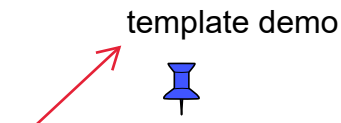
Agenda

- STL
- Modularity
- Streams
- File IO
- Friend Function and Class
- ~~Shallow copy and deep copy~~
- ~~Copy constructor~~
- ~~Singleton class~~

STL



- We can not divide template code into multiple files.
- Standard Template Library(STL) is a collection of readymade template data structure classes and algorithms.
- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a library of container classes, algorithms, and iterators.
- It is a generalized library and so, its components are parameterized.
- Working knowledge of template classes is a prerequisite for working with STL.
- STL has 4 components:
    1. Algorithms what kind of operations can be performed on contents of container
    2. Containers what data structure to use to store data.
    3. Functions overloaded function call operators(functors)
    4. Iterators to traverse the container

1. Algorithm

- They act on containers and provide means for various operations for the contents of the containers.
    - Sorting
    - Searching

2. container

- Containers or container classes store objects and data.

3. Functions

- The STL includes classes that overload the function call operator.
- Instances of such classes are called function objects or functors.
- Functors allow the working of the associated function to be customized with the help of parameters to be passed.

4. Iterators

- As the name suggests, iterators are used for working upon a sequence of values.
- They are the major feature that allows generality in STL.
- Iterators are used to point at the memory addresses of STL containers.
- They are primarily used in sequences of numbers, characters etc.
- They reduce the complexity and execution time of the program.

Container Categories

- The C++ container library categorizes containers into four types:

1. Sequence containers
2. Sequence container adapters
3. Associative containers
4. Unordered associative containers

Sequence Containers

- Sequence containers are used for data structures that store objects of the same type in a linear manner.
- The STL Sequence Container types are:
- array
  - represents a static contiguous array
- vector
  -represents a dynamic contiguous array
- forward_list
  - represents a singly-linked list
- list
  - represents a doubly-linked list
- deque
  - represents a double-ended queue, where elements can be added to the front or back of the queue.

Vector

- Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.
- Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.
- In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens.
- Inserting and erasing at the beginning or in the middle is linear in time.
- begin() – Returns an iterator pointing to the first element in the vector
- end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
- size() – Returns the number of elements in the vector.

when you use list.begin() and try to output i t directly with std::cout, it won't give you the expected output because list.begin() returns an iterator, not a value that can be directly printed.

```
vector<int> list ;
  list.push_back(1);
  list.push_back(2);
  list.push_back(3);
  list.push_back(4);

  for(auto it = list.begin(); it != list.end(); it++){
     cout << *it << " ";
  }
```

- empty() – Returns whether the container is empty.
- front() – Returns a reference to the first element in the vector
- back() – Returns a reference to the last element in the vector
- assign() – It assigns new value to the vector elements by replacing old ones
- push_back() – It push the elements into a vector from the back
- pop_back() – It is used to pop or remove elements from a vector from the back.
- insert() – It inserts new elements before the element at the specified position
- erase() – It is used to remove elements from a container from the specified position or range.

Stream

```
list.insert(list.begin()+2,6);   list.begin() returns address
```

```
std::vector<int> vec = {1, 2, 3, 4, 5};
auto first = vec.begin() + 1; // Iterator pointing to the second element (index 1)
auto last = vec.begin() + 3; // Iterator pointing to the fourth element (index 3)
vec.erase(first, last); // Remove elements from index 1 to index 2 (excluding index 3)
```

- 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().

- streambuf :

    - This class contains a pointer which points to the buffer which is used to manage the input and output streams.

- fstreambase :

    - This class provides operations common to the file streams. Serves as a base for fstream, ifstream and ofstream class.
    - This class contains open() and close() function.

- 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.

- filebuf :

    - Its purpose is to set the file buffers to read and write.
    - We can also use file buffer member function to determine the length of the file.

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.

Modularity (Multiple Files)

- "/usr/include" directory is called standard directory for header files.
- It contains all the standard header files of C/C++
- If we include header file in angular bracket (e.g #include<filename.h>) then preprocessor try to locate and load header file from standard directory only(/usr/include).
- If we include header file in double quotes (e.g #include“filename.h") then preprocessor try to locateand load header file first from current project directory if not found then it try to locate and load from standard directory.

```
// Header Guard
  #ifndef HEADER_FILE_NAME_H
  #define HEADER_FILE_NAME_H
    //TODO : Type declaration here
  #endif
```

Friend function & class

- If we want to access private members inside derived class
- Either we should use member function(getter/setter).
- Or we should declare a facilitator function as a friend function.
- Or we should declare derived class as a friend inside base class.
- Friend function is non-member function of the class, that can access/modify the private members of the class.
- It can be a global function.
- Or member function of another class.
- Friend functions are mostly used in operator overloading.
- If class C1 is declared as friend of class C2, all members of class C1 can access private members of C2.
- Friend classes are mostly used to implement data struct like linked lists.

Assignments

1. Copy Day08->Demo07.cpp and convert the static array into the vector<Perosn *>. Do the necessary changes in the code and execute successfully
2. cretae a class student(rollno,name,vector) who can take any no of courses. create a class of course (id,course_name,fees) hint- create vector of course

```
class Course{
int cid;
string course_name;
```

```
double fees;
};

class Student{
int rollno;
string name;
vector<Course> coursesList;
public:

  void addcourse(){
    Course c;
    coursesList.push_back(c);
  }
};
findStudent(vector<Student *> &obj){
  int rollno;
  for()
  if(obj[i].getRollno() == rollno)
}
int menu(){
  cout<<"1. Add Student-";
  cout<<"2. Add Course - "; ->
}
int main(){
  vector<Student *> studentList;
}
```