Rs = 90/=

C++

NEW NOTES

BY

Mr.Sathish Gupta

SRI RAGHAVENDRA XEROX

Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

S.U

C++

Q.1 what is C++ ?
C++ is an object oriented Programming Language.

2. what is language ?
(A) Language is a software.
↳ Language is a programming tool.
↳ Language provide some set of Predefined instructions to develop Programs.

3. what is Program ?
(A) Program is a collection of data and instructions

4. what is programming ?
(A) It is a Process of organizing of data and instructions according to a given Problem.
This organization is done by Following certain rules and regulations. These rules and regulations are called programming concepts or Programming Principals.
↳ Programming elements are
1. Data
2. Instructions

5. what is object oriented ?
A:- Object oriented is a programming Principals or concepts.

6 when developed C++ ?
A: C++ was developed in the year 1983.

NOTE- C++ is a general purpose P.L.
Means with help of this language we can develop any softwares.

7. Who developed C++?
(A) C++ was developed by Bjourne Stroudstroup.

8. Why developed C++?
(A) To overcome drawbacks in procedural oriented ~~languages~~ Programming.

APPLICATIONS OF 'C++'
C++ is general Purpose P.L

1. Application softwares
   - word processors
     - ↳ MS word
   - spreadsheets
     - ↳ MS excel
   - Database
     - ↳ oracle

2. System Softwares
   - operating systems
     - ↳ windows, Linux, unix
   - Device Drivers
     - ↳ printer Drivers
     - ↳ mouse drivers
   - Network Protocals
   - virus
   - Antivirus

Programming concepts :-

1. Monolithic programming
2. Procedural // (POP)
3. Modural // (MOP)
4. Structured // (SOP)
5. object oriented // (OOP)
6. Aspect // // (AOP)

1. what is Monolithic programming? (unstructed)

A:- ~~we call~~

↳ Assembley language and basic P.L are called monolithic Programming language.

↳ In this approach Programming instructions organized in sequential orders.

↳ A sequence is set of instructions used to solve a given problem

Disadvantages:-

↳ Code Redundency

↳ Size of program increased this leads to more space and less efficiency.

2. what is Procedural Programming? (POP)

A: Cobol and Fortran are called Procedural P.L.

In this approach instructions are organized by dividing into small Programs, called subroutines. these subroutines can be procedure or Functions.

Advantages

1. Reusability: write code once and use it more than once.
2. Modularity: Dividing instructions according to their operations.
3. Readebility: easy to understand
4. Efficiency: Occupies less space

* Disadvantages:-

1. Data is not secured: Data is declared as local or global. it is not secured from unrelated operations.
2. Debugging is complex: In a large program identify which operation perform on what data is complex.
3. There is no Proper organization of data and instructions

3. what is modular programming?

A:- A module is a program

↳ In this approach an application contains one or more programs - Each program is divided into Sub-programs (Sub-routines)

↳ A program which contain 'main' is compiled as '.exe'.

↳ A program which doesn't have 'main' is compiled as 'DLL/.Lib'. This is reusable program.

4. What is Structured Programming?

A ↳ It is called Structed. P.L.

↳ characteristics of Structed Programming

Modular Programming is supported by Structed programming.

↳ Allows you to develop the user defined datatypes

↳ Exchanging data beth modules.

↳ Scope of data / variables

↳ Top-down approach

Disadvantages

5. Object-oriented programming :-

1. Encapsulation
2. Polymorphism
3. Inheritance
4. Abstraction
5. class
6. object

1. Encapsulation :-

Encapsulation is a process of grouping data and operations which operates on that data into single entity.

Or

It is a process of binding data with related operations.

Advantage :- 1. Data hiding : To preventing data access from other parts of program
2. Binding.

What is data hiding?

A Preventing data access from other parts of program / unrelated operations

Adv :- ↳ which allows to develop secure applications

What is binding?

↳ Linking or grouping data with related operations.

class :-

* what is a 'class'?

A: (i) class is a building block of a object oriented program.

(ii) class is a collection of data and instructions.

(iii) class is a collection of members. These members are 2 types
1. Data members
2. Members Functions

(iv) class is a datatype. It is user-defined datatype.

(v) class is a blueprint of object.

(vi) class define structure of object.

(vii) class contain metadata.

* what is object?

A: Object :- object is an instance of a class.

(ii) An instance is allocating memory for members of the class.

(iii) Object is a class variable.

* what is SBI of object?

SBI stands for State Behaviour and identity.

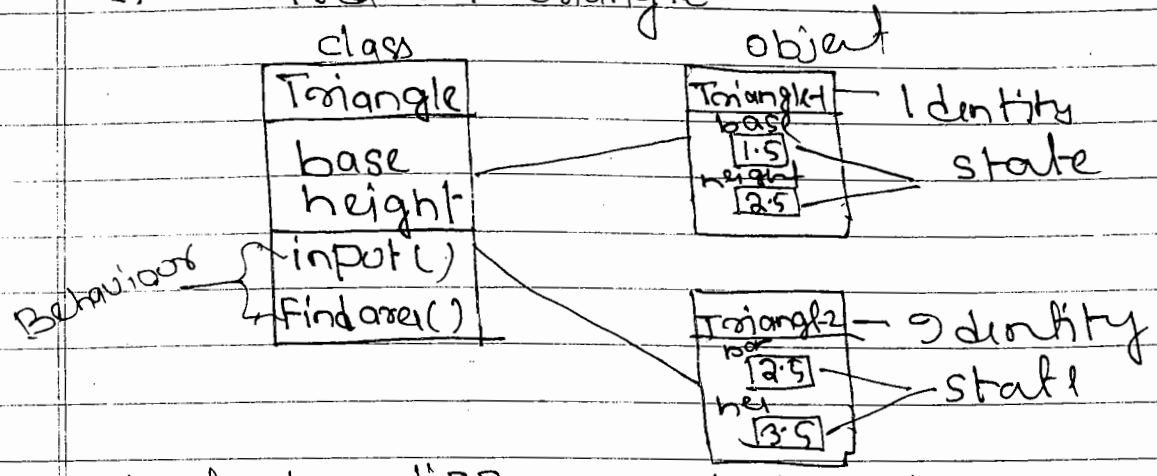
State - It is a value given to each attribute of an object

Behaviour :- An operation performed on an object is called behaviour.

Behaviour change the state of the object.

Identity :- Name given to object.

Ex Area of triangle



* what is difference bet^n class and object?

A:-

Class	Object
class is a logical entity.	Object is a physical entity.

Advantages of C++ :-

1. object oriented :-
   a. Modularity
   b. Reusability
   c. Readability
   d. Security
   e. extensibility
   F. Efficiency
2. Portable
3. Embeded

4. 'C' compatiable

C++ compilers:-

- C++0c
- C++0xc
- C++12 -> latest
  - -> Microsoft C++
  - -> Borland - C++
  - -> Turbo - C++
  - -> C++ on unix
  - -> C++ on Linux
  - -> C++ on solaris

Character set of C++:

C++ supports 128 characters, which is called standard character set.

A-Z (upper case)
a-z (lower case)
0-9 (Digits)

Special characters :-

, . / ; ' [ ] \ = -) (* & ^ % # !
~ < > ? : " { } | + _ .

Tokens of C++: A token is smallest individual unit within program.

Tokens are of following types

1. Keywords
2. Identifiers
3. constants
4. Datatypes
5. Operators

1. Keywords :- Keywords are some language related words, And each keywords having special meaning within Language.

(ii) Keyword is called one predefined instructions

(iii) The meaning of this keyword understand by compiler and it is ~~never~~ reserved. C++ supports 49 Key in which 32 from C, remaining belo. These are some new keywords added to C++ they are

bool	friend	namespace
class	Virtual	using
Public	inline	this
Private		new
Protected	catch	delete
operator	throw	

2. Identifiers :- i. Identifier is a user-defined word.

(ii) Identifier is a collection of alphabets, digits, and allows one special character and that special character is _ underscore.

iii) Identifier is a user-defined word used to identify programming constructs like variable, Function and class.

* Rules For Identifiers :-

1. Identifier must start with alphabet or '_'.

_rno ✓	RNO ✓
1rno ✗	Rno ✓
rno1 ✓	int ✗
rno_ ✓	INT ✓
*rno ✗	

Datatypes :- Datatypes reserves memory for data.

Datatypes

Simple D.t	size.	Derived D.t	user-defined	Empty D.t
1. int	2 bytes	1. array	1. Struct	1. Void
2. Float	4 bytes	2. Pointer	2. union	
3. double	8 bytes	3. reference. *	3. enum	
4. short int	2 bytes		4. class *	
5. Long int	4 bytes			
6. signed int	2 byte			
7. unsigned int	2 bytes			
8. long double	10 bytes			
9. char	1 byte			
10. signed char	1 byte			
11. unsigned char	1 byte			
12. bool *	(New) 1 byte d.t added			

Structure of C++ programming :-

1	Document section
2	Linkage section
3	global declration section
4	Main Program section
5th	global Defn section

Document Section :- It is information about program or program heading.

(ii) This information is used by other programmers to understand programs.

(iii) This information is given by using comments.

(iv) comments are ignored by compiler.

(v) C++ supports 2 types of comments
- Single lined comments //
- Multi lined comments /* */

LINKAGE SECTION :- i) C++ supports modular programming.

ii) This section is allowed to link one program with another program. (optional)

Global declaration Section :- (optional)

1. It is used to declare global variable, Functions, and datatypes.

Main program Section :- (optional)

1. Every executeable program must have main.
2. Keyword whose meaning is understand by linker.

Types of main()

```
void main()
{ }
void main(int arg, char arg)
{ }
```
main with argument

```
void main(int arg, char arg, char envir)
```

Global defination section:

1. writing the definations of classes and functions.

#include<iostream.h>
↳ search in include directory
#include"iostream.h" include path
↳ search in current directory

I/O operations in c++ :-

Q. what is input?
(i) Information is given to a program is called input.
(ii) Input data is Flow inside program.

Q what is output?
The information is given out by program
(ii) In output data is Flow outside the program.

↳ In c++ these input & output operations are done using Streams.

↳ A stream represents input source & output destination.
↳ A Stream is a Flow of data, which
↳ A Stream is a class which provid set of functions ~~which~~ to perform input and output operations.

↳ c++ provides 2 Pre-defined objects to perform input & output operations
1. Cout
2. Cin
These objects are declared inside in
iostream.h or iostream
↓ ↓
Turbo-c++ DEV c++

Q. what is a difference between #include<Filename> and #include"Filename"?
Ans #include<Filename>

#include "c:\ll\iostream.h"

#include<Filename>	#include("Filename"
This tell the compiler to look for given Filename inside include path.	This tells the compiler look for given File in source path.

Cout :- (i) Cout is pre-defined object a ostream class.

(ii) cout object is used to perform o/p operations.

(iii) cout represents console output.

(iv) Cout object is used to call the Function of ostream class to perform output operations.

(v) cout uses << (insertion operator) to write any type of data

Q. What is insertion operator ?

A:- It is an overloaded operator or special Function whose name operator symbol (<<).

↳ It is a member Function of ostream class

↳ This Function not required any formatting specifiers.

↳ One insertion operator insert only one value.

Syntax :-

```
cout << expression;
cout << constant;
cout << variable;
```

cout << 10 << 20 << 30; → 10, 20 30

(Evaluating ← , Insertion →)

cout << 10;	10
cout << 10+20;	30
cout << 20-10;	10
cout << 20, 30;	20
cout << 20.30;	20.30
cout << "c++";	c++ → string
cout << 'c';	c → single character
cout << true;	true

Q. How to insert/print multiple values using cout?

Ans:- By cascading method

Syntax:- cout << opr1 << opr2 << opr3 ... ;

Using multi-insertion operators with cout is called cascading.

Srikanth

Q. ~~what is~~

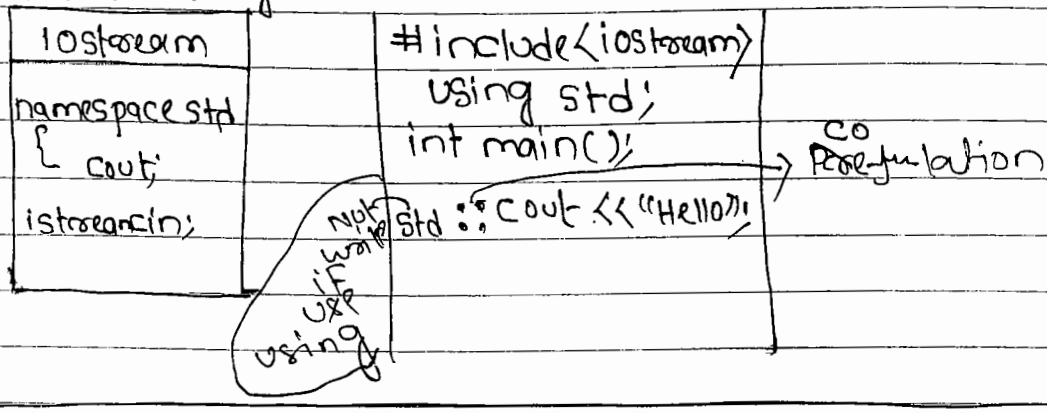
Ans.
```
#include<iostream>
#include<conio.h>
namespace void main()
{
    std::cout<<"welcome to c++";
    getch();
    return 0;
}
```

Namespace :- Namespace is a collections of classes, functions & variables.

Namespaces are of 2 types
1. Pre-defined namespaces
2. User-defined namespaces

1. Pre-defined namespaces are provided by C++ compiler. These are called libraries.

Only For understanding



Q How to use members of namespace?

A:-
1. namespace::membername
2. Using keyword
syntax using namespace.name;

Cin :- i. cin is an object of istream class.
ii. cin represents standard input device ?:e keyboard
iii. cin uses >> (Extraction operator) to read data from keyboard.
iv. Syntax:-

```
cin>>var;
cin>>var1>>var2>>var3;
```

endl - manipulator under std namespace
↳ doesn't take any memory

Pa.
```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    cout<<"C++";          | Cout<<"C++"<<endl;
    cout<<'A';            | Cout<<
    cout<<10;
    cout<<010;   → Octal
    cout<<0x10;  → Hexadecimal
    cout<<1.5;
    cout<<1.5f;
    cout<<true;
    cout<<False;
getch();
  return 0;
}
```

O/P- ~~C++A10~~1 C++A10816151510

Q. what is the difference betn endl and \n ?

Ans:-

endl	\n
It is a manipolator.	It is a single character constant.
(ii) It doesn't occupy space.	(ii) It occupy 1byte space
(iii) endl is available in std namespace	

Variable :-

Q. what is variable ?

i. A variable is a named memory location.

ace

ii. A variable is a container which contain value.

Local variables :-

i. A variable declared inside a function is called local variable.

Properties of local variable.

LIFETIME :- untill execution OF Function

scope :- Within the Function

Initial value :- garbage or unknown

* C++ allows you to declare Local variables within the function.

Ex.

P3.
```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{ int x;
  cout<<x<<endl;
int y;
cout<<y<<endl;
getch();
}
```

O/→ garbage 1, garbage 2

P4. //input values from keyboard
```
#include<iostream>
#include<conio.h>
```

```
using namespace std;
int main()
{
  int x, y, z;
cout << "\n input x, y values";
cin >> x >> y ;
z= x+y;
cout<<endl<< z;
getch();
return 0;
}
```

O/P→ input x, y values
10
20
30

Initializing a variable :-

1. Declaring variable by assigning a value is called initialization.
2. C++ allows to initialize a variable by assigning constant or expression.

Syntax:- datatype variable-name = value;
datatype variable-name = expr;

ex:- int a=10; } → static initialization
int b= 20; }
int c = a+b; → Dynamic initialization

int x; → declaration
x=10; update or assignment

Q. what is static initialization?

(i) Declaring variable by assigning const is called static initialization.
(ii) In this initial value of variable is known at compile time.

Q What is dynamic initialization?
(i) Declaring variable by assigning expression is called dynamic initialization.
(ii) In this initial value of variable is known at run time.

<iostream> → contain namespace
<iostream.h> - don't contain namespace

Prog 5
```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ int x=10;
  int y= 20;
  int z= x+y;        o/p- 30
  cout << z;
  return 0; → It is returned to OS to ensure operation is successfully performed.
}
```

or

```
#include <iostream>
#include <conio.h>          () - constructor
using namespace std;
int main()
{ int x(10);
  int y(20);          o/p -30
  int z(x+y);
  cout << z;
  getch();
  return 0; }
```

↳ C++ allows two types of initializations
1. using assignment (=) operator
2. ()- (constructor (compilers developed C++ version can use () this for initialization)
int x=10; or int x(10);

Declaring constant in C++
↳ The value of constant variable is never changed.
↳ In C++ we declare constants using const Keyboard.

Syntax:- const datatype variable-name = value;
ex const Float Pi = 3.147;
Float r;
cin >> r;
Float a = Pi * r * r;

P6. Program to find simple interest?

A:-
```
#include<iostream>
#Include<conio.h>
using namespace std
int main()
{
  Float amt;
 const Float rate=1.5;
    int t;
 Cout<<"Input amount";
 cin>>amt;
 cout<<"\n input time";
  cin>>t;
 Float si = (amt*rate*t)/100;
 cout<<"\n simple interest is"<<si;
   getch();
   return 0;
}
```

P7. Finding area of circle?

A:-
```
#include <iostream>
#include <conio.h>
#define Pi = 3.147
using namespace std;
int main()
{
    Float r;
  cout <<"\n input radius";
  cin >> r;
 Float a = Pi * r * r;
 cout << "\n Area is" << a;
  getch();
    return 0;
    }
```

Global variable:-

→ A variable declared outside the Function is called good global variable.
→ It is used to share data between multiple Functions or classes.
→ Local variables default value - garbarag
→ Global variables default value is
int, shortint, long int, → 0
double, Float → 0.0
Boolean → 0 (false)
char → 0 \0
↳ 1st priority is given to local variables not the global variables.

P8.
```
#include<iostream>
#include<conio.h>
using namespace std;
```

```
int x;
Float y;
boolean z;
int main()
{ cout<<x<<endl;
  cout<<y<<endl;
  cout<<z<<endl;
getch();
 return 0;
}
```

Pq. Program For variable hiding?

A:-
```
#include<iostream>
#include<conio.h>
using namespace std;
int x=10;
int main()
{ int y = 20;
  int x=30;
cout<<x<<endl;
 cout<<y;
getch();
return 0;
 }
```

We can access global variable within the Function using scope resodtion operator.

↳ C++ allows you to declare global and local variable with the same name.

Q What is Variable hiding?

A:- Declaring variable inside Function with same name as global variable is called hiding variable.

In this global variable is hidden inside the Function.

Q. How can a '::' operator be used as unary operator?

Ans: The scope operator can be used to refer to members of the global namespace.

↳ Because the global namespace doesn't have a name, the notation :: member-name refers to the member of global namespace.

↳ This can be useful for referring to members global namespace whose names have been hidden by names declared in nested local scoped.

Eg. // use of scope-resolution operator

```
#include<iostream>
#include<conio.h>
using namespace std;
    int x=10;
int main()
{  int x=20;
    int y=30;
cout<<"\n local variable x="<<x;
cout<<"\n local variable y="<<y;
cout<<"\n Global variable x="<<::x;
    getch();
    return 0;
}
```

O/p local variable x = 20
Local variable y = 30
Global variable x = 10

Pg. // Accessing global variables with/without ::

```
#include<iostream.h>
#include<conio.h>
using namespace std;
int y=20;
int main()
{  int z=30;
 cout<<"\n local variable z="<<z;
 cout<<"\n local variable y="<<y;
 cout<<"\n local variable y="<<::y; getch();
```

NOTE> (we can access global variable using '::' operator or without it. If use then it will directly goes to global area, otherwise 1st search in Local then goes to global area.)

Block level Variable :- The variables declared in any one of the Following blocks are called block level variable.

1. Anonymous block
2. conditional block
3. Iterational Block

1. Anonymous block - A block which doesn't have any name is called anonymous block.

```
{
    Statements;
}
```

Scope of this variable is inside block, lifetime untill execution of block.

~~2. conditional block :-~~

P12 // Anoymous block example.

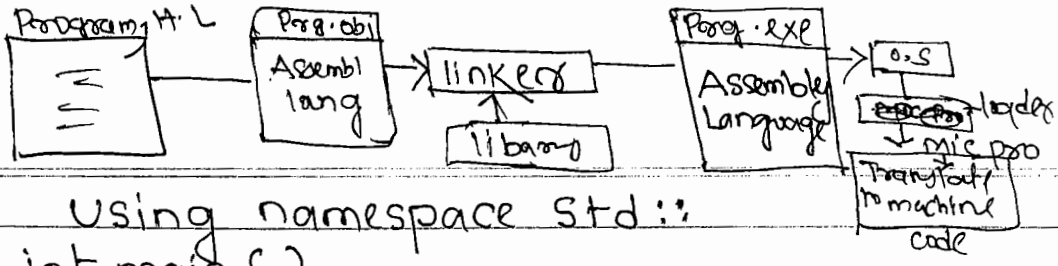
```
#include <iostream>
#include <conio.h>
int main()
{  int x=10; (local level)
   {
      int y=20; (block level)
      cout<<y;
      cout<<x;
   }
  cout<<y; X → Error
  getch();
  return 0;
}
```

2. Condition block :- A block followed by conditional block statement like if, switch. is called conditional block.

```
if (boolean expression)
   {
      Statements;
   }
  else
     { statements;
     }
```

P13. // Example of conditional block

```
#include <iostream>
#include <conio.h>
```




```
Using namespace std;
int main()
{ char name[10];
  int sub1, sub2;
 cout<<"\input name";
  cin >> name;
cout<<"\ input two subjects";
 Cin >> sub1 >> sub2;
iF(sub1>=40 && sub2>=40)  //It is true then onl
                          memory to b an
  {
     int total = sub1+sub2;
     Float avg = total/2;
   cout<<"\n total" << total;
    cout<<"\n avg" <<avg;
    cout<<"\n Result pass";
}
else
   cout <<"\n Result Fail";
   getch();
return 0;
   }
```

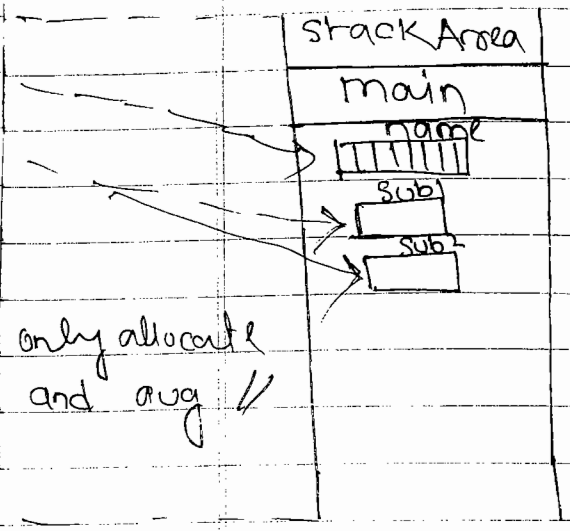
2. what is compiler, Linker, loader?

A Compiler :- c++ compiler translates source code to object code, object code is an assembly language code.

Linker :- Linker link/load all executable blocks and generate .exe.

loader :- ~~C++~~ loading the program from secondary to primary memory.

→ Microprocessors having an assembler which translates assembly code to machine code.



3. Iterational block :- A block which followed by while, for, do are called iterational block.

```
ex  while (condition)
    {
        statement ;
    }
For(exp1 ; exp2 ; exp3)
    {   statement ;
    }

do
  { statement;
  } while(condition)
```

P14

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
```

```
{ For(int num=1; num<=10; num++)
    cout<<num<<endl;

    cout<<num << endl;   // error
    getch();  ↳ outside the
    return 0;          For block
}
```

Pointer:- i. Pointer is a derived datatype.

(ii) A variable of type pointer is called pointer variable.

(iii) Pointer variable hold address of memory location.

Q ~~what is address~~?

A(i)

Aduantage :-

1. Pointers increases the performance.
2. Pointers avoid wastage memory
3. Dynamic allocations.
4. Share data betn Functions
5. Low level programming can be done using pointer

Disaduantages :-

1. Pointers are not secured.

Q. Who generate address?

A Address is generated by O.S.

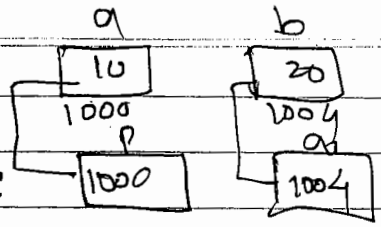
Syntax For pointers :

datatype *variable-name;

* - is called as indirectional operator.
- is called as dereferencing operator.
- Value at given address operator.

P4. //
```
#include<stdio.h>
#include<conio.h>
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
  int *p;
  Float *a;
  double *d;
cout<<"\size of int pointer"<<size of(p);
cout<<"\size of Float pointer"<<size of(a);
cout<<"\size of double "" "<<size of(d);
    getch();
    return 0;
  }
```
o/p size of int pointer 4
" " Float " 4
" double " 4

P5 // Reading values using pointer
```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{  int a=10, b=20;
    int *p, *a;
```




```
    P= &a;
    q= &b;
cout<<"\n a="<<*P;
cout<<"\n b="<<*q;
    getch();
    return 0;
  }
```

P16 // swapping two nos using pointers

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{  int a,b;
   int *P, *q;
   P= &a;
   q= &b;
cout<<"\n input any two numbers";
cin>>*p>>*q;
  *P= *P+*q;
  *q= *P-*q;
  *P= *P-*q;
cout<<"\n After swapping";
cout<<"\n a="<<a;
cout<<"\n b="<<b;
    getch();
  return 0;
  }
```

o/p 20 10

Dynamic memory allocations
New
Delete

Q What is Dynamic memory allocation?
A Allocating / managing the memory during runtime and execution of program is called DMA.

Advantages:—
1. Avoiding wastage of memory
2. It increases efficiency of program or applicatn

Q. what is the difference bet^n SMA & DMA.

S.M.A	D.M.A
1. Allocating memory before executing program is called S.M.A.	1. Allocating memory during execution of program.
2. wastage of memory.	2. Avoid wastage of memory
3. Stack area	3. Heap area

new :-
1. new is a keyword or operator.
2. new reserves memory of given size in heap area and return address.
3. syntax :-

```
<pointer-variable> = new
  datatype[size];
```

Q what is difference between new and malloc?

A:-

new	malloc
1. It is a keyword	1. It is a Function.
2. It allocates memory for one or more than one value.	2. allocates only block of memory.
3. new doesn't require type casting.	3. malloc requires type casting 4. malloc return type void*

Pg. // Program to add 2 nos ?

A:-

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
  int *a,*b;
 a=new int;
 b=new int;
cout<<"\n input any 2 numbers";
cin>>*a>>*b;
int *c=new int;
 *c=*a+*b;
cout<<"\sum is "<<*c;
 getch();
  return 0;
 }
```

delete keyword / operator:

1. delete unreserves memory which is reserved by new operator/keyword.

Syntax

delete <pointer-variable>;

2. delete unreserve memory during run-time or execution of ~~memory~~ program.

Q what is dangling pointer?

A:- A pointer variable which holds address of unreserved memory location or variable whose life time is over is called dangling operator.

Q. what is memory leak?

Memory which is reserved by a program is unable to unreserve, that memory is leaked from program.

P18.

```
//
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
   int *a;
  a= new int;
   *a=10;
cout<<*a<<endl;
 delete a;
cout<<*a<<endl;
getch();
 return 0;
 }
```

o/p -

	Dev C++	Turbo C++
	10	10
	0	10

Q. what is Dynamic array?

A:- Allocating memory for more than one value of similar type during execution of program is called dynamic array.

1. Dynamic single-dimension array
2. Dynamic multi-dimension "

Dynamic single-dimesional array :-

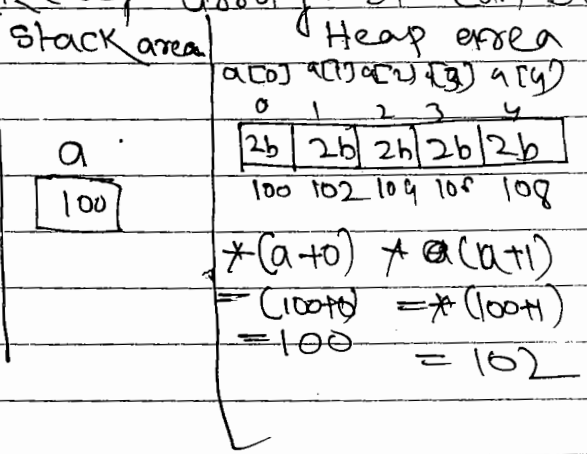
i. An arrary whose elements referred with one sub-script is called single dimension array.

ii. In this elements organized into logically by dividing into row and columns.

Syntax:- <pointer-variable> = new data-type[size];

Size -> it is size of array. it can be variable or constant.

```
int *a;
a = new int[5]
delete a;
```



Pg. // write a program to read 'n' integer value and display.

```
#include<iostream>
#include<conio.h>
using namespace std
int main()
{ int *a, n;
    int i;
  cout<<"\n input how many values";
```

```
cin >> n;
 a= new int[n];
 For (i=0; i<n; i++)
 cin>>a[i];
cout<<"\n values are";
cout<<a[i];<<endl;
 delete

delete a;
 getch();
 return 0;
}
```

P20. // wap to read the scores of n players and display total?

A:-

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{   int *a,n;
    cout<<"\n Input no. of players";
    cin>> n;
   a= new int[n];
    For(int i= 0; i<n ; i++)
    cinn>> a[i];
cout<<" players scores are";
 cout<<a[i] <<endl;
int c = 0;
    c= c+a[i]
   For( i =0; i<n; i++) // Summing
     C= C+a[i];
   cout<< c <<endl; // Displaying scores
    getch(); return 0; }
```

Pa1 // wap to read, name, n subjects marks & display total and avg.

H/W

(Home work)

Pa2 wap to read n elements and display which one is max and min?

H/W

Ans:-

(Home work)

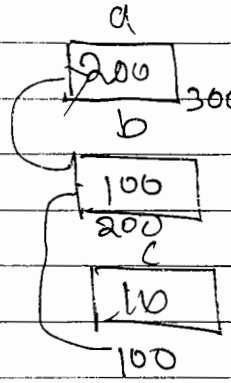
Dynamic double dimensional array:-

1. An array of array is called double dimensional array. Creating this array at runtime is called dynamic double dimensional array.
ii. This array is refer with two subscripts.
(iii) Elements are organized by dividing into rows and columns.

Steps For creating dynamic double dimesional array
1. Declare pointer to pointer
2. Create array of pointers (rows)
3. Create array of values hold by each row (no. of columns)

Syntax :- Declaring pointer to pointer
<datatype> = **variablename;

```
int **a;
int *b;
int c=10;
b=&c;
a=&b;
cout<<a; - 200
cout<<b; - 100
cout<<c; - 10
cout<<*a; - 100
cout<<**a; - 10
```



2. Create array of pointers (rows)

new datatype*[size];

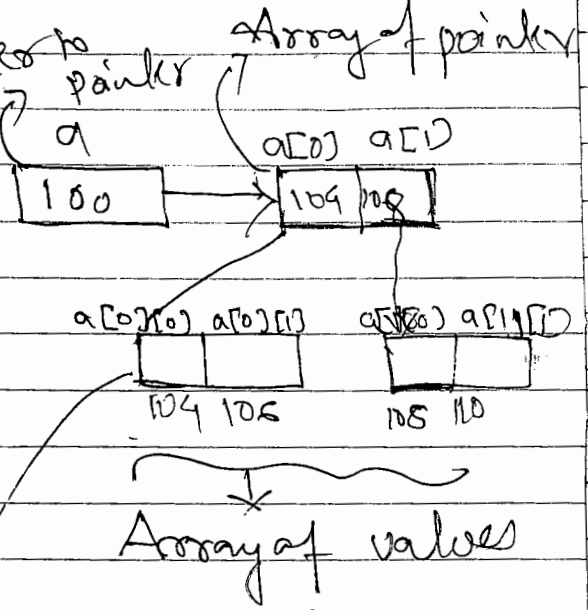
3. Create array of values (columns)

new datatype[size];

int a[2][2] → Array of values.
| ↳ Array of pointer
Pointer to pointer

1. int **a;
2. a = new int*[2];
3. *(a+0) = new int[2];
*(100+0) = "
*(100) = "

*(a+1) = new int[2];
*(100+1) = "
*(102) = "



((a+0)+1)
= *(*(100+0)+4)
= *(*(100)+1)
= *(104+1)
= *(106)

((a+0)+0)
= *(*(100+0)+0)
= *(104)

((a+0)+1)
= *(*(100+1)+0)
= *(*(102)+0)
= *(108)

P23 // wap read 2x2 matrix and display?

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{  int **a;
   int i,j ;  // i-row index & j-column index
   a= new int* [2];
  *(a+0) = new int[2];
  *(a+1) = new int[2];
cout<<"\n input elements";
 For(i=0; i<2 ; i++)
 For(j=0; j<2 ; j++)
cin>> *(*(a+i)+j);

Cout<<"\n elements are";
   For(i=0; i<2; i++)
   For(j=0; j<2; j++)
cout<< *(*(a+i)+j)<<"\t";
   getch()
  return 0;
  }
```

P24 // wap to read M students and N subjects marks and display.

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{   int **marks;
      int m, n, i, j ; // i-row indx & j-column index
cout<<"\n input how many students";
```

```
    Cinn >> m;
cout << "\n input how many students";
 Cin >> n;
  marks = new int*[m];
    For(i=0; i<m; i++)
    *(marks+i) = new int[n];
cout << "\n input marks";
 For( i=0; i<m; i++)
 {
   For(j=0; j<n; j++)
cout << *(*(marks+i)+j) << "\t";
   cout << endl;
 }
  getch();
    return 0;
}
```

```
For(i=0; i<m; i++)
For(j=0; j<n; j++)
cin >> *(*marks+i)+j);
```

```
For(i=0; i<m; i++)
delete *(marks+i);
delete marks;
```

```
cout << "marks are \n";
```

Reference :-

1. Reference is a derived datatype.
2. Reference is a implicit pointer.
3. Reference hold address of variable.
4. Reference variable is an alias variable.
5. It doesn't require indirection operator to read or write values.

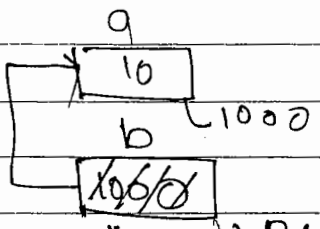
Advantage

1. It is used in functions to share local data.
2. restricted pointer, address can't manipulated.

:-Syntax :-

datatype &variable = variable;

```
int a=10;
int &b=a;
Cout<< a :-> 10
Cout<<b -> 10
   b=100;
Cout<<a; 100
cout<<b; 100
```



Pgs. // ex: of reference

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{ int a=10;
  int &b=a;
 cout<<a<<endl;
 cout<<b<<endl;
  getch();
  return 0;
}
```

o/p 10
10

P26. // Find output

```
#include<iostream>
#include<conio.h>
Using namespace std;
int main()
{   int a=10;
    int &b=a;
```

```
    b= 50;
    a=100;
    cout<< a <<endl;
    cout <<b <<endl;
    getch();
    return 0;
}   o/p - 100
              100
```

p 27

```
//
#include<iostream>
#include<conio.h>
using namespace std
int main()
{
    int &b = 100;
    getch();
    return 0;
}
```

o/p - Error

Reason:- Above program shows compile time error because reference variable can't initialize with constant. Initialize with variable only.

```
int a=10;
int &b=a;
int &c= b; // alias of a
cout<<a; 10
cout<<b; 10
cout <<c; 10
```

c=100;
cout<< a; -> 100
cout<<b; -> 100
cout<<c; -> 100

Q. What is the difference betn reference & pointer ?

Ans:-

Pointer	Reference
Address hold by Pointer can be changed.	Address hold by reference cannot be changed.
(ii) Require indirection operator to refer value	(ii). It does not require indirection operator.

Functions in C++ :-

Q what is a Function?

Ans: i. A Function is a self contained block which contain set of instructions.

ii. A function is a small program within program.

iii. A Function is a sub-routine.

Advantages of Functions:-

1. Modularity : Dividing instructions according to their operations.
2. Reusability : write code once & use many times.
3. This avoid redundency.
3. Efficiency :- Function decrease size of program.
4. Simplicity :- Easy to understand.

Syntax:

```
returntype Function-name([parameters])
{
    statements;
}
```

returntype :- (i) returntype is a datatype, which define the type of value return by Function.
(ii) A function which doesn't return any value is defined with void.

Function-name :- Function-name is an identifier (user-defined name).

Parameters :- Parameters are local variables, which receive values from another Function or caller.
C++ allows to write Function in 4 ways
1. Function with returntype with parameters
2. function with " without " .
3. function without " with " .
4. function " " without " .

P28:
A:-

```
// Function without returntype & without parameter
#include<iostream>
#include<conio.h>
using namespace std;
void draw_line()  // called Function
{  int i
   For(i=1; i<=40; i++)
      cout<<("*");
}
   int main()     // calling Function
   {  draw_line();
      cout<<endl<<"c++";
       draw_line();
      cout<<endl<<"object oriented";
       draw_line();
```

```
    cout<<endl<<
    getch();
    return 0;
}
```

o/p- ****- - - - *
c++
****- - - - *
object oriented
***- - - *

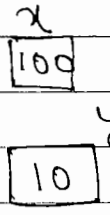
↳ when Function is called execution control is mould From calling function to called function, After execution it returns to calling function.

↳ Memory For fnction is allocated on calling and deallocated after execution.

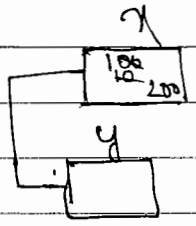
↳ c++ allows to call function in 3 ways
1. Pass by value
2. Pass by address
3. Pass by alias/reference

1. Pass by value :-
i. In pass by value function is called by sending value types (constants/variables)
ii Pass by value uses deepcopy. values are copied from calling to called functions.

Deep copy
int x=10;
int y=x;
x=100;



Shallow copy
int x=10;
int *y=x;
x=100;
y=200;



↳ The values which are passing From calling Functions to called Function are called actual arguments / parameters.

↳ Function parameters are called Formal arguments / parameters.

Pag //

```
#include<iostream>
#include<conio.h>
using namespace std;
void draw_line(int size) // called
 { int i;
   cout<<endl;
   For(i=1; i<=size; i++)
    cout<<("*");
 }
int main()  //calling
{  draw_line(30);
   cout<<("\nc++";
    draw_line(20);
   cout<<("\n object oriented";
    draw_line(40);
    getch();
    return 0;
   }
```

P30. //

```
#include<iostream>
#include<conio.h>
using namespace std;
void Find_result(int s1, int s2)
{ if(s1<40||s2<40)
   cout<<"\n Result Fail";
   else
    cout<<"\n Result pass";
}
 int main()
 {  char name[10];
    int sub1, sub2;
  cout<<"\n input name";
   cin>>name;
 cout<<"\n input 2 subjects";
   cin>>sub1>>sub2;
   cout<<"\n name is"<<name;
 Find_result(sub1, sub2);
getch();
 return 0;
}
```

Inline Functions :-

i) Inline is a keyword. It is used with Functions.

P31. Avoiding control switching between calling Function to called Function?

A:

```
Void sum(x+i, 1+y)
{   i+z;
    z= x+y;
}  cout<<z;
```

```
int main()
{ sum 10,20;
  void add()
  }
```

1. Everytime a function is called, it takes a lot of extra time in executing a series of instruction like jumping to the function, pushing arguments to the stack and returning to the calling function.

ii When function is small a sub: percentage of execution time may be spent in overheads.

iii. To eliminates the cost of call to small functions the solution is intime Function.

(iv). An ~~implementation~~ inline function is declared with keyboard inline.

(v). An inline function is a function that is expanded intime when it is involved.

(vi) The compiler replaces the function call with corresponding function code.

Syntax inlineFunction - header
{ Function body
}

P32

```
inline void print()
{ cout<<"Inside inline function";
}
int main()
{ print();
  print();
  return 0;
}
```

* The speed benefits of the functions diminish as the function grows in size.

* Care has to be taken before making a function inline.

* If the overhead of the function call becomes small compared to the execution of the function, the benefits of the inline may be lost

* Usually the Function are made inline when they are small.

Support Dev c++ :-

P33

```
#include<iostream>
#include<conio.h>
using namespace std;
inline void print()
{ cout<<"inside inline function";
}
int main()
{ print();
  getch();
  return 0; }
```

Situations where inline expansion may not work :-

(i) For Function returning values, if a loop is switch or a goto exists.
(ii) For a function returning values if return statement exists.
(iii) If functions contains static variables
(iv) If inline function are recursive.

Drawbacks of macros :-

1. macros are not functions
2. If it is used in complex expression it leads to logical errors.
3. Type checking is not done.

Examples :- P34

```
#include <iostream>
#include <conio.h>
Using namespace std;
inline int sqr(int n)
{
    return n*n;
}
int main()
{
    int s = sqr(5);
    cout<<"\n SQR IS "<<s;
    getch();
    return 0;
}
```

Function with default arguments or optional arguments

* C++ allows to call function without specifying all its arguments.
* The Function assigns a default value to the parameter, which doesn't have a matching arguments in the Function call.

* Default values are specified when the Function is declared.
* Default values are given from right to left.

```
#include<iostream>
#include<conio.h>
using namespace std;
void simple_interest(float p, int n, float r=1.5f)
{
    float si = (p*n*r)/100;
    cout<<" In simple interest "<<si;
}
int main()
{
    simple_interest(5000, 12);
    simple_interest(6000, 24, 2.5f);
    getch();
    return 0;
}
```

P36 // Function with default argument

```
#include<iostream.h>
#include<conio.h>
using namespace std;
Void Fun1(int x=10, int y=20)
 {   cout<<x<<endl;
     cout<<y<<endl;
 }
int main()
 {  Fun1();
    Fun1(30);
    Fun1(40,50);
   getch();
  return 0;
}
```

P37 // Inserting an element in array

```
#include<stdio.h>
#include<stdio.h>
using namespace std;
Void insert(int a[], int e, int s, int p=1)
 { For(int i=s; i>p; i--)
     a[i]=a[i-1];
     a[i]=e;
   }
 int main()
{  int a[5]= {10, 20, 30}
    insert(a, 40, 3, 3)
   cout<<"After inserting element\n";
   For(int i=0; i<4; i++)
       cout<<a[i]<<endl;
        getch();
       return 0;
}
```

o/p 10
20
30
40

Pass by address :-

calling or invoking function by sending address type value or pointer is called pass by address.

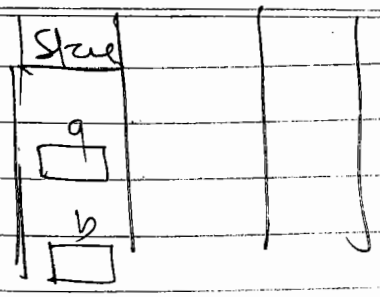
↳ In pass by address function is declared with parameters of type pointers.

Adv :- (i) Sharing data betn functions.
(ii) Avoiding wastage of memory.
(iii) Modifying actual arguments or local variables outside the function.

P38. // swapping two numbers

```
#include<iostream>
#include<conio.h>
using namespace std;
void swap(int *p, int *q)
{ int s= *p;
 *p = *q;
  *q = s;
}
  int main()
{ cout<<"\input a,b values";
   cin>>a>>b;
  swap(&a, &b);
cout<<"\n after swapping \n";
 cout<<"\n a = "<<a;
  cout<<"\n b = "<<b;
   getch(); return 0;  }
```

o/p input 2 values
10
20
After swapping
a = 20
b = 10



Pass by alias @ / Reference :-

1. In pass by reference Function is called by sending value type variable. These variables called Function refere with reference type or alias type.
2. In pass by reference function parameters are declared of type reference.

Adv :- 1. Sharing data between functions
2 references are secure then pointers.

Prg.
```
// swapping two numbers
// using pass by alias
#include<iostream>
#include<conio.h>
using namespace std;
void swap(int &p, int &q)
{ int s=p;
  p=q;
  q=s;
}
int main()
{ int a,b;
```

```
    cout<<"\n Input a, b values";
    cin>>a>>b;
     swap(a, b);
   cout<<"\n After swapping \n";
    cout<<"\n a="<<a;
    cout<<"\n b="<<b;
   getch();
    return 0;
}
```

O/P Input a values
10
20
After Swappings
a = 20
b = 10

Function Overloading :-

1. Defining more than one function with same name by changing
   1. No. of parameter
   2. Types of "
   3. order of "
   is called function Overload.

2. when more than one function perform similar operation with different implementation then function is overloaded.

3. Group of functions which perform similar operations are refer with single name.
   - overloaded function having polymorphic behaviour.

4. Function overloading is called name polymorphism.

5. Defining one thing in many forms is called polymorphism.

Q6. what is name mangling ?

Ans:- i. Name mangling is the process through which your c++ compiler gives each function in our program a unique name.

ii. In c++ all programs have at-least few functions with the same name. Name mangling is a concession to the fact that linkers always insist on all function names being unique.

compiler	linker
1. Compiler recognize a function with name, no. of parameters, type of parameter, order of parameter	1.

P40 // Function Overloading ?

```
#include<stdio.h>
#include<conio.h>
using namespace std;
int max(int x, int y) // compile give name max → unique name
{ if(x>y)
    return x;
  else return y;
}
```

unique name

```
Float max ( Float x, Float y) // Float ma_xy
    {   if(x>y)
          return x;
        else
          return y;
    }
int main()
{    int m1 = max (10, 20);
     Float ma = max (1.5, 2.5);
cout <<"\n max of two integers" <<m1;
cout <<"\n max of two Floats" <<ma;
   getch();
   return 0;
  }
```

1. compiler recognize function with its signature which includes no. of parameters, types of parameters and order of parameters.

P41. // searching element within array?

```
#include<iostream>
#include<conio.h>
using namespace std;
Void search (int *a, int s, int e)
{ bool Flag = false;
For(int i=0; i<s; i++)
{ if(a[i]==e)
   Flag = true;
  }
if(Flage = true)
cout <<"\n element Found";
  else
   cout <<"\n element not found";
}
```

```
Void Search(int *a, int s, int e, int p )
{  bool Flag = False;
 For(int i=p; i<s; i++)
 {
      iF(a[i]==e)
        Flag = true;
}
  iF( Flag==true)
cout<<"\n Element Found";
else
cout<<"\n elemut not found";
}
  int main()
{
  int a[10], n, p, e;
cout<<"\n input how many elements";
 cin>>n;
cout<<"\ninput elements";
For(int i=0; i<n; i++)
 cin>>a[i];

cout<<"\ninput search element";
 cin>>e;
Search(a, n, e);
cout<<"\n input search element;
cin>>e;
cout<<"\ninput pos";
  cin>>p;
Search(a, n, e, p);
getch();
 return 0;
      }
```

O/P - Input how many elements

Sreekanth

P4a. // overloading Function to Find area of triangle and circle.

```
#include<iostream>
#include<conio.h>
using namespace std;
Float Find_area( Float b, Float h)
{ return 0.5*b*h;
}
Float Find_area (Float r)
{ return 3.14*r*r;
}
int main()
{
  int opt;
Float base, height, r, a1, a2;
cout<<"1. Triangle"<<endl;
cout<<"2. circle"<<endl;
cout<<"\n input option";
cin>>opt;
```

```
switch (opt)
{ case 1:
    cout << "\n input base height";
    cin >> base >> height;
    a1 = Find_area (base, height);
    cout << "\n area of triangle" << a1;
    break;
case 2:
    cout << "\n input r";
    cin >> r;
    a2 = Find_area (r);
cout << "\n Area of circle" << a2;
break;
    }

getch ();
  return 0;
}
```

Q1.
```
int Fun1 ()
{ }
Float Fun1 ()
{
}
```

Above overloading is invalid becoz Function is overloaded by changing no. of parameters or types of parameters, not by changing return type.

2.
```
int Fun1 (int x, int y)
  { }
int Fun1 (int x, int y, int z=10)
  { }
```

Above overloading is invalid becoz Fun1 with default argument match with existing Function.

Q3.
```
Void Fun1(int x, Float y)
{
}
Void Fun2(Float x, int y)
{
}
```
Above overloading is valid becoz function can be Overloading can be overloading changing type of parameters.

Q4.
```
Void Fun1(int x, int y)
{
}
void Fun2(int *x, int*y)
{
}
```
Above overloading is valid.

Q.
```
Void Fun1(int x, int y)
{
}
Void Fun2(int &x, int &y)
{ }
int a=10, b=20;
Fun1(a,b);
```
Above overloading is invalid, because function with value type can't be overloaded with alias/reference type.

Syntax of calling function passing value and alias is same.

Classes & object :-

Class- i. class is encapsulated with data members and member functions.

ii. class is collection of members. These members are of 2 types
1. Data member
2. Member function

iii. class is datatype/ user defined data types.

Q. what is difference bet'n structure in 'c' & class in C++?

Structure	Class
1. It contains only variables.	It contains variables & functions.
2. Members are Public/ global.	2. Members are private, Public or protected.
3. It doesn't support Polymorphism & inheritance	3. It supports polymorphism & inheritance.

(iv). class is a reusable module.
object oriented application is collection of modules and each module is called class

Programming elements are 2
1. Data
2. Instructions
data and instructions which operates on data encapsulated within container called class.

v. class define structure of object.
vi. class allocates memory for object.

Syntax:- class class-name
{ Private:
data members ;
member-Functions ;
Protected:
data members;
member-Functions;
Public:
data-members;
member-function;
};

Q what is data member ?
A:- i. Variables declared inside class are called data members.
ii. Variables are declared to define state of the object.

Q what is member Function ?
A:- i. member Function is Function which is bind with class or object.
ii. It defines behaviour of object.
iii.

Binding of data member and member-functions in a class is called encapsulation.
Advantage of encapsulation is data hiding.

object :-

i) object is an instance of class.

ii. It is process of allocating memory for members of the object.

iii. Object is a class variable.

Syntax :-

```
class-name object-name;
```

P43. // Explaining class & objects :-

```
#include <iostream.h>
#include <conio.h>
using namespace std;
  class date                    //default access specifier - Private
{ int dd, mm, yy;
};
int main()
{ date dob;        // dob -> Local object
   dob.dd = 5;  // . -> is called member access operator
   dob.mm=6;
   dob.yy=2014;
    getch();
     return 0;
  }
```

O/p - Error

The above Program display compile time error, because members of class private. private members cannot access by non members functions.

Q. How to achieve data hiding in object oriented?

i. By declaring variables as private.

ii. Default members of class are private.

iii. Private members are access by members of same class can't access outside.

Syntax :-

object-name . member-name .

P44 // example of public

```
#include<iostream>
#include<conio.h>
Using namespace std;
Class time
{ public:
  int hh, mm, ss;

};
int main()
{
    time time_in;
    time_in.hh = 10;
    time_in.mm = 2;
    time_in.ss = 10;
getch();
  return 0;
}
```

Q what is Access specifier ?

Ans Access specifier define scope or visibility/ accessability of members defined within class.

C++ provide 3-access specifiers.

1. Private
2. Protected
3. Public

Private :- Private members of the class access only by members of same class but cannot access by non members or members of other classes.

```
class A
{ private:
    int x;
    Void Fun1()
    {
    }
    Void Fun2()
    {
    }
};

classB
{ void Fun3()
  {
    x
  }
};
```

(ii.) data hiding is achieved by declaring data members of the class as private.

(iii.) These private members are accessible outside the class with help of public member functions.

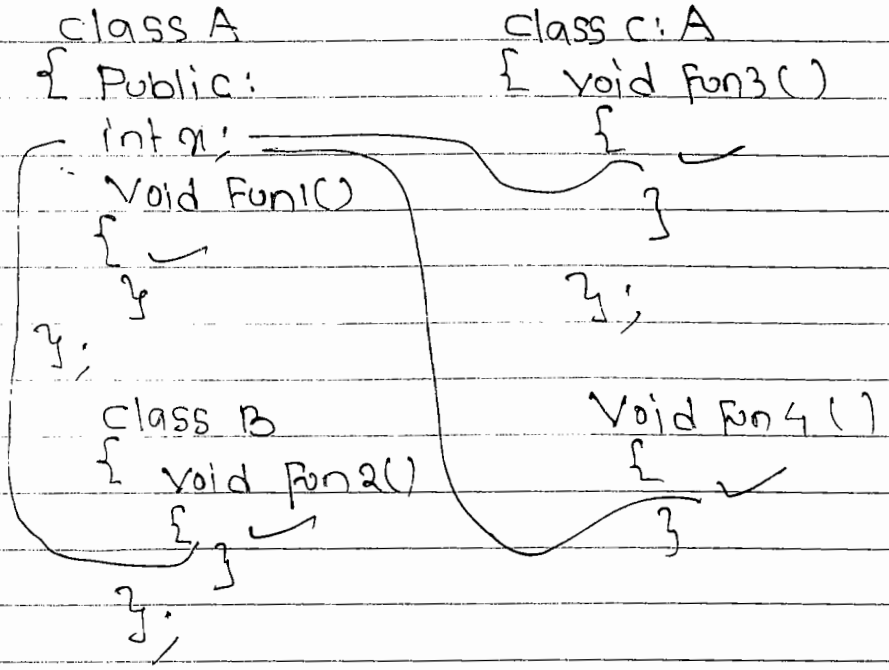
Protected :-

i. Protected members accessible with class and derived class.

ii. Cannot accessible in non-members or members of other classes.

```
class A
{ Protected:
    int x;
    Void Fun1()
    {
    }
};

class B
{ void Fun2()
  {
    x
  }
};

class C :A
{ void Fun3()
  {
  }
};

Void Function
{
  x
}
```

Public- Public members are accessible within class, derived class, non-members and members of other classes.



P45 // Program with data member & member function

```
#include<iostream>
#include<conio.h>
using namespace std;
class date
{ Private :
    int dd, mm, yy;
Public :
  Void set_date()
  { dd = 12;
    mm = 4;
    yy = 2014;
  }
  Void print_date()
  { cout<<dd<<"/"<<mm<<"/"<<yy;
  }
};
```

```
int main()
{ date d1;
  d1.set_date();
  d1.Print_date();
  date d2;
  d2.set_date();
  d2.print_date();
  getch();
    return 0;
}
```

o/p 12/4/2014
12/4/2014

↳ On creation of object memory is allocated for data-members of class, but doesn't allocate memory for member functions.

↳ Member memory is allocated for member functions when they are called by other function.

Q. What is modifier/mutator?
A member function which change state/value of an object is called modifier or mutator.

Q. What is accessor?
A member function which doesn't change the state of object is called accessor.

member function with Parameters:-

A member function having parameters receive values from calling program.
This member function is called for sending values to object.

P46. // Example of member function with parameters

```
#include<iostream>
#include<conio.h>
using namespace std;
class triangle
{
   Float base;
   Float height;
 Public:
void set_base_height(Float b, Floath)
 {
    base = b;
   height = h;
 }
Void Find_area()
{  Find
   Float area = 0.5*base*height;
  cout<<"\nArea is "<<area;
 }
 };
int main()
{   triangle triangle1;
    triangle triangle2;
  triangle1.set_base_height(1.5, 2.5);
 triangle2.set_base_height(2.5, 3.5);
  triangle1.Find_area();
 ,triangle2.Find_area();
  getch();
   return 0;
 }
```

O/p- Area is
Area is

Defining member Functions

C++ allows to define member function at two places.

1. Inside class
2. Outside class

1. Inside class :- i. The function declaration inside the class be replaced by the actual function definition.

(ii.) when a a function is defined inside a class it ~~relat-~~ treated as an ~~intimal~~ Inline function.

(iii.) Normally small functions are defined inside class

Q Define a way other than using the keyword inline to make a function inline?

A:- Function must be defined inside the class.

Defining member function outside a class :-

Member function definition defination is given outside the class to avoid inline expansion.

syntax :-

```
returntype class-name::member function name
                                    ([parameters])
{   statements;
}
```

class name tells the compiler that function name belongs to the class name.

The symbol :: is called scope resolution operator.
The scope of the function is restricted to class name.

P47. // write a program to find result of student.

```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{
    int rno;
    int sub1, sub2;
public:
  void set_student(int, int, int);
  void Find_result();
};
void student::set_student(int r, int s1, int s2)
{  rno=r;
   sub1=s1;
   sub2=s2;
}
void student::Find_result()
 {  cout<<"\n Roll no"<<rno;
    cout<<"\n subject1"<<sub1;
    cout<<"\n subject2"<<sub2;
 if(sub1<40 || sub2<40)
  cout<<"\n Result is Fail";
else
  cout<<"\Result is pass";
  }
   int main()
   {
```

```
Student stud1;
Student stud2;
stud1.set_student(101, 60, 70);
stud2.set_student(102, 40, 30);
stud1.Find_result();
stud2.Find_result();
getch();
return 0;
}
```

P48. // write a program to find a area of rectangle

```
#include<iostream>
#include<conio.h>
using namespace std;
class rectangle
{
  Protected:
    Float b, l;
  Public:
    Void read();
    Void Find_area();
};
Void rectangle :: read()
{
  cout<<"\n input b, l";
  cin>>b>>l;
}
Void rectangle :: Find_area()
{
  Float area = b*l;
  cout<<"\n area is "<<area;
}
```

```
int main()
{
    rectangle rect1, rect2;
    rect1.read();
    rect2.read();
    rect1.find_area();
    rect2.find_area();
    getch();
    return 0;
}
```

o/p -

Array of objects :-

Allocating memory For more than one object of similar type.

Syntax : class-name array-name [size];

ex int a[5], Student stud[5];

Each object exist in array is identified with index number.

P49. // wap to read scores of Five players and display, and each player having name, runs scored.

```
#include<iostream>
#include<conio.h>
using name space std;
class player
{
    Protected :
    int runs;
    char name[15];
```

```
Public :
Void read ();
Void display ();
};
Void player :: read ()
{
   cout << "Enter name";
   cin >> name;
  cout << "\n Enter runs";
   cin >> runs;
}

Void player :: display ()
{
    cout << "\n Name of player" << name;
    cout << "\n No. of runs" << runs;

}
int main ()
{  player P[5];
    int i;
   For (i=0; i<5; i++)
     P[i]. read ();

   for (i=0; i<5; i++)
      P[i]. display ();
  getch ();
   return 0;
   }
```

O/p -

H/W

P50 //wap to find result of 5 students.
(Home work)

~~X~~ / 10

P51 //wap to reread details of 3 Product, display total amount? dyn
(name, price).

(Home work)

class

root? dynamic object :-

i. Creating object during runtime is called as dynamic object.

ii. This object is created within heap area.

iii. dynamic object is created using new operator.

Syntax :-

<pointer_to_object> = new class name;

~~class name;~~

<pointer-variable-name> = new class name;

Pointer variable of type class hold address of object. It is called as pointer to object.

P52

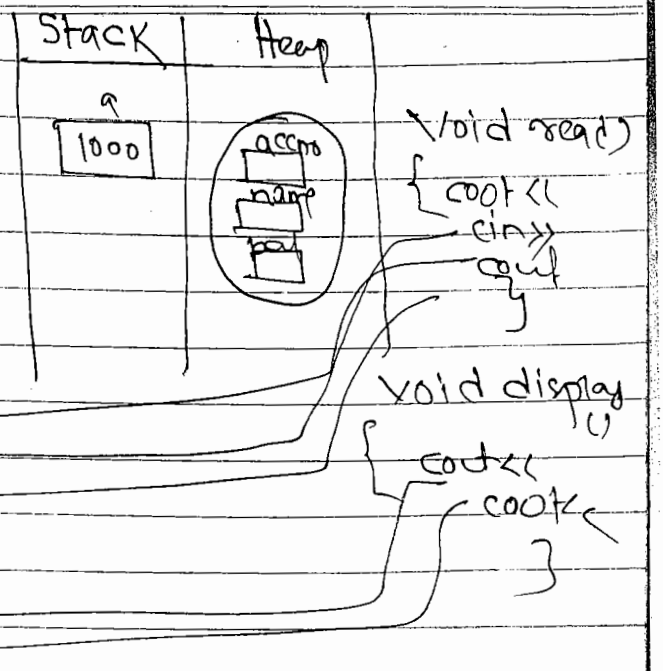
```
// Dynamic object example
#include <iostream>
#include <conio.h>
using namespace std;
class account
{ int accno;
  char name[10];
  Float bal;
Public :
Void read ();
Void display();
Void deposit ();
};
Void account :: read ()
{
  cout<<"\n input accno";
  cin>> accno;
 cout<<"\n input name";
 cin>> name;
 cout<<"\n input balance";
   cin>> bal;
}
```

```
Void account::display()
{   cout<<"\n Account no"<<accno;
    cout<<"\n customer name"<<name;
    cout<<"\n Balance"<<bal;
}
Void account::deposit()
{
    Float tamt;
   cout<<" Input amount";
    cin>>tamt;
   bal = bal+tamt;
}
int main()
{   account *acc1 , *acc2;
   acc1 = new account;
   acc2 = new account;
 (*acc1).read();
 (*acc2).read();
(*acc1).display();
(*acc2).display();
(*acc1).deposit();
(*acc1).display();
delete acc1;
 delete acc2;
 getch();
return 0;
}
```

O/P->

Explanation:-

```
int main()
{
    account *a;
    a = new account;
    *a.accno = 101;
    *a.read();
    *a.display();
```



Dynamic array of objects:-
creating more than one object of similar type inside heap area.
Syntax:-
(pointer-variable) = new class-name [size];
size can be declared as constant or variable.

P53. // wap to read the details of 'n' books and display?

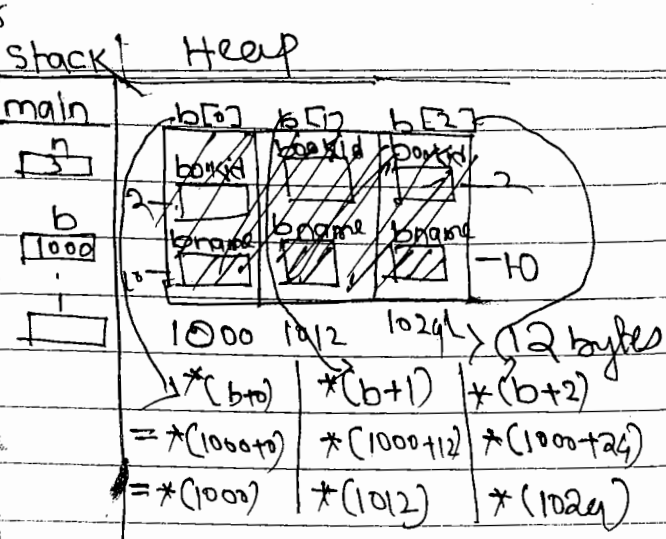
A:-
```
#include<iostream>
#include<conio.h>
using namespace std;
class book
{
    int bookid;
    char bname[10];
Public:
    Void read_book();
    Void Print_book();

};
```

```
Void book :: read_book()
{
    cout<<"\n Input bookid";
    cin>>bookid;
    cout<<"\n input bookname";
      cin>>bname;
}
Void book :: print_book()
{
 cout<<bookid<<"\t"<<bname<<endl;
}
  int main()
{   int n;
    book *b;
   int i;
cout<<"\n input how many books";
 cin>>n;
 b= new book;
 b = new book[n];
 For(i=0; i<n; i++)
(*(b+i)).read_book();

 For(i=0; i<n; i++)
 (*(b+i)).Print_book();

 delete b;
  getch();
    return 0;
}
```

1 Explanation :- non-member

```
int main()
{ int n;
  book *b;
  int i;
cout<<"input how may books";
  cin>>n
*(b+0).read_book();
*(b+1).read_book();
*(b+2).read_read();
For(i=0; i<n; i++)
(*(b+i)).point_book();
```

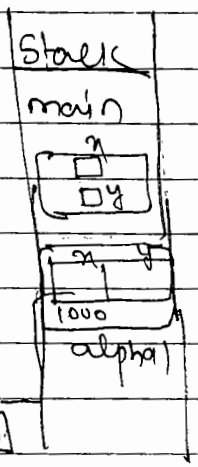
local variables



P54. // Program to access private members using pointers. // Pointers are not secured

A

```
#include<iostream>
#include<conio.h>
using namespace std;
class alpha
  { int x, y;
  };
int main()
  { alpha alpha1;
    int *p;
    p = (int*)&alpha1;
    *p=10;
    cout<<"\n"<<*p;
  getch(); return 0; }
```



Q How to access private members of the class?

A:- Private members of class accessed using

Illegal → 1. member-functions

2. Friend functions

Illegal → 3. Pointer

Constructor:-

1. Why constructor?

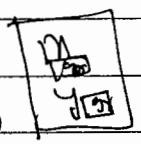
1. Data members can't initilize within class.

```
class triangle
{
    Float base = 1.5  // Invalid
    Float height = 8.2 // Invalid
}
```

2. If data members are not initialize then object is created with garbage values.

1.
```
class A
{ int x=10; X
int y=20; X
};
```

2.
```
class A
{ int x;
  int y;
};
int main()
{ A obj;
}
```



3.
```
class A
{ Public
  int x, y;
};
int main()
{ A Obj1={1,2}; X
}
```

```
class A
{ int x, y;
};
int main()
{
A Obj={10,20};
}
```

3. If data members of class are private Public, object can be initialize by assigning values. Public data members does not have security.

4. If the datamembers of class are private, object cannot be initialize assigning values at the time of creation.

Q what is constructor?

A:- i. constructor is a special member function.

(ii) Automatic initialization of object is done using constructors.

iii. It is used for allocating resources.

→ block of code which has to be executed on creation of object included inside constructed

Properties / characteristics of the constructor

1. constructors name must be same as class name.
2. It doesn't have any return type, not even void.
3. It is called automatically at the time of creating object.
4. It is called only once of the object.

c++ supports 3 types of constructors

1. Default constructor
2. Parameterized constructor
3. copy constructor

Q what are all the implicit members function of the class
or what are the all functions which compiler implements for us if we don't define one?

A define constructors
copy ''
Assignment operator
default destructor
address operator

default constructor:-

Non parameterized constructor of class is called default constructor.

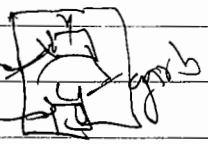
1. Userdefined
2. compilerdefined

Compiler defined default constructor.

```
Prog1.cpp
class alpha
{ int x;
  int y;
};
Void main()
{ alpha a1;
}
```

→ [c++ compiler] →

```
Prog1.cpp
class alpha
{ int x;
  int y;
public:
  alpha()
  {
    y;
int main()
{alpha a1;
}
```



If there is no constructor inside class then compiler provide constructor without parameters.

Userdefined default constructor:-

It is defined by programmer/user to Perform initialization.

Q what is the difference beth member function and constructor?

(A)

	member Function	constructor
1.	Its name can be any thing, it shouldn't be same as class	1. constructor name is same as class name.
2.	It can be called any no. of times on object	2. It ~~can be~~ is called only one on object.
3.	Its operation is modifying accessation.	3. Its operation is initialization.
4	with returntype	4. without returntype

4. It is called explicitly.	4. It is called implicitly.

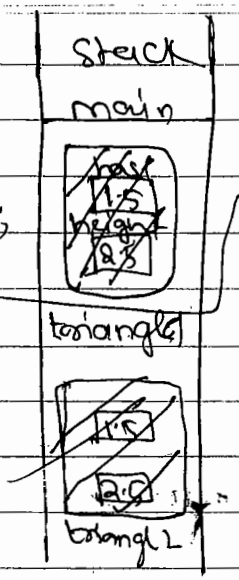
P55. // This a example for user defined default constructor?

A:-

```
#include<iostream>
#include<conio.h>
using namespace std;
class triangle
{
   Float base, height;
Public:
   triangle();
 Void Find_area();
};
triangle :: triangle()
{  base = 1.5;
   height = 2.5;
}
Void triangle :: Find_area()
 {
     Float area = 0.5*base*height;
    cout<<"\narea is"<<area;
}
int main()
{ triangle triangle1;
  triangle triangle2;
triangle1. Find_area();
triangle2. Find_area();
     getch();
 return 0;
   }
```

Explanation OF above Program

```
int main()
{ triangle triangle1;
  triangle.triangle();
  triangle triangle2;
```



P56. // default constructor

```
#include<iostream>
#include<conio.h>
using namespace std;
class array
{ int *a;
Public:
   array()
   { a= new int[5];
   }
void read_elements()
{
  cout<<"\n input elements";
   For(int i=0; i<5; i++)
    cin>>a[i];
}
void print_elements()
  {
   Cout<<"\n Enter elements are \n";
     For(int i=0; i<5; i++)
       cout<<endl<<a[i];
  }
};
```

```
int main()
{   array array1;
    array1.read_elements();
    array1.Print_elements();
getch();
return 0;
}
```

Parameterized constructor :-

1. constructor with parameter of type value address or reference.
2. This constructor receives values on creation of object.
3. It performs variable initialization.

P57 Book

```
// Example of Parameterized constructor
#include<iostream>
#include<conio.h>
using namespace std;
class date
{  int dd, mm, yy;
 Public:
date(int, int, int);
void set_date(int, int, int);
void print_date();
};
void date::date(~~int d, int m, int y~~)(int d, int m, int y)
{  dd=d;
   mm=m;
   yy=y;
}
void date::set_date(int d, int m, int y)
{   dd=d;
    mm=m;
    yy=y;  }
```

```
Void date :: Print_date( )
{
 cout<<"\n dd"<<dd;
 cout<<"\n mm"<<mm;
 cout<<"\n yy"<<yy;
}
int main( )
{
    date doj(10, 6, 2014);
    date dob(4, 6, 2014);
     doj. Print_date( );
      dob. Print_date( ); doj.set_date(6, 6, 2014);
    doj. Print_date( );
 getch( );
 return 0;
 }
```

O/P->

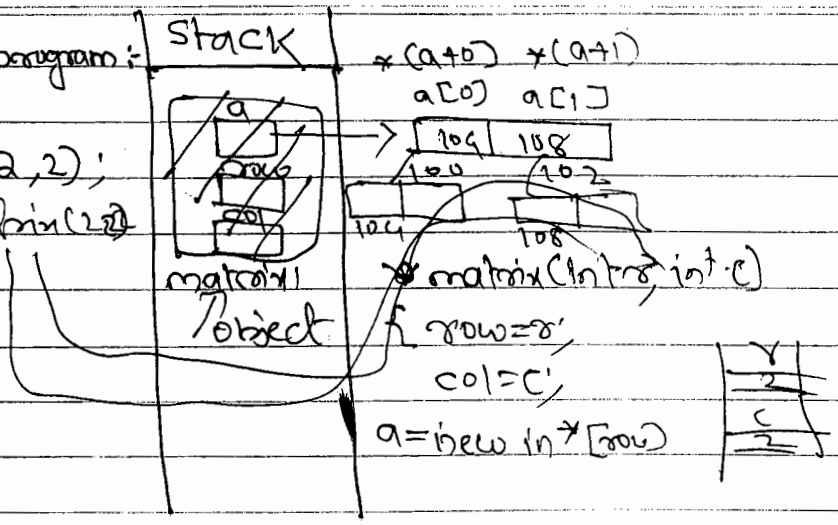
P58. // Parameterized constructor

```
A:- #include<iostream>
 #include<conio.h>
Using namespace std;
class matrix
{ int **a;
 int row, col;
Public :
 matrix(int, int);
Void read_elements( );
 Void print_elements( );
};
matrix :: matrix (int r,int c)
  {
     row=r;
     col=c;
```

```
a= int *[r] ;
For (int i= 0; i<r; i++)
*(a+i) = new int[c];
}
Void matrix :: read_element()
{ For(int i=0; i<row; i++)
   For(int j= 0; j<col; j++)
   cin >> (*(a+i)+j);

}
Void matrix :: Print_elements()
[   For(int i=0; i<row; i++)
  { For(int j=0; j<col; j++)
      cout<<"\t"<<(*(a+i)+j);
      cout<<endl;
   }
}
int main()
{ matrix matrix1(2,2);
 matrix1.read_elements();
 matrix1.Print_elements();
matrix matrix2(3,2);
matrix2.read_elements();
matrix2.Print_elements();
 getch();
   return 0;
}
```

Explanation of above program :-

```
int main()
{ matrix matrix1(2,2);
  matrix1.matrix(2,2)
```



Constructor overloading :-

↳ Defining more than one constructor within class by changing
1. Number of parameters
2. Types of "
3. Order "
is called constructor overloading.

↳ constructor is overloaded in order to extend the functionality of existing constructors.

P59 // constructor overloading

```
#include<iostream>
#include<conio.h>
using namespace std;
class array
{   int *a;
    int size;
 Public:
   array();
   array(int);
```

```
Void read_elements();
Void Sum();
};
array::array()
{  size=5;
   a= new int[size];
}
array::array(int s)
{
    size =s;
   a= new int[size];
}
Void array::read_elements()
{  For(int i=0; i<size; i++)
     cin>>a[i];
}
Void array::Sum()
{  int s=0;
  For(int i=0; i<size; i++)
   s=s+a[i];
cout<<"\n sum is "<<s;
}
int main()
{  array players;
   Players.read_elements();
     Players.Sum();
    array std1(3);
  std1.read_elements();
    std1.Sum();
getch();
 return 0;
}
```

Pgm. // constructor overloading

```
#include<iostream>
#include<conio.h>
#include<string.h>
using namespace std;
class student
{
   char name[10];
   char course[10];
    float Fees;
public:
   student(char*, char*);
   student(char*, char*, ~~Fee~~ float);
void print_student();
};
student::student(char *n, char *c)
{   strcpy(name,n);
    strcpy(course,c);
      Fess=0;
}
student::student(char*n,char*c,float F)
 {
   ~~cout<<"\n Name~~
        strcpy(name,n);
        strcpy(~~name~~course,c);
          float=F;
    }
void student::Print_student()
{   cout<<"\n name"<<name;
    cout<<"\n course"<<course;
    cout<<"\n fees"<<fee;
}
```

```
int main()
{ student student1("srikanth", "c++");
  student student2("[redacted]", "c++", 500);
  student1.print_student();
  student2.print_student();
  getch();
  return 0;
  }
```

copy constructor :-

1. copy constructor is parameterized constructor.
2. A copy constructor is used to initialize an object from another object.
3. copy constructor is parameterized constructor having reference type parameters
   - ↳ It is having parameters of type class.
   - ↳ Copy constructor create copy of an object.

Pgl create object by copying contents from existing object.

```
#include <iostream>
#include<conio.h>
using namespace std;
class triangle
{
    float base, height;
public:
triangle (float, float);
triangle (triangle &);
  void find_area();
};
```

```
triangle :: triangle (Float b, Float h)
{
    base = b;
    height = h;
}
triangle :: triangle (triangle &t)
{
    base = t.base;
    height = t.height;
}
Void triangle :: Find_area ()
{
  Flocut area = 0.5 * base * height;
  cout << "\n area is = " << area;
}
int main()
{
  triangle triangle1(1.5, 2.5);
  triangle triangle2( triangle1);
 triangle1. Find_area ();
 triangle2. Find_area ();
  getch ();

  return 0;
```

P62. // copy Constructor

```
#include<iostream>
#include<conio.h>
using namespace std;
class Time
{
```

```
                                        SS
 int ~~dd, mm~~ hh, mm, ~~ss~~;
Public:
Time (int, int, int);
 Time(Time &);
Void set_hh (int);
Void set_mm(int);
 Void set_ss (int);
Void print_time ();
};

Time :: Time(int h, int m, int s)
 {
     hh = h;
     mm = m;
     SS = S;
}
Time :: Time (Time &t)
 {   hh = t.hh;
     mm = t.mm;
    SS = t.ss;
  }
 Void Time :: set_hh(int h)
  {
      hh = h;
  }
Void Time :: set_mm (int m)
 {
    mm = m;
}
Void Time :: set_s(int s)
  {   SS = S;
  }
```

```
void Time :: Point_time();
{
   cout <<"\n hh = " <<hh;
   cout <<"\n mm = " <<mm;
   cout<<"\n ss = " <<ss;
}
int main()
{
   Time time1(10, 5, 20);
   Time time2(time1);
   time1.Point_time();
time2.set_hh(12);
   time2.PrintF_time();
   getch();
   return 0;
   }
```

Defining constructors with default arguments or Parameters

P63

```
#include <iostream>
#include<conio.h>
# include <string.h>
using namespace std;

class account
{ int accno;
   char name[10];
 float balance;
Public:
account(int a, char *n; Float b=0)
{
```

```
accno = a;
Strcpy (name, n);
balance = b;
}
Void print_account()
{
cout << " Account no" << accno;
cout << "\n customer name" << name;
cout << \n Balance" << balance;
}

};
int main()
{
account account1(101, "Ramu", 5000);
account account2(102, "Vinod");
account1. Print_account();
account2. Print_account();
getch();
return 0;
}
}
```

or

Destructor :-

↳ Destructor is a special member function used For deallocating resources which are allocated by constructor.

↳ Destructor is executed before object is deleted or destroyed by O.S or program.

Properties OF destructor :-

1. destructor name is same as class but prefix with ~ (tild operator).

2. There is only one destructor within class
3. It is without any parameter.
4. It doesn't have return type not even void.

Syntax :-

```
~ destructor-name ()
{
    statements;
}
```

P64 // Destructor explanation ~~use~~ with help of an program.

A:-

```
#include<iostream>
#include<conio.h>
using namespace std;
class array
{
    int *a;
    int size;
Public:
    array(size);
    Void read_elements();
    Void Print_elements();
    ~array();
};
array :: array(int s)
{
    size=s;
    a= new int[size];
}
```

```
Void array :: read_elements()
 {
 For(int i=0; i<size; i++)
  cin>>a[i];

}
Void array :: Print_elements()
 {
  For(int i=0; i<size; i++)
   cout<<a[i]<<endl;

 }

~~Void~ array ::~array()

{
   delete a;
cout<<"\n Inside destructor";

}
 int main()
 {
    array array1(3);
 array1. read_elements();
 array1. Print_elements();

   getch();

   return 0;

   }
```

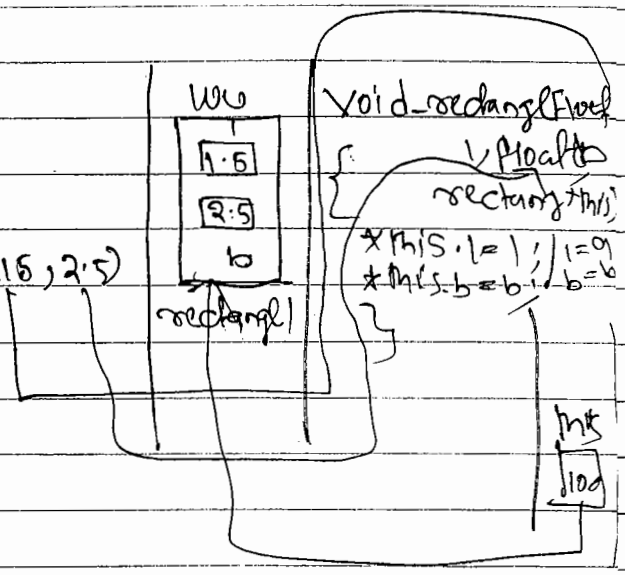
*this :- 1. this is a keyword in c++.
2. It hold the address of current object.
3. Every non-static member Function of a class having default parameter is called *this.
IF local variables and data members declared with same name, data members of current object accessed within function using *this.

P65:-

```
#include<iostream>
#include<conio.h>
using namespace std;
class rectangle
{
    Float l,b;
Public:
 Void set_rectangle (Float ,Float);
 Void Find_area();
};
Void rectangle
Void rectangle :: set_rectangle(Float,l Float b)
 {
   (*this).l = l ;
   (*this).b = b ;
 }
Void rectangle :: Find_area()
{   Float area = l*b;
    cout << "\n Area is " << area;
}
   int main()
 {     rectangle rect1;
       rect1.set_rectangle (1.5, 2.5);
       rect1.find_area();
       getch();
       return 0;
}
```

Explanation :-

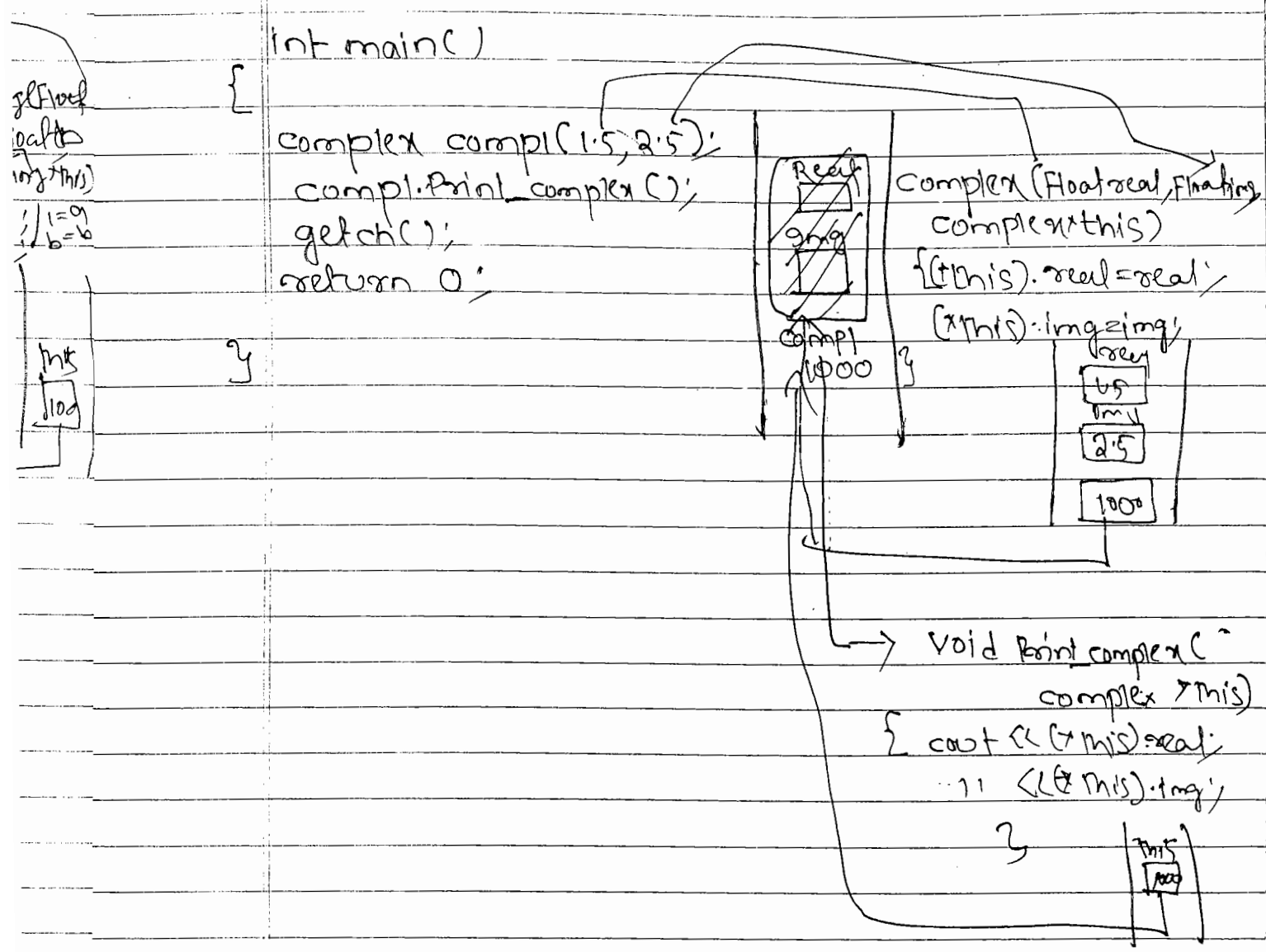
```
int main()
{
  rectangle rectangle1;
   rectangl 1=1.5 (X)

rectangl1.set_rectangle(1.5, 2.5)
```



P66-

```
#include <iostream>
#include <conio.h>
   using namespace std;
class complex
{
    Float real, img;
Public :
complex (Float, Float);
void Print_complex( );
};

complex :: complex (Float, Float)
```

```
{  (*this).real = real;
   (*this).img = img;
}
void complex :: Point_complex()
{
  cout << "\n real" << real;
  cout << "\n img" << img;
}

int main()
{
complex compl(1.5, 2.5);
 compl.Print_complex();
 getch();
return 0;
}
```



Static members :-

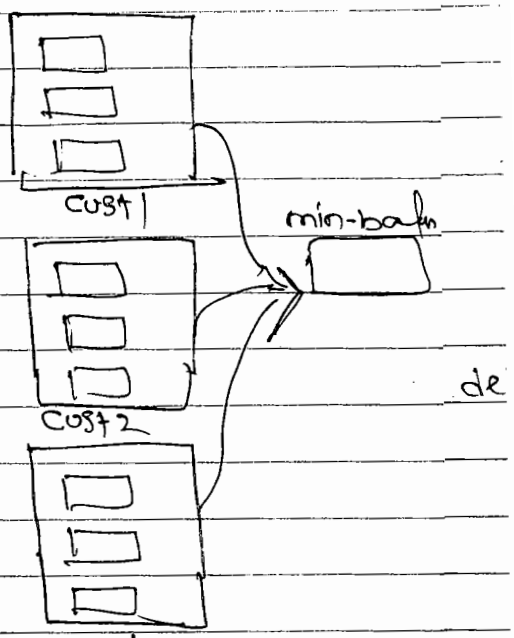
There are 2 types

1. Static Data members
2. Static member functions

1. Static Data member :-

(i) If data member is declared inside class with static keyword is called static data member.

Ex
```
class account
{   int accno;
    char name[20];
    float balance;
    static float minbalance;
};
account cust1;
account cust2;
account cust3;
```



(ii). It is global member, which is global to one or more than object.

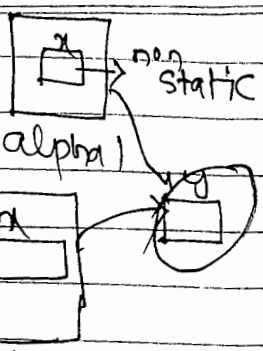
iii. This data member memory is allocated/created when first object of class is created or when it is accessed using class name.

(iv) Memory for this data member is allocated within global data area.

(v). This data member is accessed with class name without creating object.

Ex
```
class alpha
{
    public
    int x;
    static int y;
};
```

alpha.y=10 ✓ alpha alpha1;
alpha.x=20 ✗ alpha alpha2;



(vi.) Static means only one copy.
(vii) One copy of data member is created, which is accessed by more than one object.

Syntax :-

```
Static datatype variable-name ;
```

definination of static data member must be given outside the class.

```
<datatype>
<class-name> :: variable-name = [value];
```

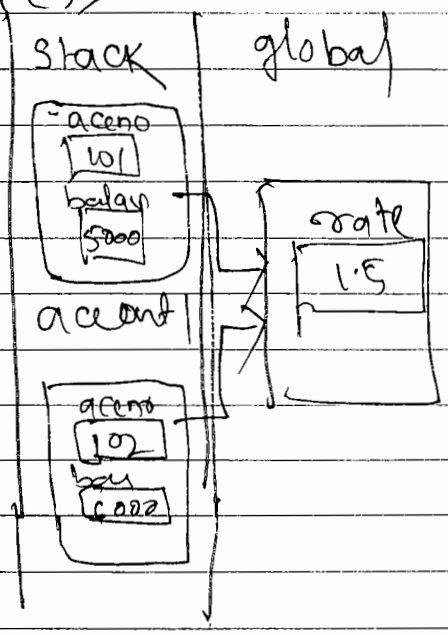
default values:- int, short, long → 0
Float, double → 0.0
char -- \0 (null)

P67. //

```
#include<iostream>
#include<conio.h>
class account
{ int accno;
  Float balance;
 Static Float rate;
Public:
  account(int, Float);
  Void deposit(Float);
  Void print_account();
};
```

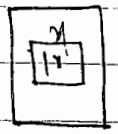
```
account :: account(int accno, float balance)
{
    (*this).accno = accno;
    (*this).balance = balance;
}
Void account :: deposit(Float tamt)
{
    balance = balance + tamt;
}
Void account:: Print_account()
{ cout << "\n Account no" << accno;
  cout << "\n balance" << balance;
  cout << "\n rate" << rate;
}
Float account :: rate = 1.5;
int main()
{ account account1(101, 5000);
  account account2(102, 6000);
  account1. Print_account();
  account2. Print_account();
  getch();
  return 0;
}
```

o/p →

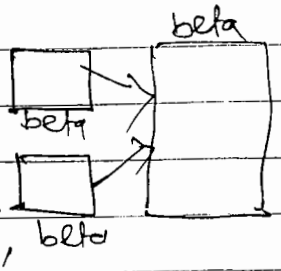


Ex

```
class alpha
{
 Public:
 int x;
};
 alpha-x (X)
 alphaalpha1;
alpha1.x=10;
```




```
class beta
{ Public:
  Static inty;
 };
int beta :: y
beta.y = 10;
beta beta1;
beta beta2;
beta1.y=30;
beta 2.y=40;
```



P68 Find output?

```
#include<iostream>
#include<conio.h>
using namespace std;
class alpha
{ Public:
  int x;
Static inty;
};
int alpha :: y;
int main()
{
  alpha:: y=10;
alpha alpha1;
  alpha1.x=10;
  alpha1.y=20;
alpha alpha2;
 alpha2.x =30
  alpha2.y=40;
alpha alpha3;
   alpha3.x=50;
    alpha3.y=60;
```

```
cout << alpha1.x << endl;
cout << alpha1.y << endl;
cout << alpha2.x << endl;
cout << alpha2.y << endl;
cout << alpha3.x << endl;
cout << alpha3.y << endl;
getch();
return 0;
}
```

O/P → alpha1.x = 10
60
30
60
50
60

2. Static member Function :-
   i. defining a member Function inside class with static keyword.
   ii. Like static data members we can also have static member Functions.
   iii. A static member Function can have access to only other static members declared in the same class.
   iv. A static member Function can be called using the class name.
   class name :: Function name;

Syntax:
```
static return-type
Function-name (parameters)
{ Statements;
}
```

Q Describe the main characteristics of static functions?

A:- The main characteristics of static functions include,

1. It is without the this pointers.
2. It can't directly access the non-static members of its class.
3. It can't be declared const, volatile or virtual.
4. It doesn't need to be invoked through an object of its class, although for convenience it may.

Pg. // counting objects:-

```
#include<iostream>
#include<conio.h>
#include<string.h>
using namespace std;
class student
{
 int rno;
 char name[10];
 static int count;
Public:
Student(int, char*);
static void print_count();
void Print_Student();
};
Student::Student(int r, char *n)
{  rno=r;
   Strcpy(name, n);
 count= count+1;
}
Void Student:: print_Student()
{
```

```
cout<<"\n Roll no"<<rno;
cout<<"\n name"<<name;
}
Void student :: Print_count()
{ cout<<"\n count of students"<<
}
int main()
{
int student :: count;
int main()
{ student :: Print_count();
  student stud1(101, "Mike");
  student stud2(102, "Ethan");
  Stud1. Print_student();
  Stud2. Print_student();
  student :: Print_count();
  getch();
  return 0;
}
```

constant members:

1. Constant data member
2. constant member Functions

1. constant data members:- data members of class can be declared as constant.

ii Constant data member value is never changed.

syntax :-

```
const <datatype> member-name;
```

Constant data member must be initialize using constructor list.

P70. // constant data member

```
#include<iostream>
#include<conio.h>
using namespace std;
{ class circle
    Float r;
const Float pi;
public:
 circle() : Pi(3.14)
 {
   r = 0;
}
Void Find_area()
{ Float area = Pi * r * r;
cout << "\n Area is " << area;
 } };
 int main()
{   circle circ1;
    circle1.set_r(2.5);
    circle1.Find_area();
 getch();
    return 0;
   }
```

Void set_r (Float)
{ (*this).r = r;
}

ed.

P71.

```
#include<stdio.h>
#include<conio.h>
using namespace std;
class circle
{   Float r;
    const Float Pi;
```

```
Public:
circle(Float r, Float P) : Pi(P)
 {
     (*this).r=r;
 }
  Void Find_area()
{ Float area = Pi*r*r;
cout<<"\n Area is"<<area;
 }
   };

int main()
{ circle circle1(1.5, 3.147);
    circle1.Find_area();
 getch();
  return 0;
}
```

2. Constant member Function :-
   1. Constant member Function is an accessor Function.
   2. It is a member Function which doesn't allow to change values of datamembers.

Syntax :- `returntype Function-name([parameters]) const`

```
{
    Statements;
}
```

Ex :-
```
class A
{ int x;
 Public:
 Void Fn1() -> modifier
{ x=10;
}x=x+10;
```

Accessor function

```
Void Fun2() const
{ x=20; X
cout<<x;
int y=x;
 x++ X
}
```

P72 //

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class marks
{
int sub1, sub2;
Public:
Void set_marks(int sub1, int sub2)
{
   (*this).sub1=sub1;
   (*this).sub2=sub2;
}
Void Find_result() const
{ if(sub1 40 || sub2<40)
  cout <<"\nResult Fail";
 else
cout<<"Result pass";
}
};
 int main()
{  marks stud1;
  stud1.set_marks(60, 70);
   stud1.Find_result();
   getch();
   return 0;
 }
```

class reusability :-

object oriented application is collection

of classes.

2. object oriented programming (C++) allows to use the contents of one class inside another using two methods.
   1. Composition (has-a) or 1. composition
   2. Inheritance (is-a) 2. Aggregation
                         3. Inheritance

1. composition :- (has-a relation) [container and contained]
   1. ex:-

```
class battery      class sim      class mobile → container
  {                  {              { sim airtel; } - contains
                                      battery b;
};                 }              }
```

↳ It is a process of creating an object of one class inside another class.
↳ The lifetime of contained object is as unthill container object exist.

P73.
```
// composition
#include <iostream>
#include <conio.h>
using namespace std;
class engine
{
  Public:
  Void start()
  {
    cout<<"\n Engine start";
  }
};
class car
  {
```

```
    engine e;
Public:
Void start()
 {
    e.start();
cout<<"\n car started ...." ;
 }
};
   int main()
 {  car car1;
   car1.start();
getch();
   return 0;
 }
```

P74 // composition ex 2

```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{   int rno;
    char name[20];
Public:
  Void read_student()
  {
    cout<<"\n Input rno";
    cin>>rno;
   cout<<"\n Input name";
   cin>> name;
}
  Void Print_ student()
  {
```

```
cout<<"\n Rollno"<<rno;
cout<<"\n Name"<<name;
} };
class marks
{ int sub1, sub2;
   Student stud;
  Public:
 Void read_marks(){
  Stud.read_student();
cout<<"\n input two subjects";
  cin>>sub1>>sub2;
}
Void Find_result(){
Stu.Print_student();
if (sub1<40 || sub2<40)
cout<<"\n Result Fail";
else
cout<<"\n Result pass";
} };
int main()
{     mark Student1;
      Student1.read_mark();
       Student2.read_result();
  getch();
    return 0;
   }
class A
{  int x;
  public:
     void read-x()
       { cin>>x;}
      int get-x(){ return x;}
}
```

```
class B
 {
   int y;
   A obja;
  public:
   void get( ){  cin>>y;
                 obja.read_x( );
               }
   void sum( )
        {
          int s = y + get_x( );
          cout<<"sum is" <<s;
        }
 }

int main( )
{
    B objb;
  objb.get( );
  objb.sum( );
  return;
}
```

Q. What is Aggregation ?

Aggregation is the relationship between the whole and apart. we can add/substract some properties in the part (slave) side. It won't affect the whole part.

ex:- Best example is car, which contains the wheels and some extra parts. Even though the parts are not there It is called as a car.

↳ Aggregation is the special type of composition

↳ In Aggregation contained object is send to container.

↳ In this contained object is exist independent of container object

P75.

```
#include<iostream>
#include<conio.h>
using namespace std;
class A
{
 Public:
int x;
 A (int x)
 {
   (*this).x=x
 }
};
class B
{  int y;
 Public:
  B(int y)
{
  (*this).y=y;
}
Void sum(A*a)
 {  int S= (*a).x+y;
     cout<<"sum is"<<S;
   }
};
```

```
int main()
{ Aobja(10);
  Bobja(20);
 objb.sum(*obja);
 getch();
  return 0;
}
```

↳ If member-function having parameters of type class, it receive object.

↳ If member function having parameters of class pointer it receive address of object.

P76.

A:-

```
#include<iostream>
#include<conio.h>
using namespace std;
class address
{
  char street[10];
  char city[10];
Public:
void read_address()
{
cout<<"\n enter street";
 cin>>street;
cout<<"\n enter city";
cin>>city;
}
void Print_address()
{ cout<<"\n street"<<street;
  cout<<"\n city"<<city;
};
```

```
class person{
char name[10];
 address *add;
Public:
Person(address *a)
{ add=a;
 }
Void read_person()
{
 cout<<"\n enter name";
   cin>>name;
  (*add).read_address();
 }
}; int main()
{ cout<<"\n enter name";
   cin>>name;
  (*add).
  address add1;
 Person person1(&add1);
 Person1.read_person();
 Person1.Print_person();
  getch();
    return 0;
}
```

Srikanth

P.T.O.

```
#include<iostream>
#include<conio.h>
using namespace std;
class matrix
{  int a[2][2];
Public:
 Void read_matrix()
```

```
{ cout<<"Enter elements";
For(int i=0; i<2; i++)
For(int j=0; j<2; i++)
    cin>> a[i][j];
}
Void print_elements()
{
 cout<<"\n Elements are \n";
 For (int i=0; i<2; i++)
 {
   For (j=0; j<2; j++)
     cout<<a[i][j]<<"\t";
   cout<<endl;
 } }
matrix add_matrix(matrix m2)
 { matrix m3;
 For( int i=0; i<2; i++)
 For( int j=0; j<2; j++)
 m3.a[i][j] = a[i][j] + m2.a[i][j];
   return m3;
  }
};
int main()
{
  matrix matrix1, matrix2, matrix3;
   matrix1.read_elements();
   matrix2.read_elements();
matrix3 = matrix1.add_matrix(matrix2);
 matrix1.Print_elements();
matrix2.Print_elements();
matrix3.Print_elements();
   getch();
    return 0;
  }
```

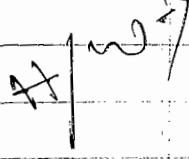
Geekanshu

If member Function having return type as class it returns object.

Pg. Wap to compare marks of two students.

A.
```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{  int sub1, sub2;
Public:
Void read_marks()
{ cout<<"\ninput two sub marks";
  cin >> sub1 >>sub2;
 }
Void compare marks (student s2)
{
if(sub1==s2.sub1 && sub2 ==s2.sub2)
 cout<<"\n marks are equal";
else
cout<<"\n marks are not equal";
}
};
int main ()
{  student stud1, stud2;
 stud1.read_marks();
 stud2. read_ marks();
stud1. compare_marks (stud2);
   getch();
     return 0;
 }
```

class Pg. Wap to add two complex numbers
(Home work)



Friend ~~class~~ Function:-

A non-member Function cannot have access to the private data of a class.
There could be a situation, where two classes have to share a particular Function.
In such situations c++ allows a ~~Function~~ common Function made ~~Friend~~ Friendly with both the classes.

1. A Friend Function declaration must be preceeded by the keyword Friend.
2. It is not in the scope of the class to which it has been declared as Friend.
3. It is not called using the object of that class.
4. It can be invoked like a normal Function without any object.
5. It cannot access the member names directly.
6. It has to use an object name dot membership operators and member name
7. It can be declared either in Public or private part of the class without affecting its meaning.

Syntax:- friend return-type Function name

Pgm

```
#include <iostream>
#include <conio.h>
using namespace std;
class alpha
{ int x;
Public:
   alpha(){
     x=10; }
friend void sum(alpha, beta);
    };
```

```
class beta
{ int y;
Public:
beta(){
 y=20; }
friend void sum(alpha, beta); };
void sum(alpha, beta)
{ int S= a.x+b.y;
 cout<<"\nsum is"<<S;
}
int main()
{  alpha alpha1;
     beta beta1;
 sum(alpha1, beta1);
 getch();
  return 0;
} o/p →
```

→ usually it has objects as arguments.
A Friend Function can be called by reference.
It should be be used only when absoolteg necessary, because such Function alters the values of the private members which is against data Hiding.

* Friend Fnctions are OFten used in operator ovr loading.

P8)

```
#include<iostream.h>
#include<conio.h>
using namespace std;
```

```
class marks;
class student
{  int rno;
    char name[10];
Public:
Void read_student()
{
 cout<<"\n Input rollno";
  cin>> rno;
 cout<<"\n Input name;
   cin>> name;
} Friend void Find_result(student, marks)
  };
class marks
{  int sub1, sub2;
 Public:
void read_marks(){
 cout<<"\n Input two subjects marks";
   cin>>sub1>>sub2;
}
friend void Find_result(student, marks)
 };
void Find_results(student s, marks m )
{
    cout<<"\n Roll no"<<s.rno;
    cout<<"\n Name "<<s.name;
 if(m.sub1<40|| m.sub2<40)
 cout<<"\n Result Fail";
  else
 cout<<"\n Result pass";
    }
    int main()
    {
```

```
Student Stud1;
marks Stud1_marks;
Stud1.read_student();
Stud1_marks.read_marks();
Find_result(Stud1 Stud1_marks);
    getch();
    return 0;
}
```

Friend class :-

1. A Friend of a class can be Function or class having Full access rights to use Private members.
2. members of one class can be access private members of the another class using Friend.

Syntax :-

```
Friend class class-name;
```

Pg2

```
#include<iostream>
#include<conio.h>
using namespace std;
class A
{
 void Fun1(){
cout<<"\n inside Fun1 OF class A";
}
Friend class B;
};
class B
{  A obja;
 Public:
```

```
void Fun2 ()
{ obja.Fun1();
 cout<<"\n Inside Fun2 OF class B";
   }
};
int main()
{ B objb;
  objb.Fun2();
 getch();
  return 0;
 }
```

P83. //

```
#include<stdio.iostream>
#include<conio.h>
using namespace std;
class alpha
{ int x;
Public:
   alpha(int x){
 (*this).x=x;
}
friend class beta;
};
class beta
{ int y;
Public:
 beta(int y) {
 (*this).y=y;
}
  void add(alpha a)
  { int s= a.x+y;
    cout<<"\n sum is "<<s;
  }
```

```
    void sub(alpha a)
  { int d = a.x+y;
    cout<<"\n DIFF is "<<d;
   } };
int main()
{   alpha alpha1(10);
     beta beta1(5);
    beta1.add(alpha1);
    beta1.sub(alpha1);
    getch();
     return 0;
  }
```

Operator Overloading (Polymorphism)

1. Polymorphism allows to define one thing in many forms.
2. Poly means many and morphism is forms, defining one thing in many forms is called polymorphism.

Function overloading allows to write more than one function with same name with different implementations.

a. what is operator overloading?

1. The mechanism of giving special meanings to an existing operator is known as operator overloading.
2. This is one of the existing features of c++
3. This technique has enhanced the power of extensibility of c++.
4. This technique permits to add two user-defined variables with the same syntax that is applied to basic types.

Provides a new definition of most of the C++ operators.

↳ Overloaded operator is special function whose name is operator symbol.
↳ existing operators performs operations on predefined datatypes.
↳ Operators overloaded inorder to perform operations on userdefined datatypes.

Syntax :- <returntype> operator <operator-symbol>(<Parameters>)
{
    Statements;
}

operand

We can overload all C++ operators expect
1. class member access operator (., .*)
2. Scope resolution operator (::)
3. sizeof operator (sizeof)
4. conditional operator (?:)

Rules For overloading operators:
1. Only existing operator can be overloaded.
2. The operand must have one operator of user defined datatype.
3. New operator cannot be created.
4. We cannot change basic meaning of an operator.

P84. // adding two complex numbers by overloading + operators.

A.

```
#include <iostream>
#include <conio.h>
using namespace std;
class complex
{ float real, img;
public:
void read_complex(){
cout<<"\n input real";
cin>>real;
cout<<"\n input img";
cin>>img; }
void print_complex(){
cout<<"\n Real"<<real;
cout<<"\n img"<<img;
}
complex operator +(complex c2)
{
complex c3;
c3.real = real + c2.real;
c3.img = img + c2.img;
return c3;
};
int main()
{ complex compl, comp2, comp3;
compl.read_complex();
comp2.read_complex();
comp3 = compl + comp2;
compl.Print_complex();
comp2.Print_complex();
comp3.Print_complex();
getch();
return 0;
}
```

Overloading Binary operators:-

1. A binary operator takes 2 operands
2. Function overloading binary operator with one explicit argument.

Ex- Overloading binary + operator

test operator + (test obj);

IF test is name of the class and obj1, obj2 and obj3 are the objects of test then the expression

obj3 = obj1 + obj2.
(obj1 → implicit, obj2 → explicit)

obj1 invokes the function and obj2 will be passed as an argument.

The function returns the sum of obj1 and obj2 and is assigned to obj3.

P.85 // Overloading binary + operator for concatenating 2 strings?

```
#include <iostream.h>
#include<conio.h>
#include <string.h>
using namespace std;
class string
{ char str[20];
 Public:
 void read_string()
{
cout<<("\nInput string";
 cin>>str;
} void Print_string()
   { cout<<"\n string is "<<str;
    }
```

capital letter in total program.

Pg

```
String operator + (String s2)
{
    String s3;
    strcpy(s3, str, str);
   strcat(s3, str s2, str);
    return s3;
 } };
int main()
{
String String1, String2, String3;
 String1.read_string();
 String2.read_string();
String3 = String1 + String2;
String1.Print_string();
String2.Print_string();
String3.Print_string();

  getch();
   return 0
   }
```

P&6. // overloading binary - operator for sub two matrices.

```
#include<iostream>
#include<conio.h>
 using namespace std;
 class matrix
{   int a[2][2];
   Public;
  Void read_matrix();
  Void Print_matrix();
  matrix
```

```
matrix operator - (matrix);
};
void matrix :: read_matrix()
{ cout<<"\n Input elements";
  For(i=0; i<2; i++)
  For(j=0; j<2; j++)
   cin >> a[i][j];
}
void matrix :: Print_matrix()
{ cout<<"\n elements are \n";
   For (int i=0; i<2; i++)
  {
    For(int j=0; j<2; j++)
    cout<<"\t"<<a[i][j];
    cout<<endl;
}}
matrix matrix :: operator - (matrix& m2)
 {
      matrix m3;
  For(int i=0; i<2; i++)
   For(int j=0; j<2; j++)
m3.a[i][j] = a[i][j] - m2.a[i][j];
   return m3;
}
int main()
{ matrix, matrix1, matrix2, matrix3;
  matrix1.read_matrix();
  matrix2.read_matrix();
  matrix3 = matrix1 - matrix2;
matrix1. Print_matrix();
 matrix2.   //
 matrix3.   >>
        getch(); return 0; }
```

Overloading unary operator :-
A unary operator take one operand.
Overloading unary minus operator
A unary minus operator changes the sign of an operand when applied to a basic datatype.
It should change the sign of datamembers when applied on an object.
The function directly access the members of the object.

P87. // overloading unary ++, -- operators.

Ans:-
```
#include <iostream>
#include <conio.h>
using namespace std;
class integer
{   int n;
  Public:
    integer (int n)
    {  (*this).n = n;
    }
  void operator ++()
    {
    }  n=n+5;
  void operator --()
  {  n=n-5;
    }
  void print(){
  cout<<endl<<n;
};
    int main()
  {  integer i1(10);
        i1.Print();
          i1++;
```

```
il.Print();
il--;
il.Print();
++il;
il.Print();
--il;
il.print();
getch();
return 0;
}
```

P88. // overload ++, -- operators For converting string to uppercase and lowercase.

```
#include<iostream>
#include<conio.h>
#include<string.h>
Using namespace std;
class String
{
    char str[10]
    public:
      String(char str)
      {
      (*this).str= str;
      }

String operator ++()
{
    for(int i=0; str[i]!='\0';
```

```
#include <iostream>
#include

using namespace std
class string
{
  char str[10];
Public:
Void read_string()
{ cout<<"\n Input String";
  cin>> str;
}
Void Point_string()
{ cout<<"\n string is"<<str;
}
  Void operator ++()
  { strupr(str);
}
 Void operator --()
{ strlwr(str);
}
};
  int main()
{ string name;
 name.read_string();
 name.Point_string();
   ++name();
 name.Point_string();
   --name;
name.Point_string
   getch();
    return 0;
}
```

O/p : check out program output in dev c++ only.

1→ Operators overloading using Friend :-
operator is overloaded using Friend inorder to perform operations one more than one datatypes (class).

Inheritance :- 1. The process by which objects of one class can acquire properties of objects of another class
2. It provides the idea of reusability.
3. We can eliminate redundancy code extent the use of existing code.
4. The mechanism of deriving a new class from an old one is called inheritance, old class is referred to as base class and new one is derived class.
5. A class can inherit properties from more than one class or from more than one level.

Advantages :-
1. Reusability
2. extensibility

defining derived class :-
A derived class is defined by specifying relationship with the base class

General form

```
class derived : visibility mode base class
                & class
{
    mem_   derive__ .
};
```

colon indicates derived class derived form base class.

Types of inheritance supported by C++

1. Single level
2. Multilevel inheritance
3. Multiple "
4. Hierarchical "
5. Hybrid "

Srikanth

Syntax :- class
derived class : visibility mode base class
{ data members;
member Functions ;
};

Visibility mode :- 1. Visibility mode defines visibility or accessability of base class members within derived class.

2. The visibility mode may be either Public or Private and is optional.
3. The default visibility mode is private.
4. When the visibility mode is private, Public members of the base class becomes the Private members of derived class.
5. The public members of base class are accessible by the member Functions of the derived class only.

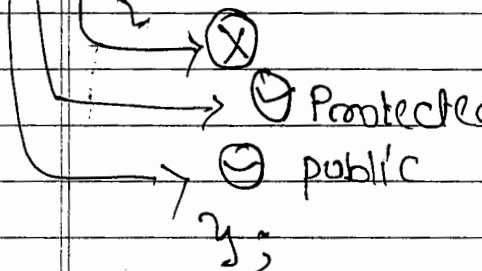
Ex:- class A
{ Private :
int x;
Protected ;
int y;
Public ;
int z;
};
class B : Private A
{ ⊗
Private
private
};

6. When the visibility mode is Public then Public members of base class becomes the Public members of the derived class.

7. They are accessible to the derived class objects.

8. In both cases Private members of the base class are not accessible to the derived class members Function.

Ex:-
```
class A
{ Private:
  int x;
  Protected:
  int y;
  Public;
  int z;
};
class B : Public A
{
```




```
};
```

Single level inheritance :- 1. A derived class with one base class is called Single level inheritance.

Ex:-
```
Class base
{
  ——
};
  class derived : Public base
  {
    . — ;
  };
```

```
P89 // single level inheritance
#include <iostream.h>
#include <conio.h>
using namespace std;
class person
{ char name[10];
Public:
Void read_name()
{ cout<<"\n input name";
  cin>> name;
}
Void Print_name()
{ cout<<"\n name is "<<name;
};
class customer : Public person
{ Float credit_limit;
Public:
Void read_credit_limit()
{
cout<<"\n input creditlist";
  cin >> credit_limit;
}
Void Print_credit_limit()
{ cout<<"\n credit limit "<<credit_limit;
} };
Int main()
{   customer cust1;
   cout<<sizeof(cust1)<<endl;
   cust1.read_name();
  cust1.read_credit_limit();
  cust1.Print_name();
  cust1.Print_credit_limit();
  return 0;
getch(); }
```

Protected:- A Protected member is accessible by the members function within its class and any class immediately derived from it.
2 It cannot be accessed by the function outside these two classes.

Q. what is difference between and Protected?

Private	Protected.
1. Private members accessible by the members of the same class.	1. Protected members accessible by the members of same and derived class, but cannot access by non members.

Pgm 11

```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{
Protected:
  char name[10];
Public:
 void read_name()
{ cout<<"\n Input name";
   cin>>name; }
};
 class mca_student :0 Public student
{  int sub1, sub2;
   Public:
     void read_subject()
```

```
{ cout<<"\n Input two subjects";
  cin>>sub1>>sub2;
}void Find_result()
 {
cout<<"\n Name"<<name;
if(sub1<40 || sub2<40)
cout<<"\n Result Fail";
else
cout<<"\n Result Pass";
}
};
int main()
{ mca_student stud1;
  stud1.read_name();
  stud1.read_subject();
  stud1.Find_result();
 getch();
 return 0;
}
```

constructors in inheritance :-

base class constructor bind with constructor of derived class.

OR

Derived class constructor calls the constructor of base class.

OR

Calling the base class constructor within derived class are two types

1. Implicit calling
2. explicit calling

compiler calls only non parameterized constructor of base class within derived class.

P.91 //

```
#include<iostream>
#include<conio.h>
using namespace std;
class alpha
{ Public:
   alpha()
  { cout<<"\n Inside constructor OF alpha";
  }
};
class beta : Public alpha
{ Public :
   beta()
 { cout<<"\n Inside constructor OF beta";
  }
};
  int main()
{   beta b1;
    getch();
     return 0;
   }
```

Explan

calling of base class constructor within derived class explicitly :-

P.92

```
#include<iostream.h>
  class base
  {
      Public:
      base(int x)
{ cout<<"\nParameterized const OF base class";
    } };
```

```
class derived : Public base
{  Public:
    derived()
   {  cout<<"\n Non parameterized constructor of derived
          class";
      }
void main()
{  derived d1;
  }
```

O/P - Errors,

Explanation: The above program display compile time error because there is no default constructor inside base class and no constructor of the base class is called explicitly within derived class.

* Overcome error call like this derived() : base(10)*

base class constructor is called explicitly with derived class as part of derived class constructor header.

class type P93

```
#include<iostream.h>
#include<string.h>
class employee
{
   char name[10];
    int empno;
  Public:
employee(int e, char n[])
   {
```

```
  empno = e;
strcpy (name, n);
}
Void Print_name()
{ cout << "\n Employee name" << name;
}
Void Print_empno()
{ cout << "\n Employee No" << empno;
}};
class salaried_employee: Public employee
{
Float salary;
Public:
Salaried_employee (inte, char n[], Float s); employee(e,n)
{
  Salary = s;
} Void print_salary()
{ cout << "\n Salary " << Salary;
}};
   Void main()
{
Salaried_employee empl(101, "abc", 6000);
   empl.Print_empno();
   empl.Print_name();
  empl.Print_Salary
      getch();
   }.
```

* constructors of any class are invoked only if an object of that class is constructed.
* if the base class constructor is having arguments, which assigns values to the datamembers of the base class, then it is mandatory to invoke the base class constructor.
* But the derived class object cannot invoke the base class constructor directly.
* The constructor of the derived class receives the entire list of the variables values and passes them to the base constructor.

P.94. // Find output?

```
#include<iostream.h>
class base
{ public:
base() {
cout<<"\n Non parameterized constructor of base class";
}
base(int x){
cout<<"\n Parameterized constructor of the base derived class";
derived(): base(10)
{ cout<<"\n Non parameterized constructor of derived class";
}};
void main()
{
derived d1;
derived d2(40);
}
```

(A)

```
#include<iostream.h>
class base
{ public:
base()
{ cout<<"\n non parameterized constructor of base class"; }
```

```
base(int x)
{ cout <<"\n parameterized constructor of base
  class";
}
};  class derived : public base
{ public:
  derived(int y)
{ cout <<"\n parameterized constructor of
derived class";
}
derived : base(10)
{
  cout <<"\n non parameterized constructor of
    derived class";
  }
  };  void main()
{ derived d1;
  derived d2(40);
}
```

O/P → Parameterized const of base class
Non " " " derived "
Non " " base "
Parameterized " " derived

P.95.

```
#include<iostream.h>
class account
{
 int accno;
 float balance;
public:
account(int accno, float balance)
  {
    (*this).accno = accno;
    (*this).balance = balance;
  }
```

P.9

```
Void deposit(Float tamt)
{ balance = balance + tamt;
}
Void Print_balance(){
cout<<"\n Balance is "<<balance;
};
class saving_account : Public account
{
    char check_fac;
Public:
Saving_account(int a, Float b, char c): account(a,b)
{ check_fac;
}
Void Print_check_Fac() {
cout<<"\n check Fac "<<check_Fac;
};
int main()
{ Saving_account acc1(101, 5000, 'y');
  acc1.deposit(2000);
  acc1.Print_balance();
  return 0;
}
```

Destructors in Inheritance :- (Bottom to Top)

* Destructor of derived calls the destructor of the base class.
* Calling of base class destructor within derived class is added as last statement.

P:96. //

```
#include<iostream.h>
class base
{ Public:
  base()
  { cout<<"\n inside no parameterized constructor of base class"; }
```

```
~base()
{ cout<<"\n Inside the destructor of derived class";
}
};
class derived : Public base
{ Public :
  derived()
  {
cout<<"\n Inside the non Parameterized of derived
  class";
  }
 ~derived()
{ cout<<"\n Inside destructor of derived class";
};
Void main()
{ derived d1;
}
```

Member Function Overloading :-

1. Redefining of base class member Function within derived class is called member Function overloading
2. Writting/defining member Function within derived class with same, number of parameters, types of parameters of member Function defined inside base class is called member Function overriding.
3. Member Function is override inorder to modify or extend Functionality of base class Function within derived class Function within derived class.
4. If derived class wants to have different imple of base class ~~wants to have~~ member Function, member Fnction is override.

```
class A
{ Public:                    -> Overriden Function
  Void Fun()
  { cout<<"Inside Function of base class";
  }
};
  class B : PublicA
  { Public:                  Overriding function
    Void Fun1()
    { cout<<"Overriding function";
    }
};
P.93 #include<iostream.h>
  class person
{ char name[10];
  Public:
  Void read()
{ cout<<"\n Input name";
    cin>>name;
}
  Void Print()
{ cout<<"\n Name is"<<name;
} };
class supplier : Public Person
{ int Suppid;
  Public:
  Void read()
{ Person :: read();
  cout<<"\n Input supplied";
  cin>> Suppid;
}
  Void print()
{ person :: Print();
```

```
cout<<"\n supplier id" <<suppid;
 }
};
 void main()
{ supplier supplier1;
  supplier1.read();
 supplier1.Print();
}
```

Pg8

```
#include<iostream.h>
class A
{ Public:
 int Fun1()
{ return 10;
}
 };
class B: public A
{ public:
 Float Fun1()
 { return 1.5f;
 } };
void main()
{ B obj1;
 cout<<obj1.Fun1();
}
```

Pg9

```
#include<iostream.h>
class dotted_line
{ Public:
  void draw()
{ For(int i=0; i<=40; i++)
    cout<<(" ");
  } };
```

```
class Star_line : Public dotted_line
{ Public:
Void draw()
{ For (int i=0; i<=40; i++)
 cout<<("*");
 } };
Void main()
{ dotted_line line1;
 line1.draw();
Star_line line2;
 cout<<("\n");
Line2.draw();
}
```

STATIC BINDING :- In Function overloading or operators overloading there will be more then one Function with the same name. The Functions are invoked by matching argument. The compiler selects the appropriate Function for a particular call at compilation time itself. This is called early binding or static binding. (compile time binding).

DYNAMIC BINDING :- 1. Selecting Appropriate ~~numbers~~ members Function while the program is running is known as runtime polymorphism. This process is known as lak binding. It is also known as dynamic binding binding as the selection of the Function is done dynamically at runtime.

2. Dynamic binding requires use of pointer to objects.
3. The Feature of runtime Polymorphism is the ability to refer to objects without regard to their classes using a single pointer.

4. We can use a pointer to a base class to refer all derived object.
5. When we use the same function name in both base class and derived class a base class pointer executes the base class function only even when it is assigned with derived class address.

* Declare a Pointer variable of base class.
* Assign object of derived class to base class Pointer

Ex-
```
class Animal
{ Public:
  Void walk()
{ cout<<"Animal walk";
  }
};
class dog: PublicAnimal
{ Public:
Void walk()
{ cout<<"dog walk");
}
};
class cat: public Animal
{ Public:
  Void walk()
{ cout<<"\n cat walk");
  }
};
```


```
Void Play
{
  (*).walk();
}
```

Advantage
1 Loosly coupled Application virtu

Virtual Function :- To invoke the derived class overriding member Function by base class pointer when it is assigned with derived class Address. The class member Fnction has to be made virtual Function.

to

2 both

s pointer

then

ss.

:

class

The keyword virtual has to be preceded to normal declaration.

Ex:- class Animal
{ Public:
void week()
}

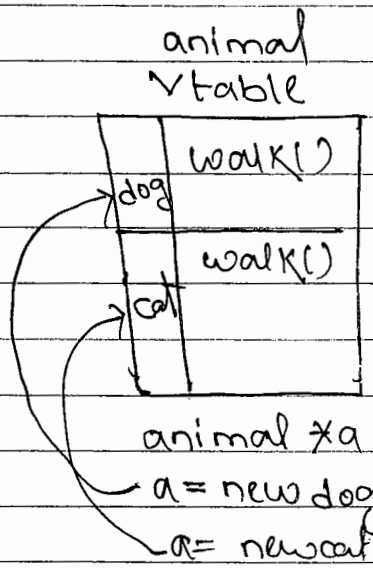
Syntax :- virtual returntype function-name(parameters)
{
---;
}

Rules For V.F :-

1. V.F must be a members of some class.
2. They cannot be static members.
3. They accessed by using object pointers.
4. A virtual Function can be a Friend of another class.
5. A V.F is a base class must be defined, even though it may not be used.

Prog// #include<iostream.h>
class animal
{ Public:
lication virtual Void walk()
{ cout<<"\n Animal walk";
}};
class dog : Public animal
{
Public!
rived Void walk():
ase { cout<<"\n Dog. walk");
h }};
Function class cat : Public Animal

```
{ Public:
void walk()
{ cout<<"\n cat walk");
}};
Void Play(animal *a)
{
 (*a).walk();
} void main()
{  dog dog1;
   cat cat1;
 Play(&dog1);
 Play(&cat1);
}
```



1. Virtual table is used by ~~virtual~~ base class Pointer to call overriding member function. Pure v

P:101 //

```
#include<iostream.h>
class creditcard
{ Public:
 void withdraw()
{  cout<<"\n withdraw of creditcard";
 }};
class sbi_creditcard : Public creditcard
 {   Public:
     void withdraw()
   { cout<<"\n I allow to withdraw 8500";
 }};
class hdfc_creditcard : Public creditcard
 { Public:
     void withdraw()
      { cout<<"\n et allow to withdraw 35000";
      }};
```

ABSTR

Syntax

```
class creditcard_machine
{ Public:
static void swap(creditcard *c)
 {
     (*c).withdraw();
}};
void main()
{
hdfc_creditcard h1;
 sbi_creditcard s1;
creditcard_machine:: swap(&h1);
creditcard_machine:: swap(&s1);

}
```

class

ction. Pure virtual Function :- A Function declared in a base class and has no definition relative to the base class.
They are redefined in derived classes. They are called as do-nothing Functions.

ABSTRACT BASE CLASS :- A class containing pure virtual function is called abstract base class.
They cannot be used to declare any objects.
It is purely for inheriting purpose only.
They are used to create a base pointer required to achieve runtime polymorphism.

Syntax :- `virtual returntype function-name(parameters)=0;`

);

35000);

Prog
```
#include<iostream.h>
class shape
{ Protected:
  Float dim1, dim2;
Public:
 Void readdim()
 { cout<<"\n Input two dim";
   cin>>dim1>>dim2;
 }
 Virtual Float Find_area() = 0;
 };
class triangle : Public shape
{ Public:
 Float Find_area()
  { return 0.5*dim1*dim2;
  }};
class rectangle : Public shape
 { Public:
   Float Find_area()
  { return dim1*dim2;
   }};
Void main()
{ triangle t1;
  rectangle r1;
 t1.read_dim();
 r1.read_dim();
Float area1 = t1.Find_area();
Float area2 = t2.Find_area();
cout<<"\n Area of triangle" <<area1;
cout<<"\n Area of rectangle" <<area2;
  }
```

Virtu

Hybri

P

```
// Runtime Polymorphism
 Shap *s;
 triangle t1;
 rectangle r1;
  s= &t1
(*s).read_dim();
Float area1 = (*s).Find_area();
s= &r1
(*s).read_dim();
Float area2 = (*s).Find_area();
cout<<"\n Area of triangle"
                <<area1;
cout<<"\n Area of red triangle "<<area2;
}
```

Virtual class :- Base class is inherited within derived class as virtual.

Hybrid inheritance :- IF there are two or more types inheritances design a program it is called hybrid inheritance.

ex class studen();
Classtest : Public student
{}; class sports : Public Student
{};
class result : Public test, Public sports;
Then this is also called as multi path inheritance as the class student inherited via test and also via Sports.
we can't achieve hybrid inheritance without virtual

```
Pr03 // hybrid inheritance
#include<iostream.h>
class A
{ Public:
 Void Fun1().
{ cout<<"\n Inside Function1 of A";
}};
class B : Public virtual A
{ Public :
 Void Fun2(){
cout<<"\n Inside Function2 of B";
}};          virtual
class C : Public A
 {
Public :
  Void Fun3 () {
cout<<"\n Inside Function3 of C";
 }};
```

```
class D:Public B, Public C
{ Public:
  void Fun4() {
cout<<"\n Inside Function 4 of D";
} };
Void main()
{ D obj;
obj. Fun1();
obj. Fun2();
obj. Fun3();
obj. Fun4();
  }
```

*→ In derived class if base class is virtual, derived class can't modify Functions of base class (Cannot override functions of base class)

P.104.

```
#include<iostream.h>
class Student
{
  Protected:
char name[10];
Public:
Void read_name()
{ cout<<("\n input name");
   cin>> name;
} Void Print_name()
{  cout<<("\n Name" << name;
} };
class test : Public Virtual Student
{  Protected:
   int sub1, sub2;
```

```
Public :
Void read_marks()
{ cout<<"Input marks";
cin>>sub1>>sub2;
};

class sports : Public virtual student
{ Protected:
    int h,w;
Public:
 Void read_sports()
{
 cout<<("\n Input h, w");
  cin>>h>>w;
}};
class result : Public test, Public Sports
{ ~~Public:~~
 int ~~tot~~ total;
  Float avg;
Public:
Void Find_result()
{  total = sub1 + sub2;
    avg= total/3;
cout<<"\n Total "<<total;
cout <<"\n Avg "<<avg;
cout <<"\n w "<<w;
cout<<"\n H" <<h; }};
Void main()
 {
  result stud1;
 stud1. read_name();
 stud1. read_marks();
 stud1. read_sports();
 stud1. Find_result(); }
```

Multi level Inheritance :- The mechanism of deriving a class from another derived class is known as multilevel inheritance.

Ex:-
```
class base
{ };
class derived1 : Public base
{ };
class derived2 : Public derived1
{ };
```

Multiple Inheritance :- If class is derived from more than one base class it is called multiple inheritance

Ex:
```
class base1
{ };
class base2
{ };
class derived : Public base1 , Public base2
{ };
```

~~Hybrid~~ Hierarchical Inheritance :- If a class is inherited by more than one class it is called hirarchical inheritance.

Ex:-
```
class student
{ };
class arts : Public student
{ };
class medical : Public student
{ };
```

Pgm5 // multi level inheritance

```
#include <iostream.h>
```

```
class Person
{
char name[10];
Public:
Void read_name() {
cout<<"\n Input name";
cin>> name; }
Void print_name () {
cout<<"\n Name"<<name;
}
class employee : Public Person
{
int id;
Public :
Void read_id() {
cout<<"\n Input id";
cin>>id; }
Void print_id() {
cout<<"\n Id"<<id;
}};

class worker : Public employee
{ int days;
  Float wage;
Public:
Void read_days_wage() {
{ cout<<"\n Input days";
  cin>>days;
    cout<<"\n input wage";
      cin>> wage; }
Void calc_salary() {
```

```
cout<<"\n Total salary "<<days*wage;
};
void main()
{ worker worker1;
worker1.read_name();
   ).read_id();
     ).read_days_wage();
     ).Print_name();
  ).Print_id();
    ).cal_salary();
}
```

Syntax

Templates :- (Generic)

Prac

1. Templates enables us to create generic classes and Functions.
2. A template can be used to create a family of classes or functions.
3. In a generic class or function, the type of data upon which the function or class operates is specified as a parameter.

Templates:

1. Function templates
2. Class templates

Generic Function or Function templates:-

1. Defines a general set of operations that will be applied to various types of data.
2. A single general procedure can be applied to a wide range of data.
3. Defines the nature of the algorithm independent of any data.
4. Compiler automatically generates the correct code for the type of data that is used at the time of compilation.

Ex-

Syntax :-

The general Form of generic function

```
template <class type>
return type Function-name (args)
    {
        // statements;
    }
```

Type is a place holder name For a datatype used by the Function. This may be used with in a function defination. The compiler automatically replaces it with an actual datatype when it create a specific version of the Function.

Prog. //

```
#include<iostream.h>
template<class T>
T max (Tx, Ty)
{   if (x>y)
      return x;
    else
      return y;
}
Void main()
{ int m1 = max(10, 20);
  Float m2 = max(1.5f, 2.5f);
  double m3 = max (4.5, 4.5);
cout<<"\n max OF two integers "<<m1;
cout<<"\n max of two Floats"<<m2;
cout<<"\n max of two double "<<m3;
}
```

Ex-

```
#include<iostream.h>
template <class T1, class T2>
void print(T1 x, T2 y)
```

```
{ cout<<"\n x="<<x;                                    Overloa
  cout<<"\n y="<<y;                                      *
} void main()
{ PrintF(10, 'A');                                      Pro
PrintF(1.5F, 20);
 printf(40, 50);
  PrintF(50, 60);
}
```

P.107
```
// Sorting elements of array :-
#include<iostream.h>
##in. template<class T>
void sort(T*a, ints).
{
   T temp;
For( int i=0; i<s; i++)
{ For( int j=0; j<s-1; j++)
{
  IF(a[j]>a[j+1])
  {  temp=a[j];
     a[j]=a[j+1];
    a[j+1]=temp;
  }}}
Void main()
{ int x[] = {30, 10, 20, 50, 40};
float y[] = (1.5, 2.5, 1.0, 0.5);
   sort(x,5);
   Sort(y,4);
For( int i=0; i<5; i++)
cout<<endl<<x[i];
   For(j=0; j<4; j++)
 Cout<<endl<<y[j];
   }
```

Overloading Function templates :-

* Defining more than one function template with same name by changing number of parameters.

Prog //

```
#include<iostream.h>
template<class T>
T max(Tx, Ty)
{  if (x>y)
   return x;
   else
      return y;
}
  template<class T>
 T max(Tx, Ty, Tz)
{  if (x>y && x>z)
     return x;
    else
     if(y>z)
     return y;
    else
      return z;
  }
    Void main()
    {  int m1= max(10, 20);
       int m2= max(50, 40, 20);
       Float m3 = max(1.5, 2.5);
    Float m4 = max(2.5, 3.5, 4.5);
cout<<"\n max of two integers"<<m1;
cout<<"\n max of two Floats"<<m3;
cout<<"\n max of three integers"<<m2;
Cout<<\n    "    "    Floats"<<m4;
   }
```

Generic class or class templates :-

1. Generic class are useful when class uses logic that can be generalized.
2. A class that defines all algorithm can be created
3. Actual type of data being manipulated will be specified as a parameterss when object are created.
4. For example the same algorithm that maintain a queue of integers will also work for queue of floats.

P109

Syntax :- The general Form of a generic class declaration

```
Templates <class type>
class class-name
{
  . . .
}
```

↳ The type name is a place holder type name which will be specified when a class is initialized. We can define more than one generic datatype.

↳ General Form of defining an object of generic class

```
class name <type> obj
```

↳ type is the type name of the data that the class will be operating upon

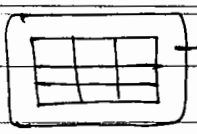
Ex

```
template <class T>
class matrix
{ T a[3][3];
};
Void main()
{   matrix <int> m1;
    matrix <float> m2;
```

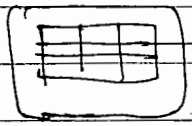
es logic

created will be

```
matrix<double> m3;
sizeof(m1);
sizeof(m2);
sizeof(m3);
}
```



M1



→ Float

Prog

intax,

```
#include<iostream.h>
template<class T>
class matrix
{ T a[2][3];
 Public:
Void read-elements()
{
cout<<"\n Input elements of matrix";
For(int i=0; i<2; i++)
For(int j=0; j<2; j++)
 cin>>a[i][j]; }
```

name

```
Void print_elements()
{ cout<<"\n elements are \n";
 For(int i=0; i<2; i++)
```

generic

```
{ For(int j=0; j<2; j++)
```

ric

```
  cout<<"\n"<< a[i][j];
    cout<<endl;
 }
 }
```

f the

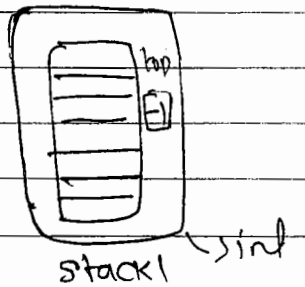
```
matrix<T> add_matrix(matrix<T> m2)
{  matrix<T> m3;
   For(int i=0; i<2; i++)
   For(int j=0; j<2; j++)
m3.a[i][j] =a[i][j]+m2.a[i][j];
};  Void main()
    { matrix<int> m1;
     m1.read_elements();
```

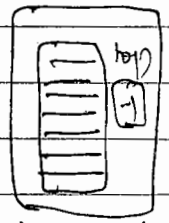
(STL library → Standard template library)

```
matrix<int> m2;
m2.read_elements();
matrix<int> m3 = m1.add_matrix(m2);
m3.Print_elements();
}
```

P110 // class template example

```
#include<iostream.h>
template<class T>
class stack
{
T a[10];
int top;
Public:
stack()
{ top=-1;
}
Void Push(T ele)
{ if(top>9)
cout<<"stack is overFlow";
else
   a[++top] = ele;
}
Void Pop()
{  if(top<0)
cout<<"\n stack is underFlow";
 else
  cout<<"\n element is "<<a[top--];
} };
```





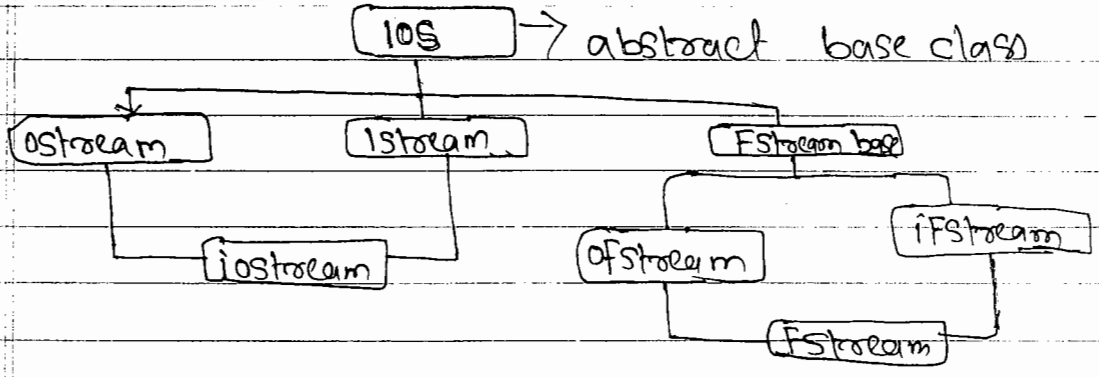
```
Void main()
{
 Stack<int> Stack1;
 Stack1.Push(10);
 Stack1.Push(20);
 Stack1.Push(30);
 Stack1.PoP();
 Stack1.PoP();
 Stack<Float> Stack2;
 Stack2.Push(1.5F)
 Stack2.Push(2.5F)
 Stack2.Push(8.1F)
 Stack2.PoP();
}
```

Iostreams and Files :-

Q. What is input ?
Data or information given to program is called input.

Q. What is ouput?
The data or information given by program is called output.

1. To perform input & output operations C++ Provide Stream classes.
   - cout
   - cin
2. Stream represents Input source or output destination



→ abstract base class

Forma

width

Cont. wi

ostream (output stream class)

This class provide Formatted and unformatted output Functions.

Functions of ostream class :-

1. Put() :-

Member Function of ostream class

cout.Put(ch); // display the values of the variable ch

P112

It can also be used to output a line of text characters by characters.

2. write() :- member Function of ostream class

write() displays the entire line

write(line, size);

Line represents the name at the String to be displayed.

Size indicates the no. at characters to display. It doesn't stop displaying the characters when NULL character is encountered.

P.111

//

Preci

```
#Include<iostream.h>
    Void main()
```

```
{ cout.Put('A');
 cout.write("object", 6);
cout.write("object", 3);
} o/p Aobjectobj
```

Formatted functions :-

width() To ~~display~~ specify the required field size for displaying an output.

```
cout.width(w);
```

The output will be printed in a field of w characters wide at the right end of the field.

Pgm //

```
#include<iostream.h>
void main()
{
int a=10, b=100, c=1000;
cout.width(5);
cout<<a<<endl;
cout.width(5);
cout<<b<<endl;
cout.width(5);
cout<<c<<endl;
}
```

```
o/p   10
     100
    1000
```

Precision ()

By default, the ~~function~~ floating are Printed with six digits after the point,

we can specify the no. of digits to be displayed after the decimal point.

```
cout.precision(d);
```

sets d number of digits to the right of decimal point.

P.113

```
//
#include <iostream.h>
Void main()
{
Float a=1.2345;
cout.Precision(2);
cout<<a<<endl;
cout.Precision(4);
cout<<a<<endl;
}
```

1st position, 2nd position

O/P 1.2
1.235

Fill():- The unused positions of the field are filled with white spaces by default. we can fill them by desired characters by using Fill() Functions.

```
cout.Fill(ch);
```

ch is used to Fill unused position.

P.114

```
//
#include <iostream.h>
Void main
{   int a=10;
    cout.width(5);
    cout.Fill('*');
    cout<<a<<endl;
}
```

O/P ***10

setf() :- cout.setf(ios::showpoint) displays trailing zeros
cout.setf(ios::showpos) displays the plus sign before a positive number.

P. 115

```
#include<iostream.h>
Void main()
 { Float a=5;
cout.setf(ios::showpoint);
cout<<a<<endl;
      }        O/P-  5.000000
```

P@116

```
#include<iostream>
 void main()
{  int a=-10;
     int b=10;
cout<<a<<endl;
Cout.setf(ios::showpos);
cout<<b<<endl;
     }
       O/p  -10
            +10
```

Filled
Fill
sing

get

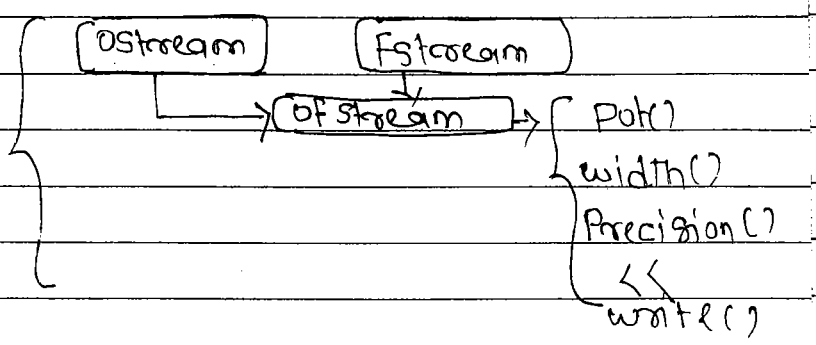
Istream class :-
get() :- member function of Istream class.
Fetches a single character including a blank space, tab and new line character, getc() can be used in two ways

```
cin.get(ch); // get a character from keyboard and assigns it to ch
    ch= cin.getc(); also can be used
```

getline() :- Member Function of istream class. we can const
the text more efficiently using line oriented
getline().
cin.getline(line, size)
Read character input into the variable line.
The reading is terminated as soon as either
'\n' is encountered or size characters are read.
char name[10];
cin.getline(name, 10);

P117 //
#include<iostream.h>
void main()
{ char name[20]; → Not print space
cout<<("\n Input name");
cin.getline(name, 20); [Print along with space]
cout<<name<<endl;
}

Ofstream :- 1. output File stream Class
2. This class object is used to write data
inside File.
3. It will create new File or open existing
File in order to write.
This class open File in output mode.



constructors :-

ofstream() ---- default constructor which create ofstream class object without Filename

ofstream(char *fname) -- this create of stream class object with given Filename.

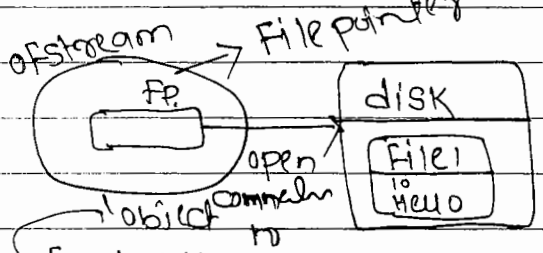
ofstream(char, fname, int mode)
This create ofstream class object with given File name and mode

default mode is ios::out
ios::____ .

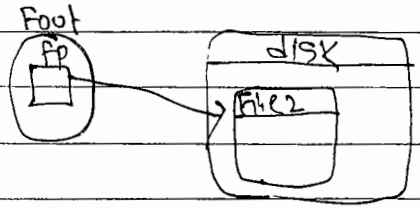
Member Function :-

open(char *fname, int mode)
This member function opens the File in given mode, default mode is ios::out.

ofstream Fout("File")



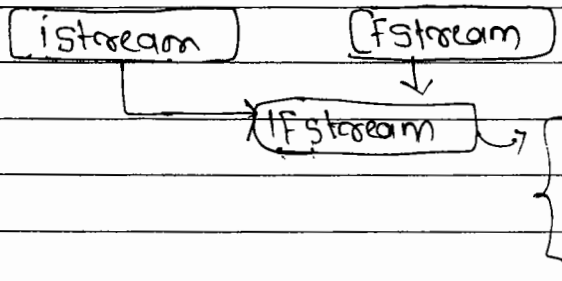
ofstream Fout;
Fout.open("File2");



P118: /* Program to create marks File and store student marks details */

```
#include <iostream.h>
#include <fstream.h>
void main()
{ char name[10];
  int sub1, sub2;
  char choice;
ofstream Fout("marks");
do
{ cout<<"\n input name";
   cin>> name;
cout<<"\n input two subjects";
 cin>> sub1>>sub2;
Fout<<name<<endl;
Fout<<sub1<<endl;
Fout<<sub2<<endl;
cout<<"\n Add another student?";
    cin>> choice;
}while(choice != 'n');
 Fout.close();
}
```

ifstream :- Input File Stream class.
This class is used to read data from existing File
it opens the File in input mode.



constructor :-
ifstream(char *fname) : This constructor create input File Stream class object with given Filename.

ifstream(): This constructor create input File Stream class object without File.

P119 // wap to list/read marks details from marks File and Find results.

```
#include<iostream.h>
#include<fstream.h>
Void main()
{ ifstream File("c:\\marks");
 char name[10];
  int sub1, sub2;
while(!Fin.eof()) // or while(1)
 {  Fin>>name;
    Fin>>sub1;
    Fin>>sub2;
                  -> //or if(Fin.eof());
                        break;
cout<<endl<<name<<"\t"<<sub1<<"\t"<<sub2;
if(sub1<40 || sub2<40)
 cout<<"\tfail";
else
cout<<"\t Pass";
}  Fin.close();
}
```

Working with text Files

P120 // wap to create text File

```
#include<iostream.h>
#include<fstream.h>
void main()
{    char ch;
```

```
ofstream Fout;
Fout.open("c:\\text.txt");
while (ch=cin.get()!='*')
 Fout.put(ch);
 Fout.close();
}
```

P.121 Wap to read from text file.

```
#include<fstream.h>
Void main()
{ char ch;
ifstream Fin("c:\\text1.text");
while(1)
{  ch= Fin.get();
  if( Fin.eof())
 break;
cout<<ch;
}
 fin.close();
}
```

↳ writing object inside File P.1

P.122

```
#include<iostream.h>
#include<fstream.h>
class address
{
 char name[10];
 char city[10];
Public:
 Void read_address()
{  cout<<"\n input name";
    cin>> name;
```

```
    cout<<"\n input city";
    cin>>city;
  }
  void Print_address()
{ cout<<"\n Name :"<<name;
  cout<<"\n city:" << city;
} };
  void main()
{ ofstream fout("c:\\address_book");
  fout("c:\\address_book");
char ch;
do {
add.read_address();
Fout.write((char*)&add, sizeof(add));
cout<<"\n Add another address";
cin>>ch;
} while(ch!='n');
Fout.close();
}
```

Reading objects Foram Files

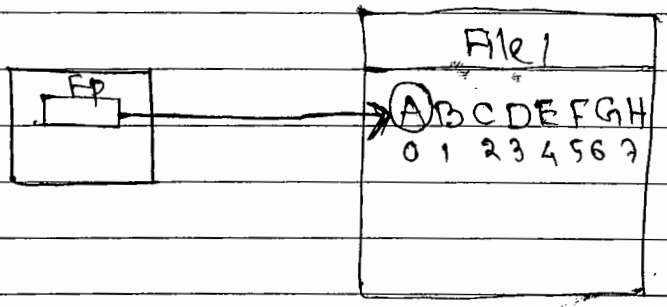
P.123

```
#include("address.cpp")
#include <iostream.h>
#include <fstream.h>
void main()
{
address a;          c:\\
 ifstream Fin("address_book");
while(1)
{ Fin.read((char*)&a, sizeof(a));
 if(Fin.eof())
 break;
  a.print_address(); } }
```

Manipulation of File pointer

1. we can move the File pointer to any desired Position in the File. we can take the control of the File positions.
2. The Functions to the position the File pointer at desired position.
3. seekg(offset, position). seekp(offset, position); offset parameter represents no. of bytes and position parameter takes one of the following constants

ios::beg, ios::cur, ios::end



Fin.get(); → A
Fin.seekg(5, ios::beg)
Fin.seekg().get(); F
Fin.seekg(2, ios::cur);
Fin.get(); H

Fin.seekg(-4, ios::end);

Fin.get(); → D

P.184

```
#include<iostream.h>
#include<fstream.h>
#include"address.cpp"
void main()
```

```
{ int recno;
  address a;
ifstream
Fin("c:\\address_book");
cout<<"\n Input record number";
cin>>recno;
Fin.seekg(recno*sizeof(a),ios::beg);
Fin.read((char*)&a,sizeof(a));
if(Fin.eof())
cout<<"\n Invalid recno";
else
a.print_address();
}
```

Exceptions :-

1. Exception is an error occurs during runtime.
   or
   exception is a runtime error.
2. Exception is logical error.

1. Exceptions are unusual conditions in a program.
2. They may cause the program to fail or may lead to errors. They must be dealth with.
3. There are two types of exceptions synchronous and asynchronous 'out of range index' and 'over-flow' are synchronous type.
4. Errors caused by the events beyond the control of the program like keyboard interrupts are called asychronous exceptions.

TC-30- we cant write try and catch.
we should write only in TC 4.5 or high version

try block :- 1. This block contain code which is excepted to throw exception.
2. This block contain condition.
3. This block throws exceptions using throw keyword.

```
try
{ code which has to be monitor for
   exception handling;
}
```

catch block :- 1. Catch is an exception handler.
2.

```
catch(exceptiontype var)
 {
    Statements;
 }
```

Pgm
```
#include<iostream.h>
Void main()
{ int n1, n2;
cout<<"\n Input any two numbers";
 cin>>n1>>n2;
try
{  if(n2==0)
       throw 0;
   else
     {
       int n3 = n1/n2;
         cout<<n3;
     }
  }        ↳ typeof exception
 catch( int e)
 {
```

version

S

```
    cout<<"\n cannot divide number with zero";
  }
   }
```

now

P126.

```
#include<iostream.h>
void main()#include<string.h>
{
class login_exception();
void main()
{ char uname[10] password[10];
cout<<"\n input user name";
cin>>uname;
cout<<"\n input password";
cin>>password;
try
{ IF(Strcmp(uname,"nit")==0 && Strcmp(Password,
  Cout<<"welcome to c++");
  else
   throw login;
} catch(login_exception &L)
  { cout<<"\n invalid username or password";
  }
}
```

The end.