C++ Streams

PAGE NO.: _____
DATE: / /

C++ stream library is the primary means by which a C++ program can interact with its environment, namely the user and the file system.

The basic unit of comm^n b/w a prog. and its environment is a stream.
A stream is a channel b/w a source and dest^n which allows source to push formatted data to the destination.

The type of source and dest^n varies from stream to stream.

When using the streams library to read and write data, you don't need to read or write all of the data at once. It is perfectly legal and common to read data one piece at a time.
cout. cout for (character output) is a stream connected to the console, a text-window that displays plain text data.

PAGE NO.: _____
DATE: / /

Stream

```
#include <iostream>
using namespace std;
int main( ) {
    cout<< "I'm good" << endl;
    return 0; }
```

→ endl → end line and it prints a newline character to cout stream.

→ << operator is called the stream insertion operator and is c++ operator that is used to push ~~this special object~~ data into stream object.

Streams are very versatile and can write any data types to stream object.

Ex:

```
cout << 137 << endl // integer
cout << 2.718 << endl // real number
cout << "Here its good" << endl // string.
```

PAGE NO.: _____
DATE: / /

→ we provide the <simpio.h> header file, which exports the i/p func^n GetLine, GetInteger, GetReal, and GetLong. These not part of C++.

→ Streams library have streams object called cin (character input) which read value from user.

→ >> ⇒ stream extraction operator.

eg:

```
int x;
cin >> x;
```

hit

provide integer and enter, ~~stored~~ stored in x variable and ready to use for prog.

→ Multiple values can be read from cin.

```
cin >> x >> y;
```

x & y can be of same or different type.

→ can be used with endl ie.

```
cin >> x >> endl;
```

↳ generate error.

PAGE NO.: _____
DATE: / /

Reading data from Keyboard :- cin, cout

Reading & Writing Files :-

C++ provides a header file called <fstream> that support ifstream & ofstream that perform input file & out file stream resp.

Reading in File:

⇒ ifstream myStream("myFile.txt");
reads from file.

⇒ Reading data from mystream.

```
ifstream mysfream ("myfile.txt");
int myInt;
mysfream >> myInt;   Read integer from myFile.txt.
```

↳ readstream variable instead of ifstream.

Also:

```
ifstream myStream;
myStream.open("myFile.txt");
```

PAGE NO.: _____
DATE: / /

Writing in File:

ofstream myStream("myFile.txt");

→ if writing to non-existing file then new file will be created.

→ if it opens a existing file then it will overwrite the file.

Stream Manipulator:-

ex:- endl — stream manupulator, object that can be inserted into a stream to change some sort of stream property.

ex:-

(1) boolalpha — cout << boolalpha << true << endl
↳ weather of not stream output value true or false.

(2) setw(n) — cout << setw(c) << 10 << endl
↳ sets minimum width of o/p for next stream operation. ⇒ ____ 10 (width)

(3) hex, dec, oct — cout << hex << 10 << endl → 10
cout << dec << 10 << endl → 10
cout << oct << 10 << endl → 12
cin >> hex >> x → Read hex value for x.

(4) ws — myStream >> ws >> value;
skip whitespace stored in the stream.

PAGE NO.: _____
DATE: / /

Alternative:

(1) getline → string mystr;
getline(cin, mystr);

(2) A string Buffer: stringstream — exported by the header <sstream>.
- stringstreams store data in temporary string buffers.
- insert data into a stringstream to convert data to string.
- and extract data from a stringstream to convert string data into a different format.

eg: stringstream myConverter;
int myInt;
double myDouble;
string mystring;
// Insert string data
myConverter << "137 Hello 2.71828";
// Extract mixed data
myConverter >> myInt >> myString >> myDouble;

PAGE NO.: _____
DATE: / /

(3) Get Integer :- the user Enter an Integer and returns only after the user enters valid i/p.

PAGE NO.: _____
DATE: / /

* If we discover a linker error, always check to make sure that you have implemented all func" you have prototyped and that those implementations match the prototypes.

C++ Across Multiple Files:-

(1) Split a program

(2) Best way to split a program.

Abstraction using headerfiles which do complicated functions and ~~write different~~ just show and do it in main function.

* # include → used to import library code into prog..
  - # include "genlib.h" } 2 form of using #include
  - # include <iostream>
- (1) # include "___.h" → preprocessor will look in the same directory for file.
- (2) # include <___> → preprocessor will look in the compiler specific directory containing C++ stdlib

PAGE NO.: _____
DATE: / /

#include is preprocessor directive, not C++ statement.

* #define phrase replacement
After encountering #define directive, whenever preprocessor find phrase it replace with replacement.
ex: #define PIE 3.14
phrase → it should be a single word without spaces.
replacement → anything that follows phrase and also it can be nothing then in prog. phrase means nothing when encountered.
if #define terminated with ";" then #preprocessor treat it as part of replacement!
ex: #define CNT 3;
int x = CNT * 3;
↓ preprocessed as
x = (3;) * 3
↓
error.

PAGE NO.: _____
DATE: / /

⇒ strongly preffered to define constants using const keyword.

Any header file contains ~~include guards.~~ include guards.

include guards:

```
#ifndef File_Included
#define File_Included

#endif
```

⇒ If 2 same headers are included in the prog. then there is error this is easy to solve in small prog but for large program it may not be easy to solve this 2 same header problem.
for this we have conditional statements for preprocessor also.
#if, #elif, #else, #endif

PAGE NO.: _____
DATE: / /

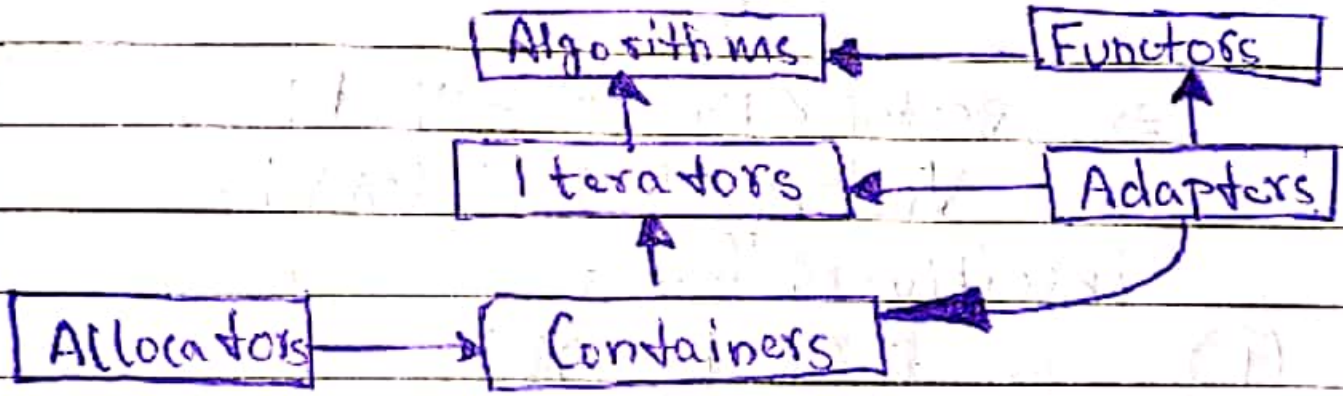
```
inline int Max(int a, int b){
    return a>b? a:b; }
```

STL

6 parts:

- Container
- Iterator
- Algorithms
- Adapters
- Functors
- Allocators

Ref[4]:



PAGE NO.: _____
DATE: / /

Vectors:

- access a variable number of objects.
- vector is an object that rep. a seq. of elements.

eg: `vector<int> myVector;`

- Can store own custom structs in a vector;

eg:
```
struct MyStruct {
    int myInt;
    double myDouble;
    string myString;
};
```

⇒ `vector<MyStruct> Vec1;`

~~⇒ `Vec1.push_back(n);`~~

inserting element:-

(1)
```
int n;
n = GetInteger();
vector1.push_back(n);
```
it append sequentially in the vector.

(2)
```
Vector.insert(vector.begin()+n, e);
```
eg: `v.insert(v.begin()+5, 42);`
↳ now 6th element is 42 in V.

PAGE NO.: _____
DATE: / /

Operations :-

→ eg: vector<int> myVector(20,10);
no of element in vector | default value of each element

but this is not possible!

vector<int> myVec;
myVec(20,10); ⟶ not possible
Error occur.

→ if vector<int> myVec(20) then it automatically make default value 0 to it.

Can resize length of vector :-

vector<int> myVec(2);
myVec.resize(5); // 00000
myVec.resize(7,1); // 0000011
myVec.resize(1,7); // 0
↳ second parameter is ignored

Erase / Delete elements :-

remove a single element from a random point in vector,

⇒ myVec.erase(myVec.begin()+n);

⇒ erase the vector content; myVec.clear().

myVec.erase(myVec.begin()+a, myVec.begin()+b);
↳ erase element from a to b.

PAGE NO.: _____
DATE: / /

➔ myVec.back() = 5
or int lastElem = myVec.back()
return reference to the last element in the vector.

➔ myVec.pop_back();
remove the last element from vector.

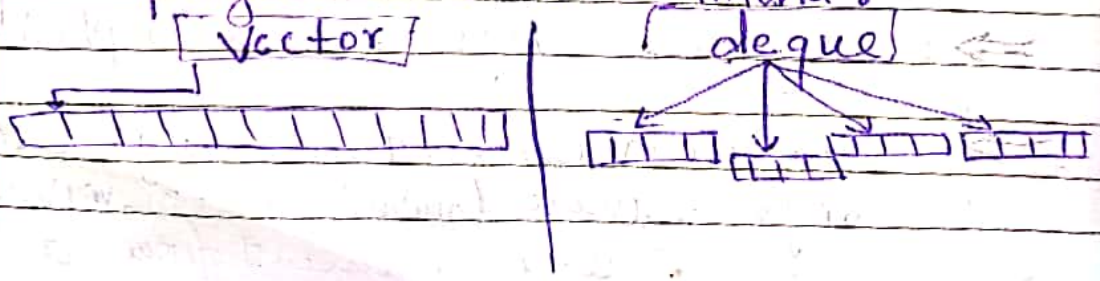
Deque :

⇒ Vector cannot extend and shrink on both side this can be problem for many cases.

⇒ Deque expand and shrink on both side.

Vector stores element in contagious order in memory.

Deque maintain a list of different pages that store info.



PAGE NO.: _____
DATE: / /

deque<int> values;
values.insert(values.begin() + n, e)
↳ insert e at nth index.

** push_front() — push elements into a deque from the front.

** push_back() — element at the back.

** pop_front() & pop_back() — remove from front & back.

** front() & back() — reference to front & back element.

valarray : similar to vector but specially for numerical computations.
support of mathematical operators.
eg: myValArr *= 2; multiply all elements of valarrays by 2.

PAGE NO.: _____
DATE: / /

STL Associative Containers & Iterators.

⇒ map, set, multimap & multiset.

Imp:— You will need to roll an n sided die roughly $\sqrt{n}$ times before one same no. will come up twice.

Set —
unordered sequence of elements, content of elements are more important than the actual sequence those elements are in.

```
set<int> generated;
generated.insert(x);   O(1)
```
↳ insert in an unordered manner without ordering the i/p.

→ Checking if element exist in set:—
```
generated.count(x):
```
return true if x exist or false if not.

complexity — O(log n)

PAGE NO.: _____
DATE: / /

- set doesn't allow duplicate elements.
- set can store all primitive types, strings and other STL containers. But cannot store custom struct inside set.

  i.e. struct Point {
  double x, y;
  };
  set <Point> myset;
  ↳ Illegal

* Set is structured on top of balanced binary tree.
- erase — myset.erase(x); if x exist.

  complexity — O(1)

PAGE NO.: _____
DATE: / /

Iterator - provide way for accessing data stored in container.