


## Agenda

- File IO
- Shallow and Deep Copy
- Copy Constructor
- Operator overloading
- ~~Conversion Function~~

## Stream

- We give input to the executing program and the execution program gives back the output.
- The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called stream.
- In other words, streams are nothing but the flow of data in a sequence.
- The input and output operation between the executing program and the devices like keyboard and monitor are known as "console I/O operation".
- The input and output operation between the executing program and files are known as "disk I/O operation".
- The I/O system of C++ contains a set of classes which define the file handling methods
- These include ifstream, ofstream and fstream classes. These classes are derived from fstream and from the corresponding istream class.
- These classes are designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.
- Standard Stream Objects of C++ associated with console:
  1. cin -> Associated with Keyboard
  2. cout -> Associated with Monitor
  3. cerr -> Error Stream
  4. clog -> Logger Stream
- ifstream is a derived class of istream class which is declared in std namespace. It is used to read record from file.
- ofstream is a derived class of ostream class which is declared in std namespace. It is used to write record inside file.
- fstream is derived class of istream class which is declared in std namespace. It is used to read/write record to/from file.

 stream hierarchy

## Classes for File stream operations

- ios:
  - ios stands for input output stream.
  - This class is the base class for other classes in this class hierarchy.
  - This class contains the necessary facilities that are used by all the other derived classes for input and output operations.
- istream :

- istream stands for input stream.
- This class is derived from the class 'ios'.
- This class handle input stream.
- The extraction operator(>>) is overloaded in this class to handle input streams from files to the program execution.
- This class declares input functions such as get(), getline() and read().
- ostream :
  - ostream stands for output stream.
  - This class is derived from the class 'ios'.
  - This class handle output stream.
  - The insertion operator(<<) is overloaded in this class to handle output streams to files from the program execution.
  - This class declares output functions such as put() and write().
- ifstream :
  - This class provides input operations.
  - It contains open() function with default input mode.
  - Inherits the functions get(), getline(), read(), seekg() and tellg() functions from the istream.
- ofstream :
  - This class provides output operations.
  - It contains open() function with default output mode.
  - Inherits the functions put(), write(), seekp() and tellp() functions from the ostream.
- fstream :
  - This class provides support for simultaneous input and output operations.
  - Inherits all the functions from istream and ostream classes through iostream.

## File Handling

- A variable is a temporary container, which is used to store record in RAM.
- A file is permanent container which is used to store record on secondary storage.
- File is operating system resource.
- Types of file:
  1. Text File
  2. Binary File

### 1. Text File

1. Example : .txt,.doc, .docx, .rtf, .c, .cpp etc
2. We can read text file using any text editor.
3. Since it requires more processing, it is slower in performance.
4. If we want to save data in human readable format then we should create text file.

### 2. Binary File

1. Example : .mp3, .jpg, .obj, .class
2. We can read binary file using specific program/application.
3. Since it requires less processing, it is faster in performance.
4. It doesn't save data in human readable format.

## File Modes in C++

- "w" mode
  - ios\_base::out:
  - ios\_base::out | ios\_base::trunc
- "r" mode
  - ios\_base::in
- "a" mode
  - ios\_base::out | ios\_base::app
  - ios\_base::app
- "r+" mode
  - ios\_base::in | ios\_base::out
- "w+" mode
  - ios\_base::in | ios\_base::out | ios\_base::trunc
- "a+" mode
  - ios\_base::in | ios\_base::out | ios\_base::app
  - ios\_base::in | ios\_base::app:
- In case of binary use "ios\_base::binary"
- In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.
- ofstream: Stream class to write on files
- ifstream: Stream class to read from files
- fstream: Stream class to both read and write from/to files.

## Copy Constructor

- Copy constructor is a parametered constructor of the class which take single parameter of same type but using reference.
- Copy constructor gets called in following conditions:

```
class ClassName
{
public:
    //this : Address of dest object
    //other : Reference of src object
    ClassName( const ClassName &other )
    {
        //TODO : Shallow/Deep Copy
    }
};
```

1. If we pass object( of structure/class ) as a argument to the function by value then on function parameter, copy constructor gets called.
  2. If we return object from function by value then to store the result compiler implicitly create annonymoys object inside memory. On that annonymous object, copy constructor gets called.
  3. If we try to initialize object from another object then on destination object, copy constructor gets called.
  4. If we throw object then its copy gets created into stack frame. To create copy on stack frame, copy constructor gets called.
  5. If we catch object by value then on catching object, copy constructor gets called.
- If we do not define copy constructor inside class then compiler generate copy constructor for the class. It is called, default copy constructor. By default it creates shallow copy.
  - Job of constructor is to initialize object. Job of destructor is to release the resources. Job of copy constructor is to initialize newly created object from existing object.
  - Note : Creating copy of object is expesive task hence we should avoid object copy operation. To avoid the copy, we should use reference.
  - During initialization of object, if there is need to create deep copy then we should define user defined copy constructor inside class.

## Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- Types of operator:
  1. Unary operator
  2. Binary Operator
  3. Ternary operator
- Unary Operator:
  - If operator require only one operand then it is called unary operator.
  - example : Unary(+,-,\*), &, !, ~, ++, --, sizeof, typeid etc.
- Binary Operator:
  - If operator require two operands then it is called binary operator.
  - Example:
    1. Arithmetic operator
    2. Relational operator

- 3. Logical operator
- 4. Bitwise operator
- 5. Assignment operator
- Ternary operator:
  - If operator require three operands then it is called ternary operator.
  - Example:
    - Conditional operator( ? : )
- In C/C++, we can use operator with objects of fundamental type directly.( No need to write extra code ).

```
int num1 = 10; //Initialization
int num2 = 20; //Initialization
int num3 = num1 + num2; //OK
```

- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should overload operator.

```
class Point
{
    int x;
    int y;
};
int main( void )
{
    struct Point pt1 = { 10,20};
    struct Point pt2 = { 30,40};
    struct Point pt3;
    pt3 = pt1 + pt2; //Not OK
    //pt3.x = pt1.x + pt2.x;
    //pt3.y = pt1.y + pt2.y;
    return 0;
}
```

- If we want to use operator with objects of user defined type then we should overload operator.
- To overload operator, we should define operator function.
- We can define operator function using 2 ways
  1. Using member function
  2. Using non member function.
- By defining operator function, it is possible to use operator with objects of user defined type. This process of giving extension to the meaning of operator is called operator overloading.
- Using operator overloading we can not define user defined operators rather we can increase capability of existing operators.

## Limitations of operator overloading

- We can not overloading following operator using member as well as non member function:

1. dot/member selection operator( . )
2. Pointer to member selection operator(.\*)
3. Scope resolution operator( :: )
4. Ternary/conditional operator( ?: )
5. sizeof() operator
6. typeid() operator
7. static\_cast operator
8. dynamic\_cast operator
9. const\_cast operator
10. reinterpret\_cast operator

- We can not overload following operators using non member function:

1. Assignment operator( = )
2. Subscript / Index operator( [] )
3. Function Call operator[ () ]
4. Arrow / Dereferencing operator( -> )

- Using operator overloading, we can change meaning of operator.
- Using operator overloading, we can not change number of parameters passed to the operator function.

## Operator overloading using member function(operator function must be member function)

- If we want to overload, binary operator using member function then operator function should take only one parameter.
- Using operator overloading, we can not change, precedence and associativity of the operator.
- If we want to overload unary operator using member function then operator function should not take any parameter.

```
c3 = c1 + c2; //c3 = c1.operator+(c2);  
  
c4 = c1 + c2 + c3; //c4 = c1.operator+( c2 ).operator+( c3 );
```

## Operator overloading using non member function( operator function must be global function )

- If we want to overload binary operator using non member function then operator function should take two parameters.
- If we want to overload unary operator using non member function then operator function should take only one parameters.

```
c3 = c1 + c2; //c3 = operator+(c1,c2);  
  
c4 = c1 + c2 + c3; //c4 = operator+(operator+(c1,c2),c3);
```

```
c2 = ++ c1; //c2=operator++( c1 );
```

## Overloading Insertion Operator(<<)

-cout is an external object of ostream class which is declared in std namespace.

- ostream class is typedef of basic\_ostream class.
- If we want print state of object on console(monitor) then we should use cout object and insertion operator(<<).
- Copy constructor of ostream class is private hence we can not copy of cout object inside our program
- If we want to avoid copy then we should use reference.
- If we want to print state of object( of structure/class ) on console then we should overload insertion operator.

```
//ostream out = cout; // NOT OK
ostream &out = cout; //OK
```

```
1. cout<<c1; //cout.operator<<( c1 );
2. cout<<c1; //operator<<(cout, c1 );
```

- According to first statement, to print state of c1 on console, we should define operator<<() function inside ostream class. But ostream class is library defined class hence we should not modify its implementation.
- According to second statement, to print state of c1 on console, we should define operator<<() function globally. Which possible for us. Hence we should overload operator<<() using non member function.

```
class ClassName
{
    friend ostream& operator<<( ostream &cout, ClassName &other );
};

ostream& operator<<( ostream &cout, ClassName &other )
{
    //TODO : print state of object using other
    return cout;
}
```

## Overloading Extraction Operator(>>)

- cin stands for character input. It represents keyboard.
- cin is external object of istream class which is declared in std namespace.
- istream class is typedef of basic\_istream class.

- If we want to accept data/state of the variable/object from console/keyboard then we should use cin object and extraction operator.
- Copy constructor of istream class is private hence, we can not create copy of cin object in our program.
- To avoid copy, we should use reference.

```
istream in = cin; // NOT OK
istream &in = cin; // OK
```

- If we want to accept state of object ( of structure/class ) from console( keyboard ) then we should overload extraction operator.

```
1. cin>>c1; //cin.operator>>( c1 )
2. cin>>c1;//operator>>( cin, c1 );
```

- According to first statement, to accept state of c1 from console, we should define operator>>( ) function inside istream class. But istream class is library defined class hence we should not modify its implementation.
- According to second statement, to accept state of c1 from console, we should define operator>>( ) function globally. Which is possible for us. Hence we should overload operator>>( ) using non member function.

```
class ClassName
{
    friend istream& operator>>( istream &cin, ClassName &other );
};
istream& operator>>( istream &cin, ClassName &other )
{
    //TODO : accept state of object using other
    return cin;
}
```

## Index/Subscript Operator Overloading

- If we want to overcome limitations of array then we should encapsulate array inside class and we should perform operations on object by considering it array.
- If we want to consider object as a array then we should overload subscript/index operator.

```
//Array *const this = &a1
int& operator[]( int index )throw( ArrayIndexOutOfBoundsException )
{
    if( index >= 0 && index < SIZE )
        return this->arr[ index ];
    throw ArrayIndexOutOfBoundsException("ArrayIndexOutOfBoundsException");
}
```



```

}

//If we use subscript operator with object at RHS of assignment operator then
expression must return value from array.

Array a1;
cin>>a1; //operator>>( cin, a1 );
cout<<a1; //operator<<( cout, a1 );
int element = a1[ 2 ]; //int element = a1.operator[] ( 1 );

// If we want to use sub script operator with object at LHS of assignment operator
then expression should not return a value rather it should return either address /
reference of memory location.

Array a1;
cin>>a1; //operator>>( cin, a1 );
a1[ 1 ] = 200; //a1.operator[] ( 1 ) = 200;
cout<<a1; //operator<<( cout, a1 );

```

## Overloading assignment operator.

- If we initialize newly created object from existing object of same class then copy constructor gets called.
- If we assign, object to the another object then assignment operator function gets called.

```

Complex c1(10,20);
Complex c2 = c1; //On c2 copy ctor will call

```

```

Complex c1(10,20);
Complex c2;
c2 = c1; //c2.operator=( c1 )

```

```

class ClassName
{
public:
    ClassName& operator=( const ClassName &other )
    {
        //TODO : Shallow/Deep Copy
        return *this;
    }
};

```

- If we do not define assignment operator function inside class then compiler generates default assignment operator function for the class. By default it creates shallow Copy.
- During assignment, if there is need to create deep copy then we should overload assignment operator function.

## Operator overloading using member function vs non member function:

- During operator overloading, if left side operand need not to be l-value then we should overload operator using non member function.
- We should overload following operators using non member function:
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
- During operator overloading, if left side operand need to be l-value then we should overload operator using member function.
- We should overload following operators using member function:
  1. =, [], (), ->
  2. Short hand operators
  3. Unary Operators( ++, -- )

SUNBEAM INFOTECH