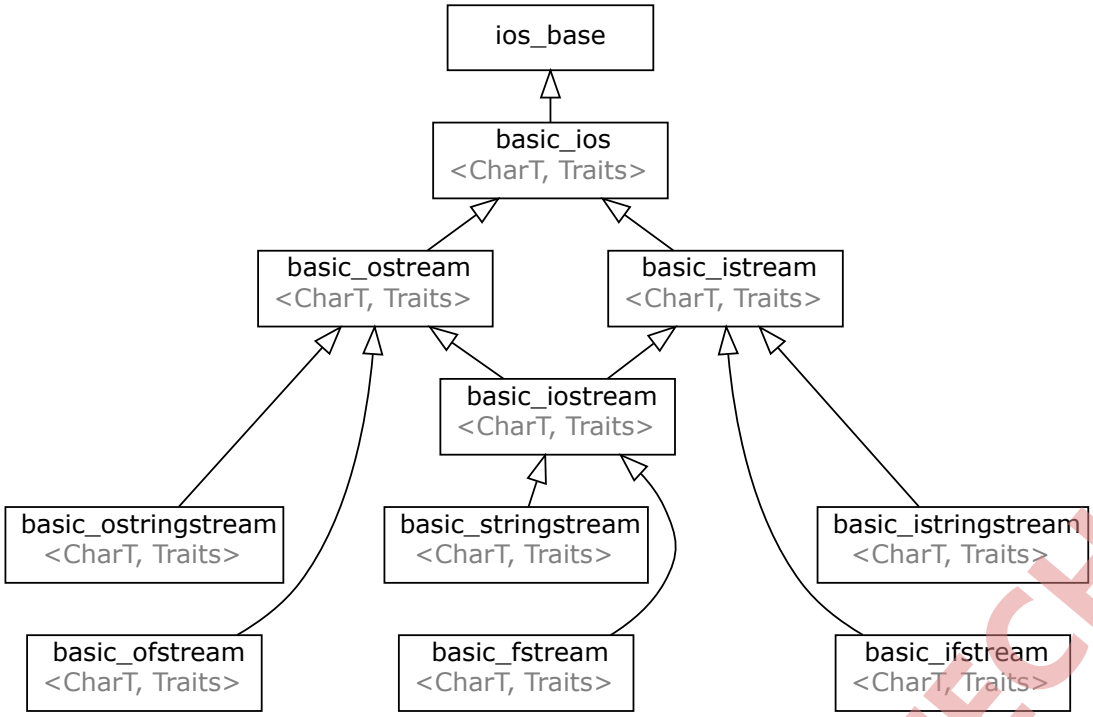
Agenda

- Streams
- File IO
- Nested and Local class
- Copy Constructor
- Shallow and Deep Copy
- ~~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 iostream 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
- ifstearm is a derived class of istream class which is declared in std namespace. It is used to read record from file.
- ofstearm 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 iostream class which is declared in std namespace. It is used to read/write record to/from file.



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 secondry 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 doesnt 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.

Serilization and DeSerilization in binary Files

- When working with string data types or other derived data types (like objects or pointers) in a class and writing or reading the data to/from a binary file, you need to handle serialization and deserialization properly.
- Directly reading or writing the object's memory representation as binary data may not work correctly for derived/user defined data types due to issues like memory layout, internal pointers, and dynamic memory allocation.
- To handle string data types (and other derived data types) correctly when reading or writing binary files, you should implement custom serialization and deserialization functions in your class.
- These functions should convert your object's data into a binary representation (serialization) and reconstruct the object from binary data (deserialization).

```
// Seralizing employee class with datamembers int id,string name,double salary.
void seralize(ofstream &fout)
{
    fout.write(reinterpret_cast<const char *>(&empid), sizeof(int));
    size_t length = name.size();
    fout.write(reinterpret_cast<const char *>(&length), sizeof(size_t));
    fout.write(name.c_str(), length);
    fout.write(reinterpret_cast<const char *>(&salary), sizeof(double));
}

//Deseralizing employee class
void deseralize(ifstream &fin)
{
  fin.read(reinterpret_cast<char *>(&empid), sizeof(int));
```

```
  size_t length;
  fin.read(reinterpret_cast<char *>(&length), sizeof(size_t));
  char *buffer = new char[length + 1];
  fin.read(buffer, length);
  buffer[length] = '\0';
  name = buffer;
  delete[] buffer;
  fin.read(reinterpret_cast<char *>(&salary), sizeof(double));
}
```

Local Class

- If inside a function you declare a class then such classes are called as local classes.
- Inside local class you can access static and global members but you cannot access the local members declared inside the function where the class is declared.
- Inside local classes we cannot declare static data member however defining static member functions are allowed.
- As every local variable declared in function goes on its individual stack frame, such variables cannot be accessed in the local class functions.
- static and global variales gets space on data section which are designed to be shared.
- In a class static data members are designed to be accessed using classname and :: , the static member are designed to be shareed however in local classes we cannot access them ouside the function in ehich they are declared.
- Hence their is no purpose of keeping static data members inside local classes

Nested class

- A class declared inside another class is called as nested class
- Inside Nested class we can access private static members of outer class directly.
- A nested class can access all the private and public members of outer class directly on the outer class object
- Inside Nested class you can access static and global variables.
- As static and global variables are designed to shared they are accessiable inside the nested class.
- Data members gets the memory only when object is created and hence the nested class cannot access outer class data non static data members directly as they do not get memory inside them.

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.