(.mif)

Memory initialization File ⇒ An ASCII text file (with extension .mif) that specifies the initial content of a memory block (CAM, RAM or ROM) that is, the initial values for each address. This file is used during Project compilation and/OR simulation.

A MIF (memory initialization file) is used as an I/P file for memory initialization in the compiler & simulator. You can also use a hexadecimal file to provide memory. A MIF contains the initial values for each address in the memory. A separate file is required for each memory block.

EX: DEPTH=32; // depth is no. of address
WIDTH=14; // width is no. of bits of data per
depth & width should be entered as decimal number

```
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
BEGIN
[0...F] : 3FFF; // Range every address from 0 to F
6:F; // single address 6=F
8: FE5; // address[8]=F address[9]=E address[A]=5
END
```

* If multiple values are specified for the same address only the last value is used.
* You can create a MIF by using the memory editor or the in system memory content editor

---

function call operator () ⇒ The function call operator () can be overloaded for objects of class type. When you overload (), you are not creating a new way to call a function.

Rather you are creating an operator function that can be passed an arbitrary number of parameters.

Ex:
```
class distance
{
int feet, inches;
public:
distance()
{ feet=0;
  inches=0;
}
distance(int f, int i)
{
feet=f; inches=i;
}
distance operator()(int a, int b, int c) //overloaded function call
{
Distance D;
D.feet=a+b+c;
D.inches=b+c+c;
return D;
}
void display()
{cout<<"feet"<<feet<<"inches"<<inches;
}
};
```

```
void main()
{distance D1(10,11), D2;
cout<<"First distance";
D1.display();
D2=D1(10,10,10);
cout<<"second distance";
D2.display();
return 0;
}
```

o/p
First F=10, inches=11
Second feet=30
inches=120

Type conversion ⇒ Type conversion, type casting & type coercion are different ways of changing an entity of one data type into another data type.

The two important way for type conversion is
- implicitly – automatic by compiler
- Explicitly – By user

Explicitly :- F=10;
float(F)
o/p = 10.00

t = 390.8
int(t).
= 390

String class ⇒
C++ has in it's definition a way to represent sequence of characters as an object of class. This class is called std:: string.
string class stores the characters as a sequence of bytes with a functionality of allowing access to single byte character.

Operation on string class ⇒

⇒ input function ⇒

* getline() ⇒ This function is used to store a streams of characters as entered by the user in the object memory.
* push_back() ⇒ This function is used to input a character at the end of string.
* pop_back() ⇒ This function is used to delete the last character from the string

⇒ Capacity function ⇒

* capacity() ⇒

Unit - IV

[container classes]

Container class ⇒

It is possible in c++ to declare a class within another class. A class declared as a member of another class is called as a nested class.

If an object of a class is declared as a member of another class. it is called as a container class.

In object oriented programming containership occurs when one class contains the objects of another class.

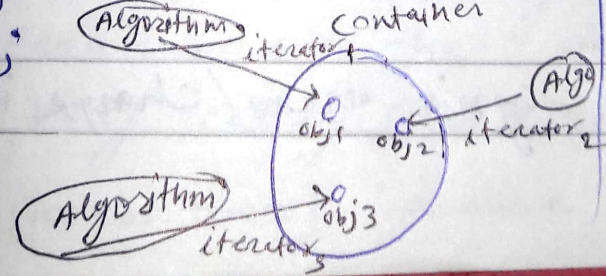
Types of relationship ⇒ "has-a relationship"

Ex:- Array, linked list, stack, queue

Syntax :-

```
class <classname-1>
  {
  ==
  };
class <classname-2>
  {
  ==
  };
class <classname-3>
  {
  classname-1 obj1;
  classname-2 obj2;
};
```

container class



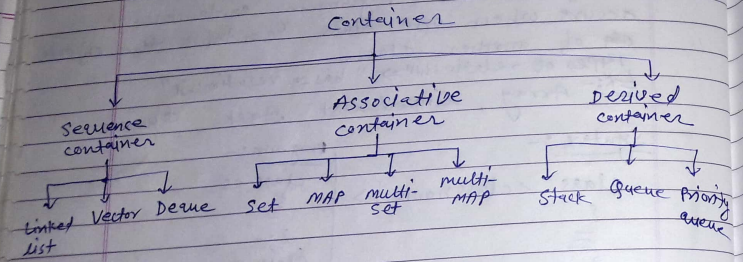
Program :-

```
class cs
  {
  ==
  };
class IT
  {
  ==
  };
class student // container class
  {
  cs obj1;
  IT obj2;
};
void main()
{
  cs obj1, IT obj2;
cout<<"Show containership";
cout<<"All object of class";
}
```

Container defines a way by which data is organised in memory.

Container ⇒ Object that holds data of some type is known as container. Each container class defines a set of functions. These functions can be used to manipulate contents of container class.

Types of container ⇒ There are following types of container



* **Sequence container ⇒** The container which stores data elements in a linear sequence such that each element is related to other elements by its position along a line is known as sequence container.

Types:-
There are following types of sequence container:-
* Linked list
* Vector (like dynic array / change in size)
* Deque

* **Associative container ⇒** The container which stores data elements in a structure called a tree is known as associative container. This type of container facilitates fast execution of operation like
⇒ searching an element (searching operation)
⇒ inserting an element (insertion operation)
⇒ Deletion of an element (deletion operation)

Types:- There are following types of associative container
* set, * MAP, * multiset, * multiMAP

* **Derived container ⇒** The container which does not support iterative operations & can not be used for data manipulation is known as derived container. These containers can be derived from sequence containers.

Types:- There are following types of derived containers.
* Stack, * queue, * priority-queue

List container ⇒ Array & vector are contiguous containers i.e. (that is) they store their data on continuous memory, thus the insertion operation at the middle of vector / array is very costly (in terms of number of operations & process time) because we have to shift all

the elements, linked list overcome this problem.
Linked list can be implemented by using the list container.

Syntax:-
```
#include <iostream>
#include <list>
Void main()
{
std:: list<int> l; // create a new empty linked list l
}
```
it's similar to Array & vector
std:: list <int> l{1,2,3};

operations:- Insertion, deletion, etc. empty, size, front & back, swap, reverse sort, merge, etc.

Vector container:-

An array works fine when we have to implement sequential data structures like arrays except it is static. we have define it's maximum size during it's initialization & it can not contain elements greater than it's maximum size.
Now suppose if during the program execution we have to store some elements more that it's size we do not know the upper bound of the number of elements there are high changes of occurrence of index out of

Heterogeneous container:- It contains different types of elements. It is not a natural or built in feature.
Homogeneous container:- It contains same types of elements

bound. error occures.
Solution of the above problem is dynamic arrays They have dynamic size. i.e. their size can change during runtime. container library provides vector container to replicate dynamic arrays

Syntax:
```
std:: vector <int> my_vector;
```

Set container :-

sets are container that store unique elements following a specific order. set containers are generally slower than unordered set containers to access individual elements by their key, but they allow the direct iteration on subsets based on their order.
sets are typically implemented as binary search trees.

Maps Container ⇒

Maps are used to replicate associative arrays. Maps contain sorted key values pair, in which each key is unique & cannot be changed & it can be inserted or deleted but can not be altered.
Ex:- A map of students where roll no. is the key & name is the value can represented graphically.

Syntax:-
```
map <int, int> m{ {1,2}, {2,3}, {3,4} };
```

Difference b/w Containership Vs Inheritance.

Containership	Inheritance
* Containership means the use of an object of a class as a member of another class	* Inheritance means deriving a new class from the old one by which objects of one class aquires the properties of another class.
* It represents "has-a" relationship	* It represents "is-a" relationship
* It is used when we need to represent a property by a variety of types	* It is used when we need to present override behavior & attributes.
* Containership relation is useful to define a set of functions that can be used to manipulate it's contents	* Inheritance relationship is useful when we have to reuse a class with adding some additional features without modifying it
* Container defines a way by which data is organised in memory	* Inheritance defines a way by which new class can be derived from old one.
* It contains objects that hold data of same type Ex:- Sequence container Associative container Derived container	* It contains base & derived class Ex:- single, multiple, multilevel, Hierarchical & Hybride inheritance.

Persistent object ⇒

An object in s/w takes up some amount of space & exists for a particular amount of time. Persistence is the property of an object through which it continues to exist after it's creator ceases to exist & it's location moves from the address space in which it was created. It deals with more than just life time of data.

Features of persistent object:-

* persistent is useful feature for object models.
* It gives rise to object oriented database.
* It provides an object oriented skin, over a relationship database
* It maintains the integrity of a database.

Template class ⇒

Class templates are generally used to implement containers.

Templates are the foundation of generic programming which involves writting code in a way that is independent of any particular type.

A template is a blue print or formula for creating a generic class or function.

The library containers like iterators & algorithms are examples of generic programming & have been developed using template concept.

Function templates ⇒
Function templates are special functions that can operate with generic types. This allow us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type

Syntax:- template < class mytype >
mytype GetMax (mytype a, mytype b)
{
return (a>b ? a:b);
}

Ex:- int x,y;
GetMax <int> (x,y);

```
template <class T>
T GetMax (T a, T b)
{
 T result;
 result = (a>b) ? a:b;
 return result;
}
```

```
Void main ()
{
int i=5, j=6, k;
k= GetMax <int> (i,j);
cout << k;
}
```


class templates:-
We also have the possibility to write class templates.

Ex:-
```
template <class T>
class mypair
{
  T value[2];
public:
  mypair (T first, T second)
  {
    values[0] = first;
    values[1] = second;
  }
};
```

Ef:-
```
Template <class T>
class mypair
{
  T a, b;
Public:
  mypair (T first, T second)
  {
  a = first , b = second; }
    T getmax ();
};
```

```
template <class T>
T mypair <T> ::getmax()
{
  T retval;
retval = a>b ? a:b;
return retval;
}
Void main ()
{
mypair <int> myobject (m,n);
cout << myobject.getmax();
}
```

STL first unit में पढ़ चुके हैं Iterator STL का component है।

Iterators ⇒ Iterators are used to point at the memory address of STL (standard template library) containers
They are primarily used in sequence of numbers, characters & symbols etc.

They reduce the complexity & Execution time of program

operations of iterators:-

* **begin()** ⇒ This function is used to return the beginning position of the container.

* **End()**:- This function is used to return the end position of the container.

* **Advance()**:- This function is used to increment the iterator position till the specified number mentioned in it's arguments
Ex:- advance (ptr, 3);

* **next()**:- This function returns the new iterator that the iterator would point after advancing the ~~points~~ positions mensions in it's arguments.

* **Prev()**:- This function returns the new iterator that the iterator would point after decrementing the positions mentioned in it's arguments.

Stream is a technique for transferring data

* **insert()** ⇒ This function is used to insert the elements an any position in the container.
Ex:

Stream ⇒ Stream is a standard interface for character input/output (I/O) devices. Stream provides an effective environment for I/O devices. Stream is a sequence of data elements made available overtime.
We use cin & cout with the operators >> & << respectively for input & output operations in c++. These keywords control the how a program takes data as a input & how it generate (display) the processed data as a output. in the desired form.

A set of I/O functions & operations provided by c++ compiler.

We need some memory devices such as floppy disk or Hard disk to store a large amount of data. The data can be stored in these storage devices in the form of files.

"A file is a collection of related data stored in a particular memory address on the disk."
It may be set of characters or text or programs.

Programs can be designed to perform the read/write operations on these files.

These are two following data communication techniques in file.

- Data transfer b/w consol unit & program
- Data transfer b/w programs & disk file.

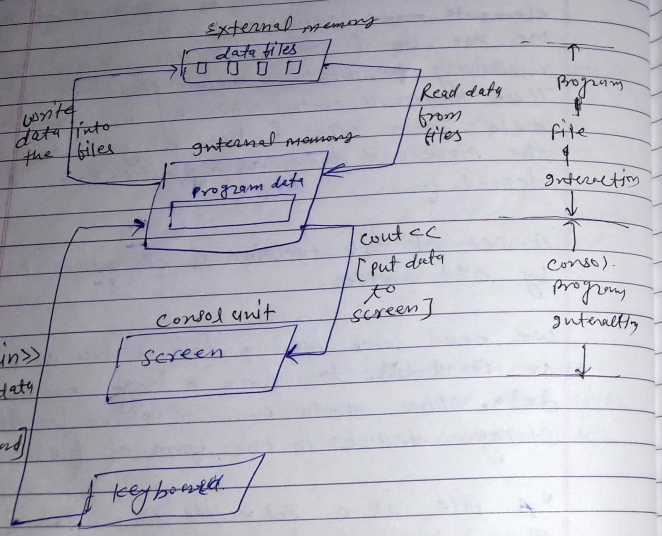


diagram:- consol program-file interaction.

Methods for storing & retrieving data from files ⇒

Data stream can act as:-

* A source form which the i/p data can be obtained
* A destination to which the o/p data can be sent.

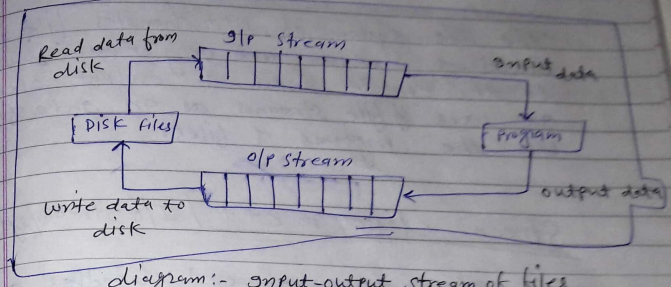


diagram:- input-output stream of files

Types of Streams ⇒

- input streams
- output streams

* Input Streams ⇒

The source stream that provides data to the program is called the input stream. ~~It read data from the files~~. A program extracts the bytes from an input stream. It read data from the files.

Output stream:-
The destination stream that receives output from the program is called the output stream.
A program inserts bytes into an output stream. It writes data to the file.

The I/O system of c++ handles file operations which are very much similar to the console input & output operations. It uses file streams as an interface b/w the programs & files.

Stream classes in c++ [I/O stream/system]

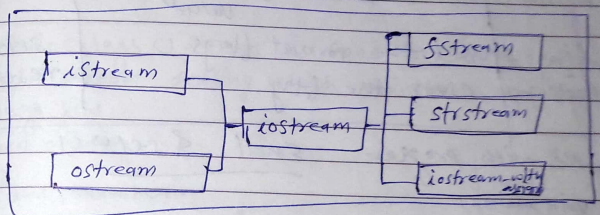
The I/O system of c++ handles a class hierarchy used to defines the various file methods. These classes are known as stream classes.
There are following stream classes includes:-

- fstream
- ifstream
- ofstream

These classes are declared in header file iostream.

*** io stream.h Header files ⇒**
Declares the basic c++ streams (I/O) routines. It provides a new method to perform the input & output operations. This method is known as iostream method. It is a standard header file which contains a set of small & specific general purpose functions to handle input from the keyboard & data output on to the screen display.
Function prototypes in iostream.h header files:-

- ios
- istream
- ostream
- iostream.h
- istream_with assign
- ostream_withassign
- iostream_with assign
- streambuf



classes:- ios, iostream, iostream_with assign, istream, istream_withassign, ostream, ostream_withassign, streambuf

Manipulators (formatted I/O operations) ⇒ special stream functions which are used to change certain characteristics of the input & output. These functions are provided by the header file "iomanip.h"
These functions help to manipulate the output formats & change format flags & their values for a stream.

We use iomanip.h header file for I/O manipulators

Name of manipulator	function	equivalent function
1) endl	insert new line & flush stream	"\n"
2) setw()	set the field width	width()
3) setfill()	set the fill character	fill()
4) setPrecision()	set the precision for floating point value	width()
5) setiosflags()	set the format flags	setf()
6) resetiosflags()	clear the flag	resetf()

Example with program endl & setw()

```
Void main()
{
int i=100, j=200;
cout << setw(5) << i << setw(5) << j << endl;
cout << setw(10) << i << setw(10) << j << endl;
}
```


setfill() ⇒ This manipulator is used to specify a different character to fill the unused field width of the value

```
Void main()
{
int i=100, j=200;
cout << setfill('*');
cout << setw(5) << i << setw(5) << j << endl;
cout << setw(9) << i << setw(10) << j << endl;
}
```

o/p

```
**100***200
*******100**********200
```

setPrecision() ⇒ This manipulator is used to control the number of digits of an output stream display of a floating point value.

```
Void main()
{
int i=2, j=3;
float k= i/j;
cout << setPrecision(1) << k << endl;
cout << setPrecision(2) << k << endl;
}
```

o/p

```
0.7
0.67
```

setiosflags() ⇒ This is used to control different I/O stream setting which maintains a collection of flags bits.

```
void main()
{
int number;
cout << "enter number";
cin >> number;
cout << setiosflags(ios::dec) << number;
cout << setiosflags(ios::hex) << number;
cout << setiosflags(ios::oct) << number;
}
```

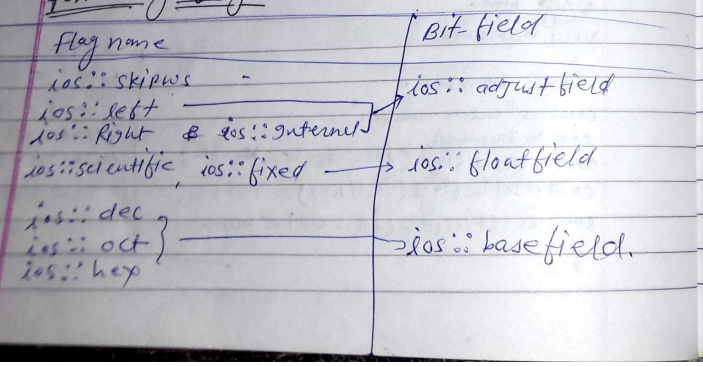
o/p

```
Enter number=15
decimal = 15
Hexadecimal = f
Octal no = 17
```

I/O stream flags ⇒

Flag name	Functionality
showpoint	Show the decimal point for all floating point no.
fixed	use floating notations
showpos	show + to before positive integers
skipws	skip white space during input
unitbuf	flush all stream after insertion
stdio	flush stdout & stderr after insertion
scientific	use E for floating notation.
internal	Pad after sign or base indicator
right	left justification of output
showbase	show base indicator on output for octal & hexadecimal number.
uppercase	show uppercase letters for hexadecimal number
dec	show integers in decimal format
oct	show integers in octal format
hex	show integers in hexadecimal format

Formatting flags ⇒



Program ⇒

```
Void main()
{
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout.precision(4);
cout<<.setf(ios::fixed, ios::floatfield);
cout.setf(ios::internal, ios::adjustfield);
cout.width(15);
cout<<985.4<<endl;
}
```

O/P

\+ 985.4

पढ़ने File पढ़े Next Page पर (131 पर)

File mode ⇒

The file mode specifies how a file would be used. It specifies the purpose for which the file is opened. we can specify whether we wish to read / write / append to a file.

Syntax ⇒

<Stream-object>.open("filename", filemode);

The function open() with two argument specifies

* first argument = file name parameter
* second argument = file mode parameter

P.T.

File mode parameter	Purpose
ios::in	open a file for reading only
ios::out	open a file for writing only
ios::app	Append at the end of a file
ios::ate	Go to the end of a file upon opening instead of beginning
ios::trunk	If the file exists, it is truncated all data is erased before writing or reading
ios::nocreate	open a file if it doesnot exist
ios::noreplace	open a file if it already exist
ios::binary	open a file binary file(by default text file)

Function Prototypes

File name (Class)	Default mode
ifstream	ios::in
ofstream	ios::out
Fstream	No default mode define

Values (in ~~Turbo~~ Borland c++)

Value	meaning
0	Default
1	Read only file
2	Hidden file
4	System file
8	Archive file (भंडारगार की किसी एक file के अलावा बहुत फाईलस को रखने की तकनीकी)

File ⇒ File is a collection of related data stored in Secondary memory.

To store the data on secondary memory i.e (such as) Hard disk, we need files.

Data can be stored in the form of files.

It (File) may be one of the following forms:-

- set of characters
- text
- Program

[when we work with a program, all data can be stored using different concepts. But when we turn off the system & restart it the next time, our all stored data will get lost. Reason behind this is that, that data was stored in the ~~memory~~ Primary memory which is volatile in nature. Contents of the primary memory exist till the power is on. If we want ~~be stored~~ that our data must be safe after power is off & stored it permanently, all data must be stored in a secondary memory.]

operation on file (File handling) ⇒ Read & write operation

Some programs can be designed to read & write data from these files

Types of files in c++ ⇒

There are mainly two types of files used in c++.

- sequential files
- Random access files.

* Sequential file ⇒ In this ~~file~~ type of files, we can store the data & Read the data in a sequential manner.
* Random access files ⇒ In this type of files, we can store the data & Read the data in randomly manner.

Working with files ⇒

To work with files, we need some file operations using c++ stream.

- → Creating an input file to be read by program (istream.h)
- → creating an output file to store the processed data (ostream.h)

Steps involved in opening & closing files ⇒

* Declare a stream
* create a file using an appropriate file name, on the disk
* Associate a file with stream
* open the file to get the file pointer.
* Process the file
* check for error while processing
* close the file after using it

Syntax:- for open a file.

```
Stream_object.open("file name");
```

Syntax for close a file

```
Stream_object.close();
```

Program →

```
#include <fstream.h>
gnt main()
{
ofstream out;
out.open("my_file.doc"); // opening a word file
cout<<"c++ is better than c";
return 0;
}
```

o/p A word file created consisting statement "c++ is better than c";

Program (without using file mode)

```
#include <fstream.h>
gnt main()
{
 of stream out;
out.open("myfile.doc");
out<<"c++ is better than c"<<endl;
out<<"c++ is the superset of c";
out.close();
 if stream in;
in.open("myfile.doc");
const int size=50;
char ch[size];
while(in)
 {
  in.getline(ch,size);
   cout<<ch<<endl;
 }
}
```

o/p

c++ is better than c
c++ is superset of c

if we use ios::trunc mode then four couple of statements are deleted only one time show

Program (using ios::app mode)

```
#include <fstream.h>
void main()
{
ofstream out;
out.open("myfile.doc", ios::app);
out<<" c++ is better than c"<<endl;
out<<"c++ is superset of c";
out.close();
ifstream in;
in.open("myfile.doc");
const int size=50;
char ch[size];
while(in)
{
in.getline(ch, size);
cout<<ch<<endl;
}
}
```

O/P

if we execute this program 5 times

c++ is better than c
c++ is superset c

5 times ~~rep~~ repeat this two msg

Stream member functions ⇒

To perform the input & output operations on disk file, c++ provides the file stream classes which support a number of member functions.

Member function	Purpose to design
get()	for handling a single character at a time
put()	for handling a single character at a time
read()	To read blocks of binary data
write()	To write blocks of binary data.

* get() ⇒

The get() member function is used to read an alphanumeric character for a specified file. it reads a single character from the associated stream.

Program

```
#include <fstream.h>
void main()
{
ifstream in;
char ch;
in.open("myfile.doc");
while((!in.eof()))
 {
  ch=in.get();
 }
}
```

end of file

* put() ⇒

The put() member function is used to write an alphanumeric character for a specified file. It writes a single character to the associated stream.

Program ⇒

```
#include <fstream.h>
void main()
 {
  ofstream out;
  char ch;
out.open("myfile.doc");
while((!out.eof()))
 {
  ch=out.get();
  out.put(ch);
 }
}
```

* Read() ⇒
The read() member function is used to read sizeof(object) bytes from an associated stream & put them in the buffer pointed by object

```
#include<fstream.h>
class cs
{
==
};
void main()
{
cs obj;
ifstream in;
in.read((char*) &obj, sizeof(obj));
==
}
```

* write() ⇒
The write() member function is used to write size of (object) bytes to the associated stream from the buffer pointed by object.

```
#include<iostream.h>
class cs
{
==
};
int main()
{
cs obj;
ofstream out;
out.write((char*) &obj, sizeof(obj));
==
return 0;
}
```

Error handling member function ⇒

- eof()
- fail()
- bad()
- good()

member function	State of I/O system
eof()	1 when end of file is encountered, otherwise 0
fail()	1 when an I/O error occurs, otherwise 0
bad()	1 when an uncoverable error occurs, otherwise 0
good()	1 when no error occurs, return 0 when no further operations performed

* eof() ⇒ This function is used to check whether a file pointer has reached the end of a file character or not

```
#include<fstream.h>
int main()
{
ifstream in;
in.open("myfile.doc");
while(!in.eof())
{
==
}
}
```

* fail() ⇒ This function is used to check whether a file has been opened for input/output successfully or to checks whether a file for input/output operation has failed.

```
#include <fstream.h>
int main()
{
ifstream in;
in.open("myfile.doc");
while(!in.fail())
{
cout<<"could not open a file";
continue;
}
}
```

* bad() ⇒ This function bad() is used to check whether any invalid file operation has been attempted or there is an unrecoverable error.

```
#include<fstream.h>
void main()
{
ifstream in;
in.open("myfile.doc");
while(!in.bad())
{
cout<<"open failure";
exit(1);
}
}
```

* good() ⇒ This function is used to check whether all previous file operation has been successful or not.

```
#include<fstream.h>
#include<stdlib.h>
void main()
{
ifstream in;
in.open("myfile.doc");
while(!in.good())
{
}
}
```

Object oriented programming languages ⇒

* OOP languages are the create complex sys.
* It make easier to implement an object oriented system design.
* It gives less complexity & increased maintainability in designing.

OOPs support following features ⇒

* object, * class, * Data abstraction
* Inheritance, * Polymorphism, * ~~Dat~~
* Dynamic binding * message passing

Various OOP languages ⇒

* c++, * Java, * simula, * small talk
* modula-3, * self, * Eiffel

* c++ ⇒

Unit-II

DATE PAGE 21

Modeling :-

A model is an abstraction of some things for the purpose of understanding it before building it. because a model omits non-essential details. It is easier to manipulate than the original entity.

importance of modeling :-

* testing a physical entity before building it.
* communication with customer.
* visualization
* Reduction of complexity.

OOM (Object oriented methodology) :-

we represent a methodology for object-oriented development & a graphical notation for representing object oriented concepts.

The methodology consists of building a model of an application domain & then adding implementation details to it during the system design. we call this approach OMT (object modeling technique)

The methodology has the following stages :-

OMT * Analysis, system design, object design, Implementation

* Analysis :- Starting from a statement of the problem The analyst builds a model of the real world situations showing it's important properties.
* System design :- The system designer makes high-level decisions about overall architecture. During system design the target system is organised into subsystems based on both the analysis structure and proposed architecture.

The system designer must decide what performance characteristics to optimize, choose a strategy of attacking the problem & make tentative resource allocations.

* object design:- The object designer builds a design model based on the analysis model but containing implementation details. The focus of object design is the data structures & algorithms needed to implement each class.
* Implementation:- During implementation, it is important to follow good s/w engineering practice. The implemented system remains flexible & extensible.

OMT (Object Modeling technique):-

Object oriented paradigm modeling is known as object modeling technique (OMT).
OMT consists of three models.
- object model
- Dynamic model
- functional model.

Method look-up ⇒
Method look-up is a procedure or method to connect the function call with the function code

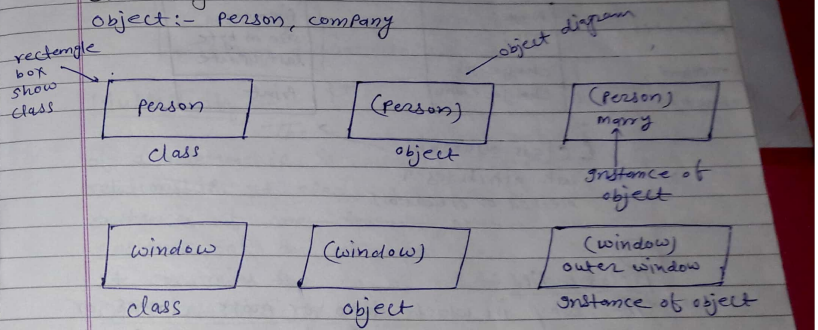
Meta data:-
Data about data is called meta data.
Ex:- define a class is a meta data

Meta class:- A meta class is a class of a class. The object of this class can themselves act as classes. So a user can add or remove attributes at run time.

Object:-
Object are the real world entity which contains three properties are identification, behavior & state.
Identity means that data is quantized into discrete, distinguishable to other data.

Class:-
An object class describes a group of objects with similar properties (attributes) common behavior (operation), common relationship to other objects & common semantics.

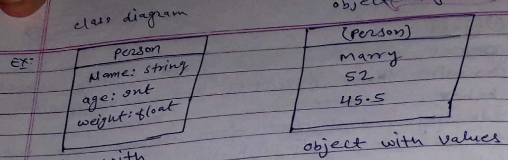
Class diagram:-
Object:- person, company



Attribute:-
An attribute is a data value held by the objects in a class
Ex:- Name, age, weight etc are attributes of person object.

class diagram   object diagram

Ex:

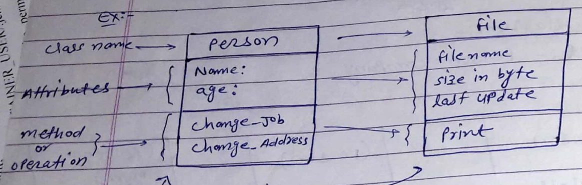


class with attributes   object with values

Operations & method :-

An operation is a function or transformation that may be applied to or by objects in a class.

Ex:-



Class diagram with attributes & method or operation

Relationships :-

The way in which two or more classes or objects are connected or the state of being connected is called relationship.

Types of relationship [Relationship b/w classes]

Inheritance, Association, aggregation, Generalization, specialization, multiplicity, attribute, ~~navigation~~ navigability, composition.

Association

An association defines a relationship b/w two entity objects based on common attributes.

In object oriented programming, association defines a relationship between classes of objects that allows one object instance to another object instance.

There are following types of association :-

- one to one association
- one to many association
- Many to one association
- Many to many association
- Ternary association.

Main three association used in object oriented design.

- one to one association
- Many to many association
- Ternary association.

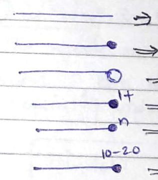
} diagram आगे 33 page पर

Multiplicity :-

The no. of objects involved on both side of a relationship is called multiplicity of the relationship. It specify (specifies) how many instences of one class may related to a single instance of an associated class.

It constrains the no. of related objects.

OMT multiplicity symbols :-



⇒ (1 to 1)
⇒ (1 to many)
⇒ (1 to 0 or 1)
⇒ (1 to 1 or many)
⇒ (1 to exactly n)
⇒ (1 to min & max)

Inheritance/generalization/specialization all same idea.

UML multiplicity symbols :-

0 ⟹ (none)
1 ⟹ (one)
* ⟹ (some (0 to ∞))
0---1 ⟹ (none of one)
1---* ⟹ (one or more)
3---5 ⟹ (Three, four or five)
6, 9 ⟹ (six or nine)

Generalization :-

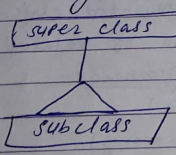
Generalization "is a relationship" defined at the class level, not at the object level. It is the relationship between a class & one or more refined version of it. It consists of two classes.

- Super class
- Sub class

* Super class :- The class being refined is called as a super class
* Sub class :- Each refined version from a super class is called as sub class.

generalization symbol is 

Ex :-



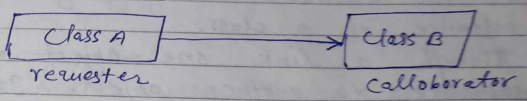
Specialization :-

Specialization refers to the fact that the sub class refines or specialize the super class. Super class represents generalized abstractions & sub class represent specializations. In which fields & methods from the super class are added hidden or modified.

Navigability :-

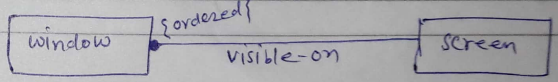
More often association go only one way but if ~~support~~ suppose both classes are request to each other then association can go both ways at onces. Thus the direction of association is called ~~Nesting~~ Navigability.

Ex :- class A requests class B to help it out but class B is not asking class A for anything then



Ordering ⇒ {Role name के बाद पढ़े} Page 28 पर

The objects on the "many" side of an association have no explicit order & can be regarded as a set. Sometimes the objects are explicitly ordered.



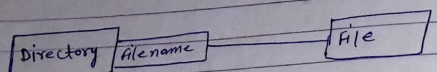
PT

Qualified Association :-

A qualifier distinguishes objects on the many side of an association.

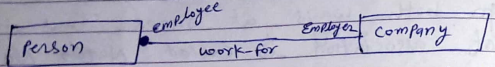
Ex:- ATM card.

one to many associations



Role Name :-

The role name describes the role that a class in the association plays from the point view of the other class.



Employee :- Joe

Employer :- Simplex

Link :-

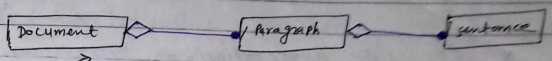
A link shows a relationship b/w two or more objects. It is an instance of an association as an object is an instance of a class.

Through a link one object may navigate to another object. It is a physical or conceptual connection b/w objects.

"A link is defined as a tuple. i.e. (that an ordered list of object instances.

Cardinality :-

Multiplicity of an association denotes cardinality.



Aggregation :-

Aggregation means one object contains other objects. Aggregation denotes a whole or part of relationship.

Aggregation is a strong form of association.

Part = attribute

Whole = aggregate.

* Attributes- Properties of object is called attribute.
* Entity :- Set of objects is called entity.

Types of aggregation :-

There are following types aggregation

- Fixed aggregation
- Variable aggregation
- Recursive aggregation

* Fixed aggregation :-

The particular numbers & types of the components are predefined, is called fixed aggregation

Ex:- car

car :- one engine, four wheel & one stering

P.T.O.

* Variable aggregation :-
The number of levels of aggregation is fixed but the number of parts may be vary (change)
Ex:- Train
Train:- No. of compartment (Bed) may be change.

* Recursive aggregation :-
The object contains components of it's own type.

Properties of aggregation :-

* Transitivity property :-
If A is a part of B & B is a part of C then A is part of C
$A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$

* Antisymmetry property :-
If A is a part of B then B is not part of A
$A \rightarrow B$ then $B \not\rightarrow A$

* Propagation property :- [under stood]
The environment of the part is the same as that of the assembly
Ex:- If car is in garage then it's part is also in garage it's under stood.

Difference b/w :-

* Aggregation Vs Generalization :-

Aggregation ⇒
* Aggregation means one object contains other objects.
* A special form of association that specifies a whole part relationship b/w the aggregation (whole) & the component part.
* It is a part of relationship
* It is sometimes called as an relationship
* Aggregation relates instances.
* Two distinguish objects are involved one of them is part of other.

Generalization ⇒
* Generalization is the relationship b/w a class & one or more refined version of it.
* A taxonomic relationship b/w a more general & more specific element
* It is a kind of relationship.
* it is sometimes called as an relationship
* ~~generation~~ Generalization relates class.
* Single object involves an object simultaneously as ~~3rd~~ instance of the super class as well as subclass.

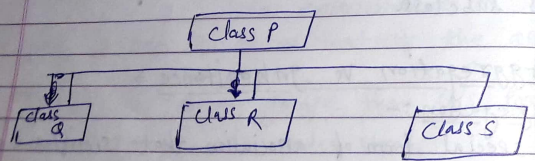
* Aggregation Vs Inheritance ⇒
Aggregation :-
* A special form of association that specifies a whole (Part) relationship b/w the aggregation (whole) & the component part. [Point-2]

* Example of aggregations are :-
  A computer whose components are screen, keyboard, Mouse & processor.
* It is a part of relationship [Point-3]
* It supports the concept of association
* Types of aggregation :-
  Fixed, variable & recursive aggregation.

Inheritance :-
* It is a process by which object of one class acquire the properties of objects of another class
* Example of inheritance :-
  A child inherits properties of their parents
* It is a classification hierarchy relationship
* It supports the concept of reusability.
* Types of inheritance :-
  Single, multiple, multilevel, hierarchical & hybrid

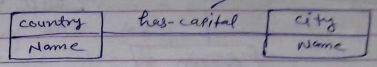
Composition :-

Stronger form of aggregation in which the parts are necessary to the whole and are more permanently bound to it is called composition.

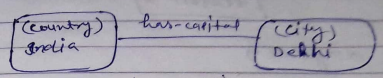


Types of association :-
* one to one association :-

class diagram

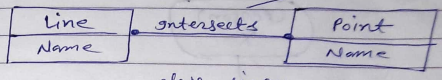


Ex :-

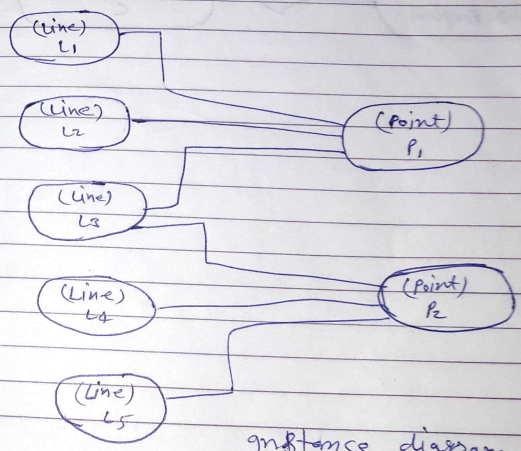


Instance diagram

* Many to Many associations :-



class diagram



Instance diagram

Delegation :-

Delegation is the assignment of responsibility or authority to another person (normally from a manager to a subordinate) to carry out specific activities. It is one of the core concepts of management leadership.