**CHAPTER 4**

**FUNCTION AND OPERATOR**

Function/Operator Overloading

\*\* C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

\*\* An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

\*\* When you call an overloaded function, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function is called overload resolution.

**Friend Function**

\*\* A friend function is a function which can access the private and protected member of a class outside the scope of that class.

\*\* A friend function usually takes objects as parameter.

\*\* A friend function can act as a bridge among different classes/objects.

\*\* Member function of a class can be friend function of another class. θ A class can be friend of another class.

\*\* The friend of a friend is not considered a friend unless explicitly specified.

\*\* The declaration of friend function should be made inside the body of class (can be anywhere inside class either in private or public section) starting with keyword friend.

Friend Class

\*\* Similar to friend functions, a friend class is a class whose members have access to the private or protected members of another class.

\*\* To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne:

friend class ClassTwo;

***[ Note by -Jannatul Ferdous Umama(Bristy)]***

**Static Class Member**

\*\* When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

\*\* We can define class members static using static keyword.

\*\* A static member is shared by all objects of the class.

\*\* All static data is initialized to zero when the first object is created, if no other initialization is present. \*\* We cannot put it in the class definition, but it can be initialized outside the class as done in the following example by re-declaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

**Static Member Function**

\*\* By declaring a function member as static, you make it independent of any particular object of the class.

\*\* A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::

\*\* A static member function can only access static data member, other static member functions and any other functions from outside the class.

**This Pointer**

\*\* Every object in C++ has access to its own address through an important pointer called this pointer.

\*\* The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object

\*\* ‘this’ pointer is a constant pointer that holds the memory address of the current object.

\*\* Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

\*\* Suppose that you create an object named x of class A, and class A has a non-static member function f(). If you call the function x.f(), the keyword this in the body of f() stores the address of x.

***[ Note by -Jannatul Ferdous Umama(Bristy)***

**Dynamic Memory: new & delete Operator**

\*\* In C++ the new and delete operators provide built-in language support for dynamic memory allocation and de-allocation.

\*\* Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.

\*\* You can allocate memory at run time within the heap for the variable of a given type using a special operator in C++ which returns the address of the space allocated. This operator is called new operator.

\*\* If you are not in need of dynamically allocated memory anymore, you can use delete operator, which de-allocates memory previously allocated by new operator.

***[ Note by -Jannatul Ferdous Umama(Bristy)]***