Date:

*) In multiple Inheritance,
class C : public A, public B.
{}
Invoking of constructor ⇒ A, B, C.
Here base classes are constructed first in order which they appear in declaration of derived class

*) In multilevel Inheritance
class A {}
class B : public A {}
class C : public B {}
constructor ⇒ A, B, C
constructor is executed in order of inheritance

*) In case of virtual base class
constructor of virtual base class before any non virtual class.
class A {}
class B {}
class C {}
class D : public A, virtual public B, public C;
{}
Constructor ⇒ B, A, C, D.

*) for single Inheritance class A : public B {}
Constructor ⇒ B, A,

9/10/2012

INITALIZATION LIST

Constructor_name(parameter list):
Initalization list
{assignment section}

eg:-

```
class ABC
{ int a, b;
  public:
  ABC(int x, int y): a(x+y), b(x-y)
  {}
};
```

3/10/2022 Date:

POLYMORPHISM

Compile time. Run time

↳ function overloading ↳ virtual function

↳ Operator overloading (Dynamic binding)

(Static Binding).

Poly + morph + ism → many + forms + noun suffix

user defined – structure, class, enum, union

derived data types – Array, pointers

(*) A pointer that can refer to variable of any data type are called generic pointers or void pointer. before using these pointer we must typecast the variables to the specific data type that they point

(*) The pointer that are not initalized in the program are the null pointer

fointers of any data type can be assigned with the value i.e. 0 or NULL

```
int a=10;
int *ptr = &a; // ptr = a;
    *ptr → show 10
    ptr → show add of a
```

POINTER WITH ARRAY & STRING

An array and string name is pointer itself and pointing to the 1st element's address of array / string

POINTERS TO FUNCTION.

The pointer to a function is also called as function pointer (call back function)

Pointer to function is used to

1. Referring to a function
2. Function can be selected dynamically at runtime
3. Function can be passed as an argument to another function.

```
Void display (int a, int b)
{ int c, d;
  c=a; d=b;
    cout << c << d;
}
void main()
{ void (*ptr)(int, int);
    ptr = &display;
    ptr(10,20);
}
```

FUNCTION AS ARGUMENT

```
void sum (int a, int b, (*ptr) (int c, int d))
{ int dp;
  dp= a+b+c+d;
}
```

POINTER TO OBJECT

```
class item
{ int code;
  float price;
  public:
  void getdata (int a, float b)
    { code =a;  price = b; }
  void showdata ()
    { cout << "code :" << code << "\t Price:" <<
                price;
    }
};
Void main()
  {
  item x; item *bptr;
  bptr. = &x;
  (*bptr).getdata (10, 2);
  (*bptr). show ();
  bptr → getdata (20, 4);
  bptr → show ();
{
```

THIS POINTER.

It is a pointer which tells us that which object is used to excess the members of the class
This unique pointer is automatically passed to a member function when it is called. A pointer 'this' acts as implicit argument to all the member functions
'This' pointer represents an object that invokes a member function

```
class person
{ int age;
  public: void init(int a){age=a;}
  person greater(person x)
  { if (x.age > age)
      return x;
    else return *this;
  };
void main()
{ person t1, t2, t3;
  t1.init(40);
  t2.init(20);
  t3.greater(t2);
}.
```

Date:

POINTER TO DERIVED CLASSES.

```
class B
{}
class D: public B
{}
void main()
{ B x;
  B * bptr;
  bptr = &x;
  D y;
  bptr = &y; // as derived class has
                prop. of base class.
}
```

A base class pointer accesses the member function depending upon the type of pointer rather than the pointer pointing to object.

In above eg. bptr can't excess original member of derived class but only derived one.

VIRTUAL FUNCTION

(*) Another way of implementing the polymorphism concept is virtual function

(*) Base class ptr is able to call base class as well as derived class function

(*) Incase of virtual function, compiler doesn't depend on type of ptr but depends on type of object

```
class B
{ public:
  virtual void show()
  void display();
  {
  };
class D: public B
{ show();
  display();
};
void main()
{ Base x;
  Base *ptr;
  ptr = &x;
  ptr -> show(); // -> B.show
  ptr -> display(); // B.display
  Derived y;
  ptr = &y
  ptr -> show();
  ptr -> display();
```

Pure v
virtu
virtu

* These
virtual
* If bas
Then
redefi
* Runtin

Date:

Pure virtual function.

virtual void show() {}
virtual void display() = 0; } Pure virtual function

* These do nothing function known as pure virtual function.
* If base class have pure virtual function, Then it is compulsory for derived class to redefine them.
* Runtime polymorphism.

TEMPLATES :- (Macro)

* Generic program works with variety of data types.

class template → Generic datatype
template < class T > → tells compiler
class class-name that we are creating a template
{
Body
}

Eg :-
```
#include <iostream.h>
#include <conio.h>
template <class T>
class xyz
{ T a;
  public:
  xyz(T x)
  { a=x; }
  void dis()
  { cout << "a" << a; }
};
void main()
{ xyz <int> t1(10);
  xyz <float> t2(50.7);
  xyz <char> t3('R');
  t1.dis();
  t2.dis();
  t3.dis();
}
```

Syntax :
class-n

when a t
data t
* Template
data ty
* Templat
progra

With mu
templat
class
{ T1 a
T2 b
xyz
{ a
void
{ c
};
void
{ x
x
}

TEMPLATES :- (Macro)

* Generic program works with variety of data types.

class template → Generic datatype
template < class T > → tells compiler that we are creating a template
class class-name
{
Body
};

Eg :-
```
#include <iostream.h>
#include <conio.h>
template <class T>
class xyz
  { T a;
    public:
    xyz(T x)
    { a=x; }
   void dis()
    { cout << "a" << a; }
};
void main()
{ xyz <int> t1(10);
  xyz <float> t2(50.7);
  xyz <char> t3('R');
  t1.dis();
  t2.dis();
  t3.dis();
}
```

Syntax:

class_name <data type> object name (arg1);

when a template class is provided with data type it is called as instantation

* Template can work with every type of data type.
* Template is used to make generic program or function

With multiple argument

```
template <class T1, class T2>
 class xyz
 { T1 a;
   T2 b;
   xyz( T1 x, T2 y)
   { a=x; b=y; }
   void dis ()
   { cout << "a" << a; }
 };
void main ()
 { xyz <int, float> t1 (10, 10.5);
   xyz <float, char> t2( );
   xyz <char, double> t3( );
}
```

The process of converting a generic program to specific type is known as instantation of a class. Its only after conversion from generic type to specific type that the compiler analysis it for error.

TEMPLATE FUNCTION

```
template <class T>
return type function (parameter type)
{ //func body
}

void show (T a)
{ cout << "a is" << a; }
void main()
{ show ('R');
  show (10);
  show (50.6);
}
```

With Multiple argument

```
Template <class T1, class T2>
void show (T1 a, T2 b)
{ cout << "a is" << a;
  cout << " b is " << b;
}
```

void ma
{
show ("
show(
show(
}

OVERLOAD
A templa
by tem
its nam
resolutio
1) Call an
exact
2) Call a
created
3) Try no
ordinar
matches
match
conversi
only ten

templat
void s
{ cout
cout
}

```
void main()
{
show('R',6);
show(10.7,6);
show(50,'N');
}
```

OVERLOADING OF TEMPLATE FUNCTION.

A template func can be overloaded either by template func. or ordinary func of its name. In such cases, overloading resolution is accomplished as follows:

1) Call an ordinary function that has an exact match
2) Call a template function that could be created with an exact match
3) Try normal overloading resolution to ordinary function & call the one that matches. An error is generated if no match is found. and that no automatic conversions are applied to arguments only template function

```
template <class T)
Void show (T a)
{ cout << a;
  cout << "show template;
}
```

```
void show(int a)
{ cout << a;
  cout << "show ordinary";
}
void main()
{ show(10.0); ---> Template
  show(60); ---> ordinary
  show('A'); ---> Template
}
```

MEMBER FUNCTION TEMPLATE

Syntax of fn outside class

```
template <class T>
return_type class_name<T>::
            fn_name(arg list)
{ // Body
}
```

Eg :-

```
Template <class T>
  class xyz
  { T a;
    public:
      xyz(T x);
  };
xyz<T>::xyz(T x)
  { a=x; }
```

NON TYPE TEMPLATE ARGUMENT.

A template can include the built-in type, string, function name, constant expression along with the generic date type

Eg :-
```
template <class T, int size>
    class arr
    { T a[size];
    }
void main()
{ arr < int, 5> a1;
    arr<float, 10> a2;
    arr < char, 20> a3;
}
```

PARAMETRIC FUNCTION/CLASSES

We know that templates are defined with Generic parameters and these parameters are replaced by specific data type at the time of actual. thats why template are sometime known as parametric classes or function.

USING PARAMETERS FOR POLYMORPHISM

1) function overloading →
2) Parametric polymorphism → Template

9/11/2016

CLASS HIERARCHY.

1. Classification Hierarchy

A relation b/w classes in which one class can be said to be a kind of other class. It is of 4 types a) Generalisation
b) Specialization.
c) Composition
d) Aggregation

Generalisation → Relation b/w class in which one class can be a kind of other

Composition → abt expressing relation. b/w object
Composed object made up of other object

Aggregation → is type of composition also made up of other obj. but other obj. may not be entirely participating in the main object

EXCEPTIONAL HANDLING

Basics:

Types ① Synchronous → Inside the Hard disk.

② Asynchronous → Due to keyboard Interrupt

Synchronous → array out of index value or overflow

(*) Exception handling deals with synchronous ones.

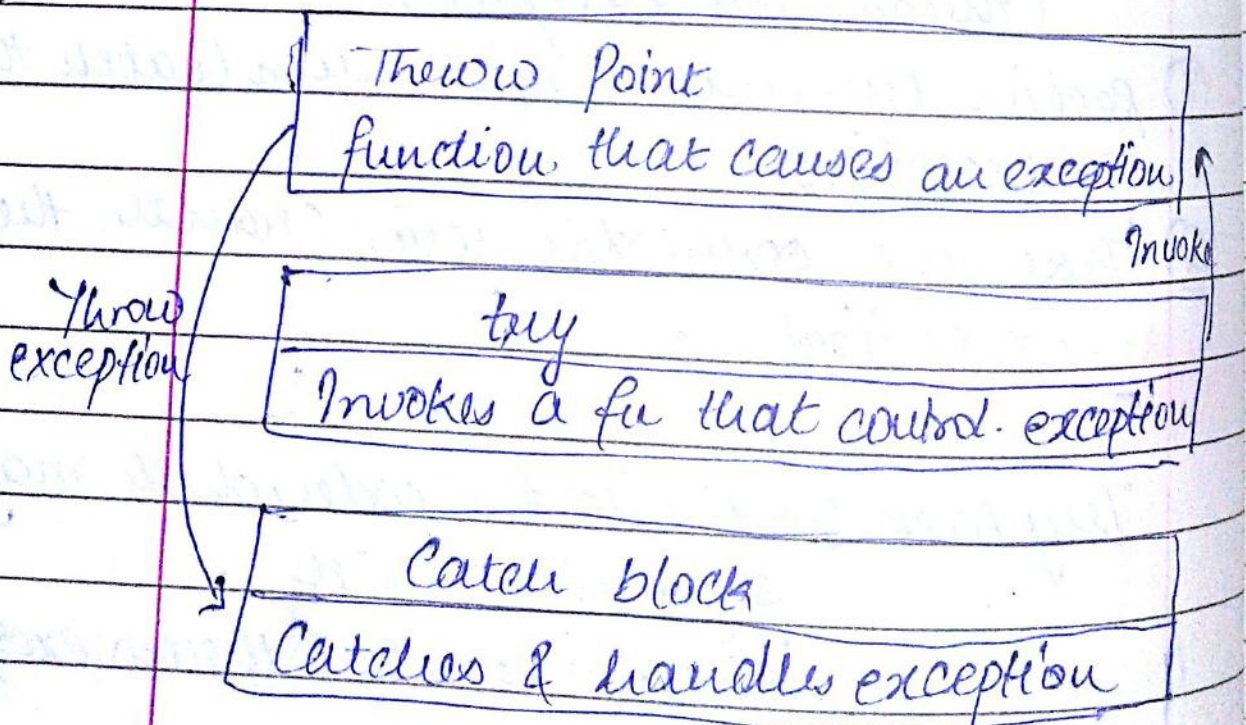
① Find the problem (that the exception)
② Inform that an error has occured (throw an exception)
③ Receive the error information (catch the exception)
④ Take the corrective action (handle the exception)

Try block :→ Write the code which may generate error.

Try block → Detects & throws exception

Catch → Catches & handles except

Eg:-
```
void main()
{ int a, b, c;
  cin >>a>>b>>c;
  try
  { if((b-c)!=0)
      cout << a/b-c;
  }
  else
    throw (b-c);
  }
  catch (int x)
  {
  cout << "Exception caught";
  }
}
```



Date:

```
return_type function name (arg list)
 { throw (object); // throw pt.
 }
try
 { // Invokes a function
 }
catch
 { // handles the exception
 }
```

eg:-
```
void divide (int a, int b, int c)
   {
      if(b-c)!=0
      cout << a/(b-c);
      else
        throw (b-c);
   }
void main()
{ try
  { divide (10,20,60);
          "  (60,50,40);
          "  (70,50,50);
  }
catch( int c)
{ cout << "exception caught" << c;
  }
{
```

Throwing Mechanism.

throw (exception object);
throw Exception-object;
throw;

Catching Mechanism.

* Data type of catch must match the that of object type

```
catch(datatype arg)
{ // write the code to handle
    the exception
}
try
{}
```

```
catch(type 1 arg)
catch( " 2 " )
catch( " 3 " )
```

* first data type is matched with 1st if matches then execution else checks with others.
If doesn't matches at all then abrupt ending of program.

Rethrowing an exception-

```
try
{ try {}
  catch() { throw; }
}
catch()
```

A file is
that is
of hard
* we use
data.

writing
data
to file

cin>>
get
data

write da

Program

read
data

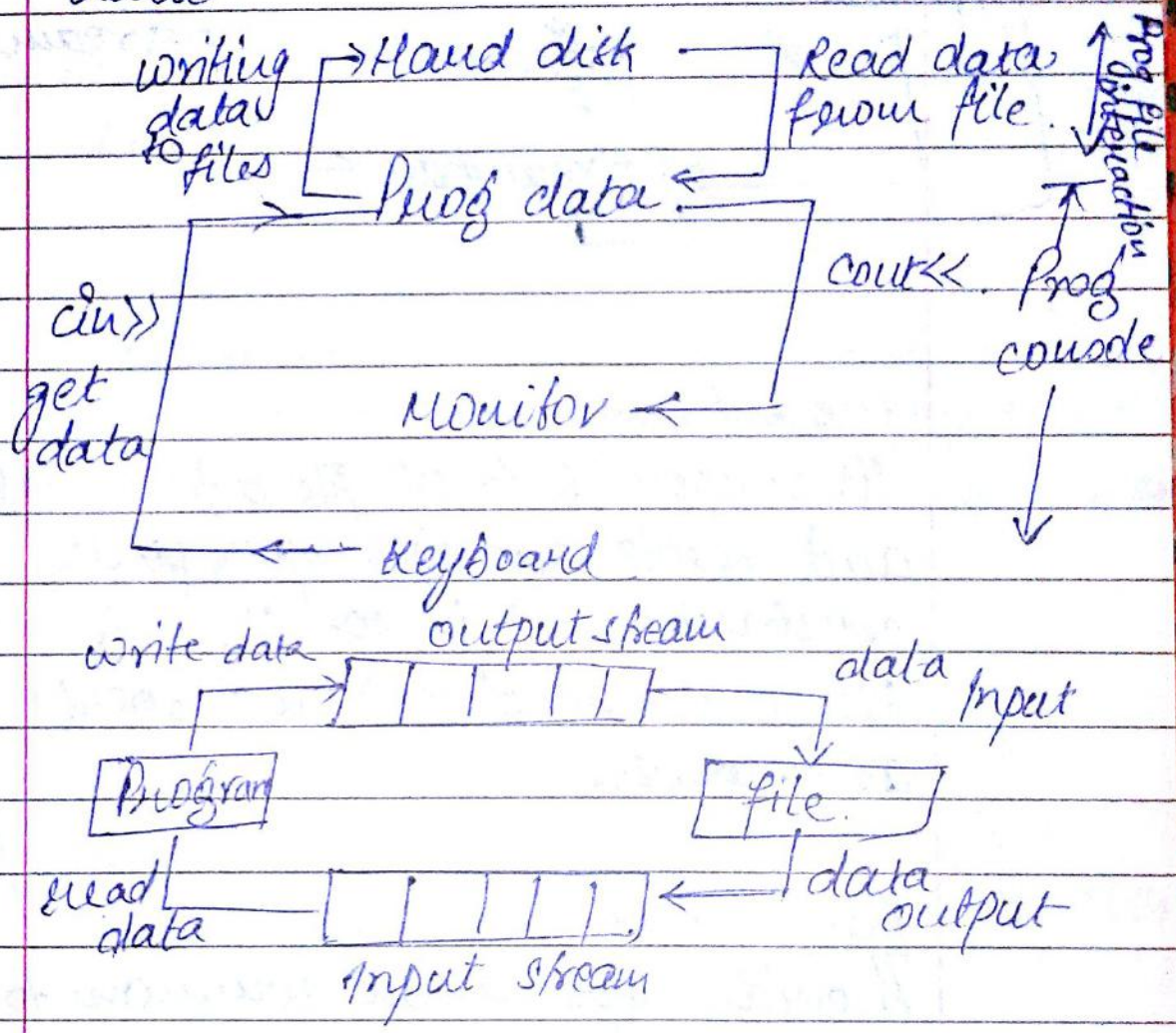
* Input sh
data f
* Output
writes

CLASSES

FILE HANDLING

A file is a collection of related data that is stored on a particular area of hard disk.

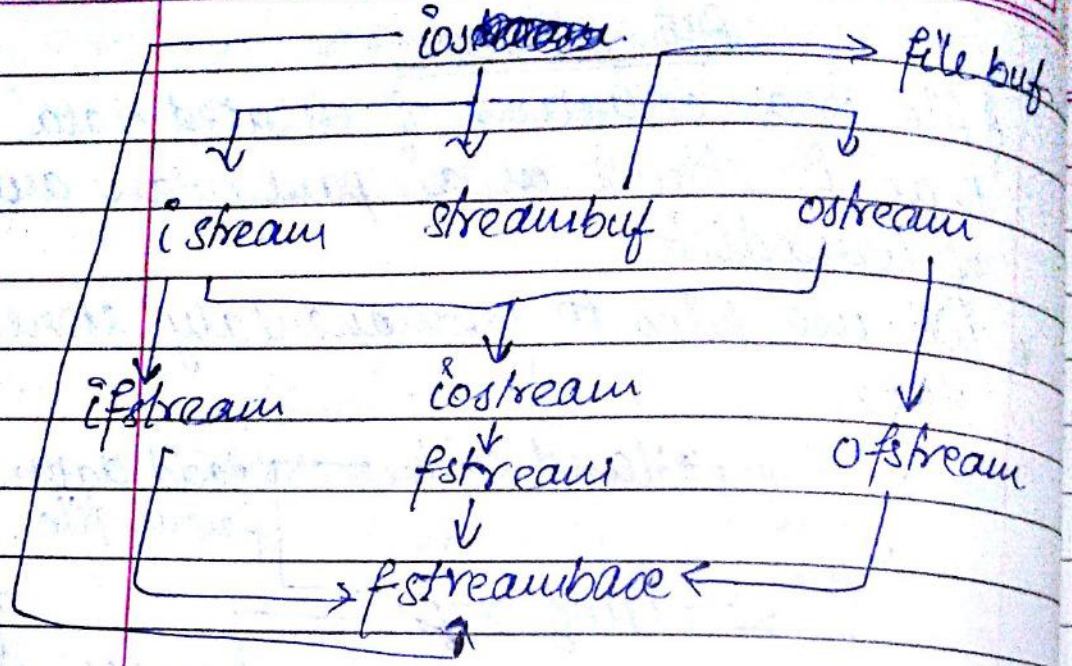
* We use files to permanentaly store data



(*) Input stream extracts the data from data file and provide to program.

(*) Output stream gets data from prog & writes to files.

CLASSES FOR FILESTREAM OPERATION



filebuf:
Its purpose is to set file buffers read and write. Contains open port constant used in open().
Also contains close() and open() as members.

fstreambase:
Provides operation common to file stream. Serves as base for ifstream, fstream, ofstream classes also contain open & close functions.

ifstream
Provides input operation.
Contain open with default in

put mod
read, se

ofstream
Provid
with de
seekp,
ostrea

fstream
Provide
operatio
I/p mod
& ostrea

(1) Name o
(2) Data ty
(3) Purpos
(4) Openin

TWO M
(1) Using c
single
(2) Using
(multip

Eg:- ios
fs

put mode. Inherit function get, getline read, seekg, tellg from istream

ofstream

Provides output operation. Contain open with default o/p mode. Inherits put, seekp, tellp, write function from ostream.

fstream

Provides support for simultaneous I/O operations. Contains open with default I/p mode. Inherits all the fn from istream & ostream classes than iostream.

1. Name of file
2. Data type & structure
3. Purpose
4. Opening Method

] factors to be noted before working with files

TWO METHODS FOR OPENING FILE

1. Using constructor function (used to open single file)
2. Using member function [open()] (multiple files)

Eg :- iostream.h.
fstream.h.

using cout()

```
main()
{ char str[100];
  cin >> str;
  ofstream f1("MSIT.txt");
  f1 << str;
  f1.close();
  ifstream f2("MSIT.txt");
  f2 >> str;
  cout << str;
  f2.close();
}
```

Eg:-

```
main()
{ ofstream outf("ITEM.txt");
cout << "Enter Item Name";
cin >> Name;
outf << name << "\n";
cout << "Enter item cost";
 cin >> cost;
outf << cost << "\n";
outf.close();
ifstream inf("Item.txt");
 inf >> name;
inf >> cost;
cout << "\n";
cout << "Name:" << name;
cout << "\n cost:" << cost;
inf.close();
```

{

Detecting

(1) while(

{ char

f >>

cou

}

(2) (End

if (

{

}

exit(0);

exit(1);

}

Detecting End of file

(f) Input stream.

(1) while (object)
{ char ch;
f >> ch;
cout << ch;
}

(2) (End of file) eof()
if (f.eof() != 0).
{ exit (1);
}

exit (0) → normal function
exit (1) → forceful function

9/11/2016

FILE HANDLING

Date:

```
#include <iostream.h>
#include <fstream.h>
void main()
{ ofstream fout;
  fout.open("country.txt");
 fout << "united. States of America\n";
 fout << "united Kingdom \n";
 fout << "South Korea \n";
 fout.close();
 fout.open("capital.txt");
 fout << "washington";
 fout << "London";
 fout << "Seoul";
  fout.close();
 const int N = 80;
 char line[N];
 ifstream fin;
 fin.open("country.txt");
 cout << "Contents of country file";
 while(fin)
 { fin.getline(line, N);
   cout << line;
 }
 fin.close();
 fin.open("capital.txt");
 cout << "Contents of capital file \n";
 while(fin)
 { fin.getline(line, N)
```

cout
{
fin.
getc
}

open(

Modes

ios::app
ios::at
ios::bi
ios::no
ios::no
ios::tru
ios::in
ios::ou

FILE PO

1) Output
2) Input

seekg(

seekp(

tellg(

tellp()

```
cout << line;
}
fin.close();
getch();
{
```

open (filename, mode)

Modes

(*) ios::app :- Append to end of file
(*) ios::ate :- Goto end of file on opening
(*) ios::binary :- Creates binary file
(*) ios::nocreate :- Open fail if file don't exists
(*) ios::noreplace :- " " " " already "
(*) ios::trunc :- deletes content of file if " "
(*) ios::in :- open file for read only
(*) ios::out :- open file for write only

FILE PTR AND MANUPULATION.

1) Output (put) pointer → Ist if in/out mode & last if app mode
2) Input (get) pointer → always at zero position

seekg() :- Moves get pointer to specified location

seekp() :- Moves put pointer to specified location.

tellg() :- Gives the current position of get pointer.

tellp() :- tells current position of put pointer

seekg (offset, ref position)
seekp (" , " ")
ios :: beg
ios :: cur } Ref. positions
ios :: end

Standard Template Library

Collection of Generic function & Generic classes.

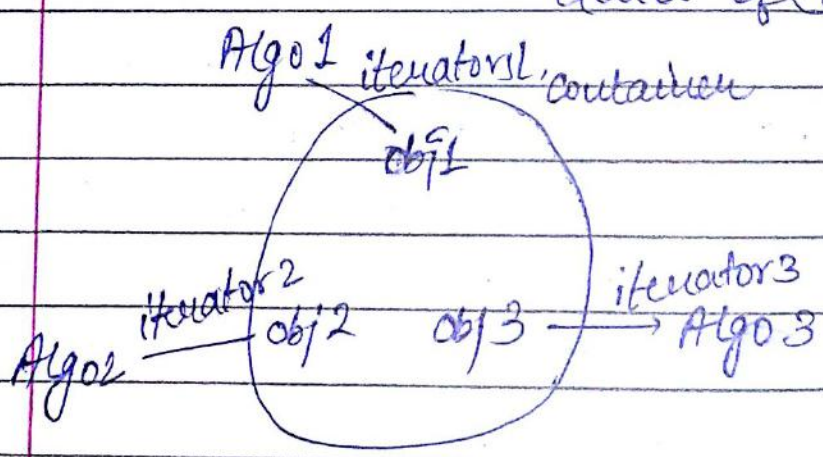
⇒ using namespace std

STL component defined in namespace

STL Components

3 main component-

① Containers → obj that store data
② Algorithms → use to process data of ①
③ iterators → like ptr. used to traverse data of ①



Containers are implemented by template cl
Algo " " " " fu

CONTAINER

3 Categories

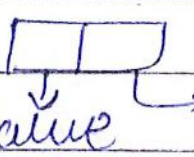
i) Sequence container → vector/deque/
ii) Associative container → set/multiset/
iii) derived container → stack/queue/Prior

Sequenc
Elemen
Heade
<vecto
<list
<deq
Associa
Not
direc
Ite
set/M
Map/
<Map>

Sequence Container
Elements stored in particular sequence.
Headers → <vector> <list>
<deque>
<vector> → dynamic array
Insert/delete at back
direct access to any element
iterator → Random Access
<list> → Linear list
ins/del any where
iterator → bidirectional
<deque> → iterator → Random access

Associative Container
Not stored sequential
direct access to data using keys
Iterator → Bidirectional
set/Multiset → stores No. of item.
↳ allows duplicacy <set>
↳ No "

Map/Multimap → data stored in pair



<Map> → Allows 1 key for 1 value
" 2 or more " " "

Derived container (container Adapter)
No. iterator supported

<stack> <queue>
↓ ↓
stack queue/priority queue.

ALGORITHMS.

functions which are used to process content of container.
STL provides approx. 60 Algorithm

Header → <Algorithm>
Categories.
① ~~Retrieve~~ Retrieve or non mutating Algo
② Mutating ④ Set
③ Sorting ⑤ Relational

ITERATOR

Just like pointer.