Part I: Object Oriented Programming using C++

Call by reference

A reference refers to an alias or an alternate name for a variable. When an argument is passed by reference an alias or a reference of the actual is passed to the formal argument.

In call by reference, a reference to an argument is created by suffixing the reference operator or ampersand (&) to the data type of an argument in both the function definition and declaration.

The syntax for declaring a function with reference type parameter is-

```
return_type    function_name(data_type    &identifier)
```

The call by reference mechanism requires no overhead of creating copies of arguments in the function and hence, and it makes the execution faster and it reduces memory utilization. It is efficient to use call by reference mechanism when the argument occupies a lot of memory space and copying the argument is not feasible.

Return by reference

Returning reference is useful in those situations when the function call is to be an Lvalue. An Lvalue is an expression that appear on the left hand side of an equals to sign. In other words a function returning a reference can be placed on left hand side of the assignment statement. To return a reference , the reference operator (&) is suffixed to the return type in the function prototype and definition. The following statement shows the return by reference-

```
char   &change(int    i);
```

inline function

inline function is a function whose body is copied and replaced with the function call statement whenever it is invoked. The keyword **inline** is used before the function name to make it inline. It is an optimization technique used by the compilers as it saves time in switching between the functions otherwise. Member functions of a class are inline by default even if the keyword **inline** is not used.

Syntax of inline Function

```
inline return_type function_name ([argument list])
```

```
{

   body of function

}
```

Inline function is suitable for small functions only. In case of large functions, it increases the execution time slowing down the performance. Also, the code size increases reasonably when a large function is called many times since the calling statement is replaced by the body of function every time. So, in case of large functions, the compiler ignores the programmers request to make a function inline even if the keyword inline is used.

Abstract data type

An **abstract data type** (or ADT) is a class that has a defined set of operations and values. In programming, an ADT has the following features:

- An ADT doesn't state how data is organized, and
- It provides only what's needed to execute its operations

An ADT is a prime example of how you can make full use of data abstraction and data hiding. This means that an abstract data type is a huge component of object-oriented programming methodologies: enforcing abstraction, allowing data hiding (protecting information from other parts of the program), and encapsulation (combining elements into a single unit, such as a class).

Class

A class is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. It is building block, that leads to Object-Oriented programming.. A C++ class is like a blueprint for an object.

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

A class is defined in C++ using keyword **class** followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

Syntax for defining a class is-

class   *class_name*

```
{
        Access specifier:
                                Data members
                                Member functions
};
```

Example of a class-

```
class student
{
        private:
                int   rollno;
                char  name[20];
        public:
                void read()
                {
                        cout<<"Enter the roll no. and name";
                        cin>>rollno>>name;
                }
                void display()
                {
                        cout<<"The roll no. and name are:"<<endl;
                        cout<<rollno<<name;
                }

};
```

Here, private and public are the access specifiers, rollno and name are the data members and read() and display() are the member functions of the class student.

Data member and member function of a class

The variables declared inside a class is called the data members of the class and the functions declared inside a class are the member functions of the class.

For the class-

```
class student
{
        private:
                int   rollno;
                char  name[20];
        public:
                void read()
                {
                        cout<<"Enter the roll no. and name";
                        cin>>rollno>>name;
```

```
            }
            void display()
            {
                    cout<<"The roll no. and name are:"<<endl;
                    cout<<rollno<<name;
            }

};
```

rollno and name[20] are the data members and read() and display() are the member functions.

Object

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Syntax for declaring Objects of a class

```
        ClassName       objectName;
```

```
class student
{
        ……
        ……..
};
```

The declaration of an object of the class student is as follows-

```
  student       S;
```

Accessing data members and member functions:

The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is *obj* to access the member function with the name *printName()* then we will have to write *obj.printName()* .

The public data members are also accessed in the same way as the member functions given however the private data members are not allowed to be accessed directly by the object.

Access specifier of member accessing modes

There are three access specifiers - **private, public and protected**.

private:- In this mode, the members are accessible to only the member functions within the class. Any function outside the class cannot access the private members. In this mode, the members are not inheritable.

public :- In this mode, the members are accessible to any functions outside or inside the class. The members are inheritable in this mode.

protected:- It is similar to the private mode, but the exception here is that the members are inheritable in this mode.

Difference between C and C++

	C	**C++**
1	C supports procedural programming paradigm for code development.	C++ supports both procedural and object oriented programming paradigms; therefore C++ is also called a hybrid language.
2	C does not support object oriented programming; therefore it has no support for polymorphism, encapsulation, and inheritance.	Being an object oriented programming language C++ supports polymorphism, encapsulation, and inheritance.
3	In C (because it is a procedural programming language), data and functions are separate and free entities.	In C++ (when it is used as object oriented programming language), data and functions are encapsulated together in form of an object. For creating objects class provides a blueprint of structure of the object.
4	In C, data are free entities and can be manipulated by outside code. This is because C does not support information hiding.	In C++, Encapsulation hides the data to ensure that data structures and operators are used as intended.
5	C, being a procedural programming, it is a function driven language.	While, C++, being an object oriented programming, it is an object driven language.
6	C does not support function and operator overloading.	C++ supports both function and operator overloading.
7	C does not allow functions to be defined inside structures.	In C++, functions can be used inside a structure.

7	C does not have namespace feature.	C++ uses NAMESPACE which avoid name collisions. A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries. All identifiers at namespace scope are visible to one another without qualification. Identifiers outside the namespace can access the members by using the fully qualified name for each identifier.
8	C uses functions for input/output. For example scanf and printf.	C++ uses objects for input output. For example cin and cout.
9	C does not support reference variables.	C++ supports reference variables.
10	C has no support for virtual and friend functions.	C++ supports virtual and friend functions.
11	C provides malloc() and calloc() functions for dynamic memory allocation, and free() for memory de-allocation.	C++ provides new operator for memory allocation and delete operator for memory de-allocation.
12	C does not provide direct support for error handling (also called exception handling)	C++ provides support for exception handling. Exceptions are used for "hard" errors that make the code incorrect.