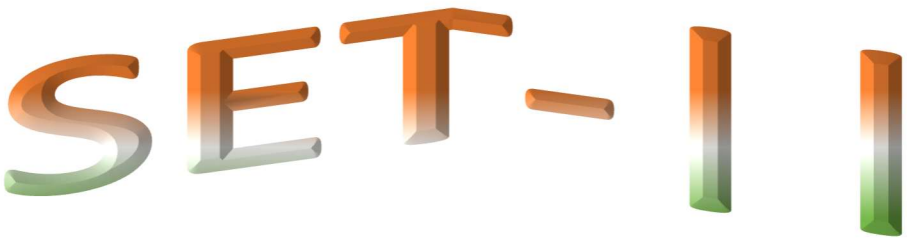
Name: Spot

Name: No name

We created the object lulu without passing any arguments to the class’s constructor. Printing lulu.name shows that the constructor assigned it the default value of “No name.”

Default C++ Constructor with No Parameters

The code above shows an example of a default C++ constructor with default parameter values. There’s also a constructor that does not have any parameters at all. This is the constructor that your compiler automatically creates when the user does not provide one to the class. Here’s how it looks:



4.)

A.) Question :- Explain the exception handling mechanism in C++?

Answer :- Exceptions allow a method to react to exceptional circumstances and errors (like runtime errors) within programs by transferring control to special functions called handlers.

For catching exceptions, a portion of code is placed under exception inspection. Exception handling was not a part of the original C++. It is a new feature that ANSI C++ included in it. Now almost all C++ compilers support this feature. Exception handling technology offers a securely integrated approach to avoid the unusual predictable problems that arise while executing a program .

There are two types of exceptions:

1. Synchronous exceptions
2. Asynchronous exceptions

Errors such as: out of range index and overflow fall under the category of synchronous type exceptions. Those errors that are caused by events beyond the control of the program are called asynchronous exceptions. The main motive of the exceptional handling concept is to provide a means to detect and report an exception so that appropriate action can be taken. This mechanism needs a separate error handling code that performs the following tasks:

- Find and hit the problem (exception)
- Inform that the error has occurred (throw exception)
- Receive the error information (Catch the exception)
- Take corrective actions (handle exception)

⇨ EXAMPLE

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

int main()
{
   try {
      throw 6;
   }

   catch (int a) {
      cout << "An exception occurred!" << endl;
      cout << "Exception number is: " << a << endl;
   }

  }
```

The following example shows handling of division by zero exception.

```
#include<iostream>

using namespace std;
```

```
double division(int var1, int var2)
{
   if (var2 == 0) {
      throw "Division by Zero.";
   }
   return (var1 / var2);
}

int main()
{
   int a = 30;
   int b = 0;
   double d = 0;

   try {
      d = division(a, b);
      cout << d << endl;
   }
   catch (const char* error) {
      cout << error << endl;
   }

   return 0;
}
```

Advantage of exception handling

1. Programmers can deal with them at some level within the program
2. If an error can't be dealt with at one level, then it will automatically be shown at the next level, where it can be dealt with.

4.)

B.)Question :- Explain about file manipulators?

Answer :-

- Manipulators are operators used in C++ for **formatting output**. The data is manipulated by the programmer's choice of display.
- In this C++ tutorial, you will learn what a manipulator is, **endl** manipulator, **setw** manipulator, **setfill** manipulator and setprecision manipulator are all explained along with syntax and examples.

endl Manipulator:

- This manipulator has the same functionality as the 'n' newline character.

For example:

1. cout << Exforsys" <<endl;

2. cout <<" Training";

etw Manipulator:

- This manipulator sets the minimum field width on output.

  Syntax :

  Setw(x)

- Here setw causes the number or string that follows it to be printed within a field of x characters wide and x is the argument set in setw manipulator.
- The header file that must be included while using setw manipulator is .

Example:

```
#include <iostream>

#include <iomanip>


void main( )

{

int x1=123,x2= 234, x3=789;

cout << setw(8) << "Exforsys" << setw(20) << "Values" << endl

<< setw(8) << "test123" << setw(20)<< x1 << endl

<< setw(8) << "exam234" << setw(20)<< x2 << endl

<< setw(8) << "result789" << setw(20)<< x3 << endl;

}
```

setfill Manipulator:

- This is used after setw manipulator.
- If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields.

```
#include <iostream>

#include <iomanip>

void main()
```

```
{
cout << setw(15) << setfill('*') << 99 << 97 << endl;
}
```

➔setprecision Manipulator:

- The setprecision Manipulator is used with floating point numbers.
- It is used to set the number of digits printed to the right of the decimal point.
- This may be used in two forms:

1. fixed
2. scientific

- These two forms are used when the keywords fixed or scientific are appropriately used before the setprecision manipulator.
- The keyword fixed before the setprecision manipulator prints the floating point number in fixed notation.
- The keyword scientific, before the setprecision manipulator, prints the floating point number in scientific notation.

Example:

```
#include <iostream>
#include <iomanip>
void main( )
{
float x = 0.1;
cout << fixed << setprecision(3) << x << endl;
cout << scientific << x << endl;
}
```

4.)

Question :- What is Template? What is the need of Template? Declare a Template class?

Answer :-

⇨ A design template or template is a file that is created with an overall layout to be used with one or more documents. For example, a program may have a template for a resume. With a resume template, the overall layout is

designed with placeholder text (e.g., your objective, previous job experience, etc.)
A program may come with pre-designed templates with the ability for a template to be created by the user. When creating a custom template to be saved and reused or share it may contain theme fonts, layouts, theme colours , theme effects, background styles and even content.

- Need of template :- A needs assessment is the systematic process of identifying and evaluating organizational needs and gaps. It is conducted across a range of settings, such as communities, schools, hospitals, states, and business organizations. The assessment gives the organization a snapshot of where they may be lacking and help them refocus their resources to progress toward goals with greater efficiency.

Template class :- The basic syntax for declaring a templated class is as follows:

```
1    template <class a_type> class a_class {...};
```

The keyword 'class' above simply means that the identifier a_type will stand for a datatype. NB: a_type is not a keyword; it is an identifier that during the execution of the program will represent a single datatype. For example, you could, when defining variables in the class, use the following line:

```
1    a_type a_var;
```

and when the programmer defines which datatype 'a_type' is to be when the program instantiates a particular instance of a_class, a_var will be of that type.

When defining a function as a member of a templated class, it is necessary to define it as a templated function:

```
1    template<class a_type> void a_class<a_type>::a_function(){...}
```

When declaring an instance of a templated class, the syntax is as follows:

```
1    a_class<int> an_example_class;
```

An instantiated object of a templated class is called a specialization; the term specialization is useful to remember because it reminds us that the original class is a generic class, whereas a specific instantiation of a class is specialized for a single datatype (although it is possible to template multiple types).

Usually when writing code it is easiest to precede from concrete to abstract; therefore, it is easier to write a class for a specific datatype and then proceed to a template - generic - class. For that brevity is the soul of wit, this example will be brief and therefore of little practical application.

4.)

A.) Question :- What is a file mode? Describe the various file mode options available?

Answer :- Most users that play Ninja Kiwi games will not see MODE files since MODE files are typically stored inside the "Game Mode Definitions" folder in the "Assets" folder of a Ninja Kiwi game installation. These files should not be moved from their location.

MODE files are meant to be referenced by Ninja Kiwi games and are not meant to be opened. However, since MODE files are saved in plain text, they can be opened by a text editor.

File Modes	Description
Ios :: in	Searches for the file and opens it in the read mode only(if the file is found)
Ios :: out	Searches for the file and opens it in the write mode. If the file is found, its content is overwritten. If the file is not found, a new file is created. Allows you to write to the file.
Ios :: app	Searches for the file and opens it in the append mode i.e. this mode allows you to append new data to the end of a file. If the file is not found, a new file is created.
Ios :: binary	Searches for the file and opens the file(if the file is found) in a binary mode to perform binary input/output file operations
Ios :: ate	Searches for the file, opens it and positions the pointer at the end of the file. This mode when used with ios::binary, ios::in and ios::out modes, allows you to modify the content of a file.
Ios :: trunc	Searches for the file and opens it to truncate or deletes all of its content(if the file is found.
Ios :: nocreate	Searches for the file and if the file is not found, a new file will not be created.

4.)

B.) Question :- List and explain the STL components?

Answer :-

C++ STL Components
In today's article, we are going to learn about all the points things that is related to STL in C++ so stay connected because you are going to learn many things in simplest way possible. Let's first understand the full form of the STL in C++. So STL stands for Standard template library.
STL definition
Standard template library (STL) in C++ is the library which provides us the set of C++ template classes which further provides us the generic classes and functions that we are using in implementation of data structures and algorithms.
Components of STL
So basically STL contains four components and those are given below:

1. Containers.
2. Algorithms
3. Iterators
4. Function object.
   These four components are further described properly.

1. Containers:-
   Containers are a one way of organizing data in memory. In other words, we can also say that a container holds the data which is of same type.
   Example – arrays, tree, list etc. we can implement these data structures using containers.
   Containers are further divided into three classifications and those are:-

- Sequence containers
- Associative containers
- Derived containers
- Un-ordered Associative containers.