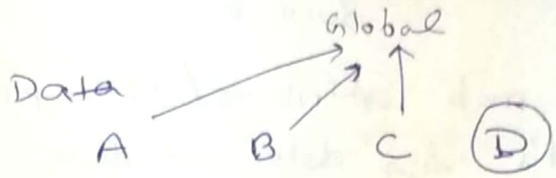
18/7

- Machine Language programming :- 01010001
- Assembly language programming (ALP) :- ALP was better than MLP in a sence that assembly LP were more like English. Ex :- ADD A, B
  Move B, C
  Jump 9000
- Structured/Procedural programming :-

C code :



Cons :-
1) Long course
2) No data Protect
3)

- objects :- An object is an entity with data & code
  Typically, objects encapsulates some or all of the data & fonctionality.
  object → attributes/characteristics.
  → functionality/behaviors.

Ex :- Design & build a Compute hockey game.

Sol :- object : players, Ground,
Characteristic/Attributes :- Position, skills, speed, weight, height, no. of goals.
Functionality/Responsiblity :- Pass the puck, shoot
skate forward, skate backward

Popular object oriented program :-

1. Java
2. Python
3. c++
4. Visual basic, .Net

Ex :- Computational modeling on Biology :-
write a program that simulates the growth of virus population in humans over time. Each virus cell reproduce itself at some time interval. Patients under some drug treatment to inhibit the reproduction process & clear the virus from the body. However some of the cells are resistent to drug or may survive.

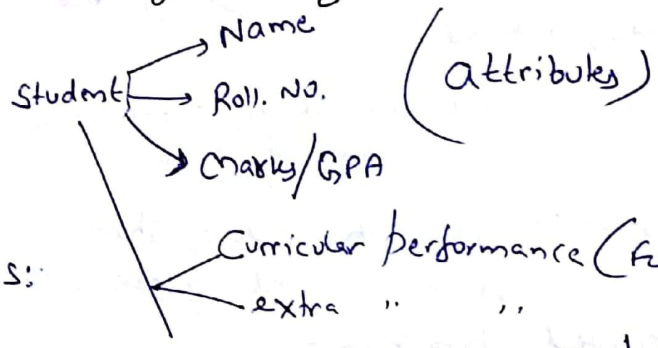
Characteristics
infected with virus / population.

Sol":- objects
Patients
Immunity to virus (%)
funtionalities / responsibilities
Take drug
Virus
object
reproduce itself
resistance (%)
funtionalities
Survive

In programming we map attributes / characteristic to nothing but variable or let's say data members. and we map functionality with functions.

2. Class :- A class is a construct that is used to describle code that is common to all objects of that class. It is specification of the object.

Student management System :-



Student S:

That means all the attributy and behaviour are related inside or that class. Object is an instance of type Student Class.

3. Abstraction :- Abstraction has two view points. The first one to take the necessary details & ignoring the rest. The second viewpoint is to create objects & class & use them without even knowing what's code. That is an another aspect of Abstraction.

23/7/21

4. Encapsulation :- { class Student

data & functionality bound together in class

data private: { Name of the Student, Roll No., marks

public { Curricular ( ), Extra ( ) } → Boundary Class

Controlling the access over data → Interface

5. Inheritance :- Class Health

Object (Father)

object (child) → properties (data)

Class vechile {

Private/protected :- String license/No.plate, int year

Public: vechile (Const string & my licence, Const. int myyear)

{ return licence "from" + string (year) }.

{ Retun maket " " + string)

}

Ex:- Class Car {

public : Vehicle :

String style;

Public: Car (Const. String & mylicence, Const int year, Const. string & style)

: vehicle (my licence, my year), style (mystyle) { }

Const.

6. Polymorphism :- A greek term "many ..." polymorphism is used to facilitate the use of Software. Via polymorphism same function can be applied to different objects to give different results

25/07/18 Software (S/w) are large & complex and they are developed through four phases :-

1. Planning
2. Analysis
3. Design
4. Implementation

The system development life cycle (SDLC) is the process of understanding how an information system (IS) support needs, build up & delivered to the users. The terms used for them are :-

1. Problem Space distribution
2. Programming Language abstraction
3. Algorithm Abstraction
4. Procedural Language/OOP Approach
5. Data Abstraction
6. Software Abstraction

The three fundamental principles by which Software can be developed are :-

1. Planning / design
2. Abstraction i.e. Concept of hiding the information with the recognition of Structures
3. Encapsulation / Data privacy.
4. Programming paradigms :-
   a) machine learning 10110101
   b) Assembly language
   c) Structured / Procedured lang.

d) Object oriented lang. (OOP)-

In OOP, all the Objects are independent, they are Connected to one another only through the messages. It's not possible in our usual procedural/structured programming.

```
class Student {
    roll No.
    name
    class

    Curr. Per. ( )
    Extra Per. ( )
}
```

Limitations of structured/procedural lang-

1. Does not effectively addresses data abstraction & information hiding.
2. Does not response to changes in the program or problem domain.
3. All data used by them end up being global to entire Software. This means any change in their representation tends to affect all other functional programming.

* Software quality measurement criteria:-

1. Correctness:- Program should meet their specifications.
2. Resilence & Reliability (Robust):- It should not fail quite often.
3. Maintainability:- Program should be easy to maintain & extended as required.
4. Luterportability:- Program should be readily Compatible with other systems/platforms. In fact they should be open system. (By open system we mean that we have built or others have built and whatever way we use it, we should expect it to work in every system with minimum effort required).
5. Reusability and generality:-
6. Efficiency:-
7. Verifiability:
8. Security:- Data knowledge & even functions may require selective & effective Concealment.

9. Integrity :- System & program needs protection against inconsistent update or malware.

10. Friendliness :- Program should be easy to use by a majority of users.

11. Describability :- It should be easily documented

12. Understandability :- It should be easy to understand or follow without any technical details.

★ Events which influenced the OOP development :-

1. Data privacy
2. Advances in computer architecture, compilers & operating system
3. Advances in programming languages, such as SIMULA, ADA, C++, JAVA, Objective C and so on.
4. Advances in programming method, ie. from machine Lang → Ass. Lang → struct. Lang → OOP.

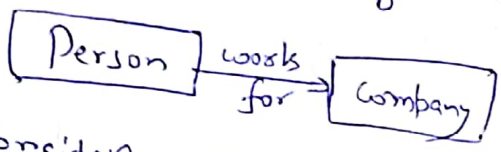
→ Benefits of OOP Approach :-

1. Reduces the total (life cycle) software costs by increasing the programmer's productivity & minimizing the maintenance cost.
2. Software resist both accidental & malicious attempts to their corruption or failure.
3. Modifications and extensions are performed in a much easier way as the object are self entity.
4. Enhances understandability & maintainability as objects and related operations are localised.
5. Provides a mechanism for formalizing the model of reality i.e., it make direct and natural correspondence between real world & its model.
6. The data centric design approach & process to secure privacy.
7. Abstracting the necessary information & observing the problem scalability.

6/2/18

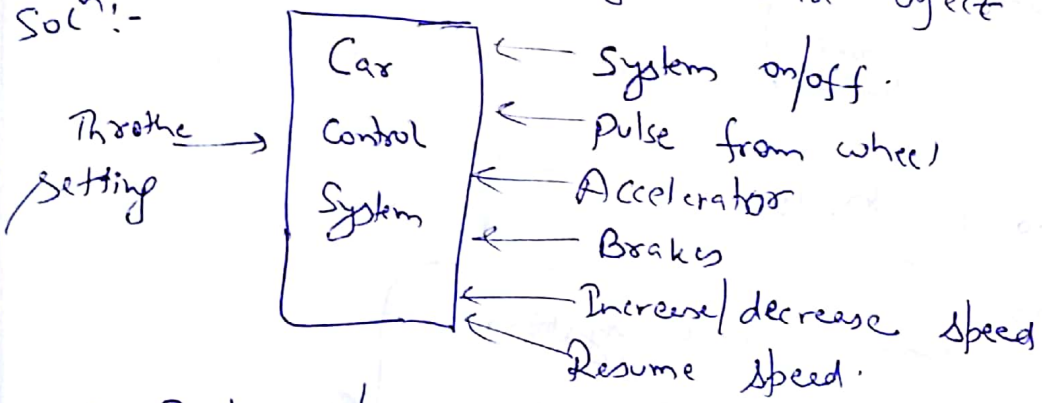
→ Object oriented modelling design notation & Concepts :-

→ Classes & objects :- They are linked together (one to one, many to one, many to many). Bullets are utilised to show the multiplicity. Association :- An association specifies a group of links between instances of the class. It is inherently bidirectional.



Ques :- Consider a Car Control System/model which is used to maintain the speed of a car. Draw the flow chart/block digram and Object model.

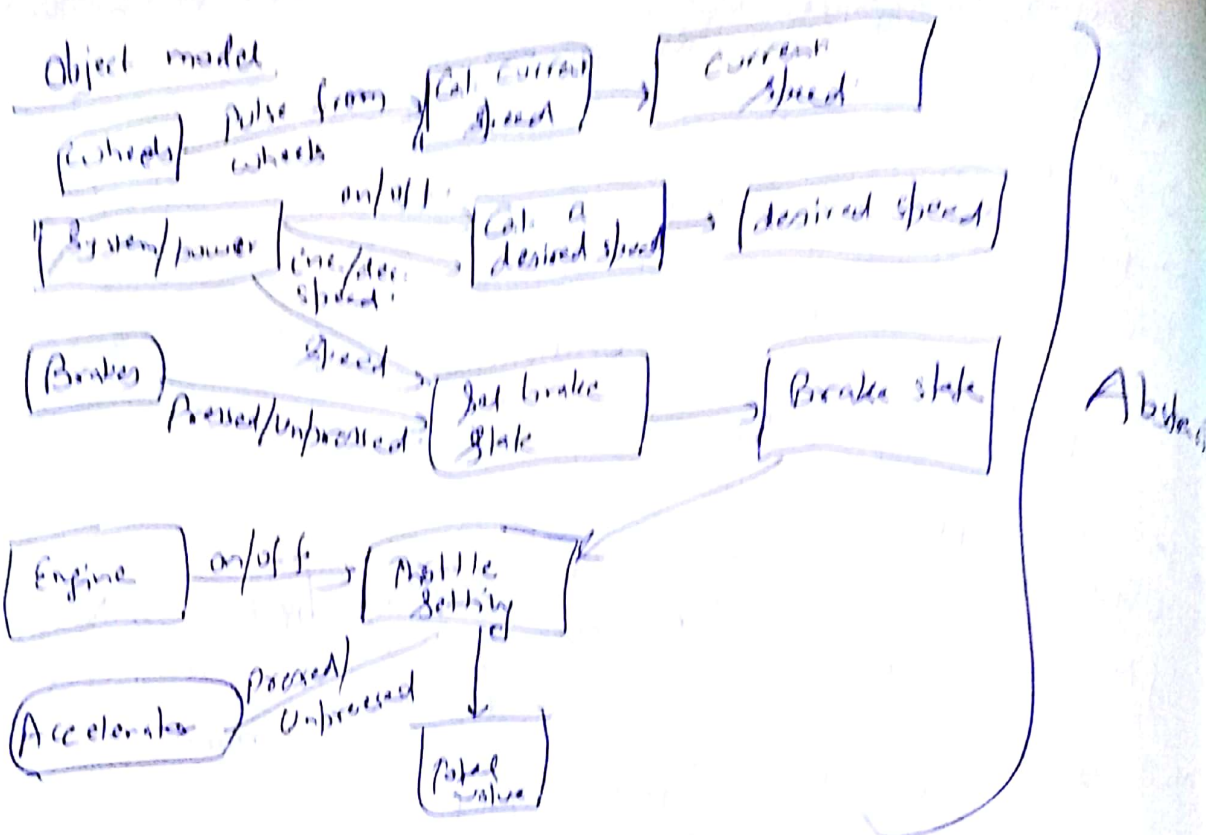
Soln :-



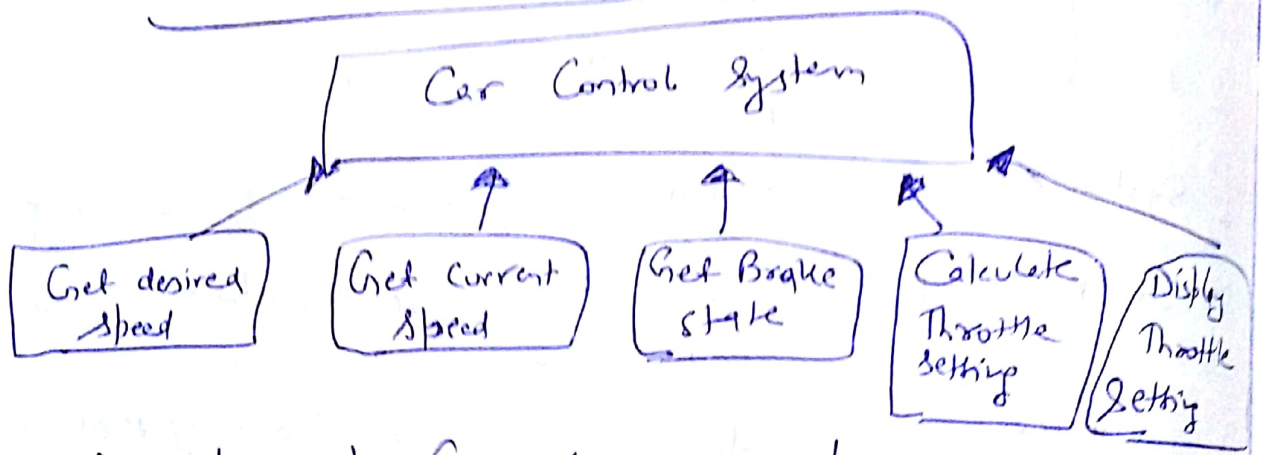
1. System on/off :- The program defats if the system is on or not.
2. Engine on/off :- The system will only be activated if the engine is on.
3. Pulse from the wheel :- A pulse is sent for every revolution of the wheel.
4. Accelerator :- It indicates how far/how hard the accelerator is pressed.
5. Brakes :- If pressed then your program for control system should be able to detect & maintain the speed accordingly.
6. Increase/decrease the speed :- Increase or decrease the speed only when the system is on.

7. Resume Speed :- Resume the last maintained speed

Object model

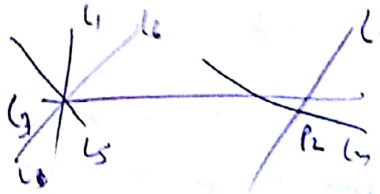


Functional model

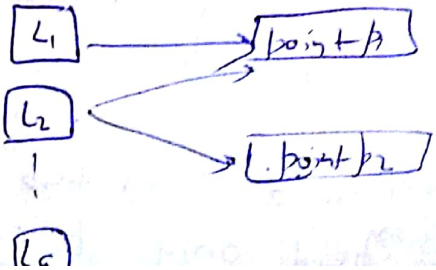


Ex:- (many to many) :- Given the intersect point P_1 & P_2. Find all the lines passing through those

Soln:-



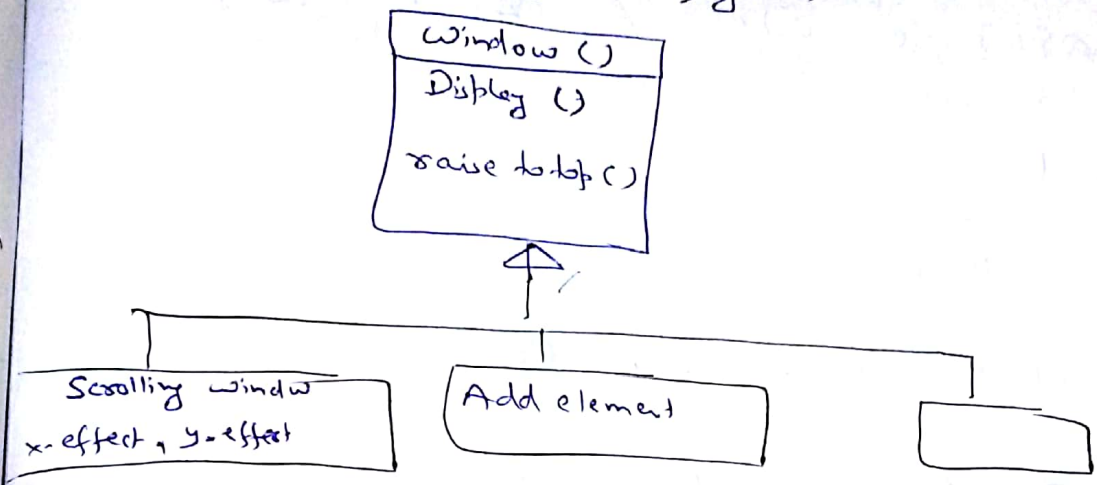
Object model



13/08/18

Aggregation :- point of relationship, transitive & anti-symmetric

Ex :- windows management System



E. 1. Type declaration statements :- Ex :- int x, float y, z, string;

2. Input/output statements :- Scanf(); printf(); getchar(), putchar(); gets(), puts().

Scanf ("<format string>", list of variables with & preceeding them),

1. Formatted
2. Unformatted :- Data means on Uniform stream of bytes. meaning these datas are interpreted by the programs which access them. This could be a binary data, text data, Command data, or even a program.

Ex :- read()/write(), scanf(), printf(), getchar(), put

Ex:- Hello → array of characters

Char a[6];

a[0] = 'H', a[1] = 'e', a[2] = 'l', a[4] = 'o', a[5] = '\0';

Ex:- Char c, d;

getchar (c);

putchar (d);

Scanf ("<formated string>", list of variables);

Printf ("formatted string", list of variable);

gets ( ) = when a white space character appears on the input line the string is terminated.

include <header.h> → means user defined files

include <iostream> → it has all operation with input output operations for object → oriented programming.

Cin → to read from keyboard

Cout → to display on monitor

Cerr → Errors to be displayed on monitor

Clog → for logging onto hardware.

C.S

Cin >> x

Cout << "Hello" → 'Hello' is considered as a string object & it will display on your monitor

Extraction operation

overloading.

§ Manipulation :-

(i) endl ; indicates the end of a line

E.g. :- Cout << x << endl ;

Cout << x << y << endl

(ii) setw() :→ It is to set width ;

Eg. int x ;

n is a the integer

Cout << setw( ) << endl ;

(iii) left / right

Eg. Count << left (right) << setw(10) << x << endl,

x will display '23 .... 70 ..

" " = 0000 12 .. 7

20/8/18.

§ **Arithematical & Logical Statements :-** Constants, variables, array names & function reference can be joined by some operators to form an expression.

Operators :- *, /, %, +, - ...

Logical operators :- != , < , > , <= , >= , &&, ||, ==

Eg :- 5.0/2 = 2.5

§ Type Casting / Co-ercion of data :- Data (type) in an expression.

Eg : (int (i+f))/4 , here i & f are integers.

§ Control instruction :-

(1) Decision Control Statement : if else Statement

if (expression) ——→ logical statement.

Statement 1, if (cdtn)

else S1

Statement 2. else S2

Nested if-else statements :-

```
if (C1
    if (C2 S2 else S3)

else
    if (C3 S4 else S5)
```

Switch statement :- switch (integer expression)

```
{ case 1 : do this
  case 2 : do this
  :
  :
}
```

default, do this :-

```
{
}
```

ii) Loop statements :- (i) while (expression) do → conditional :

```
{ - -

}
```

ii) for(exp1, exp 2, exp 3)
- exp1 → initial.
- exp 2 → Cdt^n value
- exp 3 → incr.

iii) do {

} while (cdt^n expression)

iii) The Comma Statement :-

for(j=0, alpha=100; j<50, jjj; alpha++).

Comma statement.

iv) Continue statement :- The control goes back to the first statements of the "for" loop, after getting the statement "continue".

v) Break :- This Statement ~~should~~ break the loop.

§ Functions :- data type
function name (list of arguments)
{ - -
return ( ); → This will return the value of the function.
}

Eg :-
```
# include <iostream>
main()
{ int a, b, c, d ;
  Cin >> a >> b >> c ;
    d = max (a, b) ;
  Cout << " maximum = " << max (c, d) << endl ;
}
```

Eg :- write a program to find the factorial of a given number.

```
# include <iostream>
  main ()
  { int a, f ;
    Cin >> a ;
    f = fact (a)
  Cout << "factorial :" << f << endl ;
  }
  { int fact (x)
      int x ;
  int  p = 1, i, j
    for (i=1, i <= x, i++)
        p = p * i ,  return (p) ; } .
```

Arrays :- Arrays is declared as a Storage Class Storage Class data type.

Array name (exp) = {val1, val2 -- valn}.
↑ indices

Eg :- int a[10] = {1, 2, -- 10}.

a[0] = 1, a[1] = 2, -- a[9] = 10

Eg.. Char a[3] = "RED"

a[0] = R, a[1] = E, a[2] = D, a[3] = '\n'

Eg :- float average();

{ avg. = average (5, list)
Size of array
↑ name of array

float list[100];

float average (int, float[])

average (a, x);

int a

float x[];

}

↑ function declaration.

23/8

Operators

```
import java.io.*
Public class test
{ Public static void main(String args[])
  int Count = 10
System.out.println("Count is" + Count)
  Count = Count >>>1;
  System.out.println("Count after is" + Count);
}.
```

Java API : Java Application programming interface

Static :- Implies the class can be accessed without creating the object of the class

Void :- implies return nothing.

Print Statement :- print(); println(); printf() are used to print any statement or a value.

Multidimensional Arrays:-

Strange class data type array name [int exp]
= {val1, val2 ... valn}

Eg:- int val[3][4] = {1,2, ...,12}

Val[0][0] = 1.

Arrays can be passed into a f

```
main()
{ int n, float list[100]
  float average (int : float[]);
  ay = average (n list);
}
float average (s, x)
{ int s
  float x[];
}
return;
```

Pointers

They are address and variable that hold them are known as pointer variable data type * Point variable

Eg:- int *a
float *x, *y;

int count &
int *ptvar
ptvar = &count;

Properties of pointer variables :-

i) A pointer can only be initialed to 0, which is equal value to NULL. defined in stdio.

ii) There are used for creating dynamics data structure.

iii) pointer can also be used to pass variable to function

iv) Pointers are closely related to arrays, hence array manipulations are simplified if pointers are utilized

Operations on pointers:

1. One pointer can be assigned to another pointer if they refer to the same data type.

2. They can be combined of equality (==), relational operators. But pointer values converted / to the changed value. Likewise you can perform (++) or (--) or add integer to pointer.

Eg: p = 2000

p+4 means p = 2000 + 4*4 = 2016

Eg:- program to convert inches into cm, by using pointe

```
#include<iostream.h>
void main()
{ void intocent() ——→ This is the function declarati
                      without any argument.
  double val = 10.0
  intocent(&val);}
```

Different type / declaration of pointer

Eg. 1. int (*p)[10];

⇒ p is a pointer of the integer type if it is 2-dimensional.

(2) int (*p)(char *a).

→ p is a pointer to a function which is taking pointer to character array "a" & returning integer value.

(3) int (*p(char *a))[10];

→ p is a pointer to a function taking character 'a' as an argument & returning an integer array of 10 element.

4) int p(char (*a)[10]);

→ p is a function taking two-dimensional array "a" of characters & returing an integer.

5) int p(char *a[]);

→ p is a function

6) int *(*p)(char (*a[3]);

→ p is a pointer to the function which is taking a two-dimensional character array "a" as an argument & returing pointer to integer.

7) int *(*p)(char *a[]);

→ p is a pointer to the function which is taking character array of pointer a as argument & returning pointer to integer.

8) int (*p[10])(void);

→ p is an array of 10 pointers to functions which is taking no arguments & returning an integer.

9) int *(*p[10])(char a);

→ p is an array of 10 pointers to the function which is taking character "a" as an argument and returning pointer to integer.

10) int *(*p[10])(char *);

→ p is an array of 10 pointers to the function which is taking pointer of character array as an ~~argument~~ argument & returning pointer to integer value.

§ Objects & classes in C++ & java/oop

1. Access specifier : public, private or protected.

functions, member function — attributes, data variables — inheritance property.

Eg:-

```
C++
Class small
{   private : int x;
    public : void set data (int d).
{ x = d
}
void display ( )
{ cout << "\n data is" << x << endl;
}
}.
```

Java

```
public class smath
{ private int x;
public void setdata(int d)
  { x=d;
  }
void display()
{ cout<<"\n data is "<<x;
```

Structures in OOP: Structure represent a no. of elements of different data type. Its general form is:

```
Struct tag {
        number 1;
        number 2;
        .
        .
        number n;
        }
```

→ tag is optional which specify the name of the structures.

i) No storage class is assigned to a structure

ii) members of a structure can not be initialised.

iii) A Composition of a structural variable is defined as

Struct tag Var1, Var2, ..., Varn.

Eg:-

```
Struct part {
        int model no.;
        int part no.;
        float cost;
      };
```

Note:- A structure can be a part of another structure.

Eg:- mantly acc* balance.

```
Struct date{
        int months;
        int day;
        int year;
    }
struct account
        {char acc_type;
    int acc_no;
    char name holder[80]
      float balance
      Struct date → last payment
    };
```

→ Structure & pointers:- If we have a pointer variable ptvar which points to a structure of data type variables then,

```
type *ptver;
ptver = &variable;
```

→ operator is used to access the ~~pointer~~ member functions of structure.

Eg:-

```
typedef struc {int acc_no;
               char acc_type;
               char name[80];
               float balance;
              }
account *pc;
pc = &acc_no;
```

Ex: typedef date { int month;
int day;
int year;
} date;

typedef account { int *acc_no;
char *acc_type;
char *name;
float *balance;
} data;

payment *pc;
*pc = &account;

* If we want to access acc_no by the structure account, account.account_no.

(.) dot operator is used to assign* & access members

* passing structures into functions :-
by values of numbers;
by reference;
by pointers;

Eg: typedef struct {
int feet;
float inch;
} distance;
distance add_length(distance, distance);

If we are pass

3/9/18

Constructor :- A Constructor in a program has following Properties :-

(i) They do not return any value.

(ii) They can create Constant objects.

(iii) They have the same name as the class.

Destructor :- A descructor in a program posses following properties :-

(i) They do not return any value & do not take any Argument.

(ii) They have the same as the class name proceeded by "~" Operator. Eg :- ~small();

(iii) They cannot be overloaded

(iv) They occur by default.

(v) object created in order are deleted in reverse order.

Eg :-

```
Class Counter
{ private : unsigned int Count;
public: Counter()
        {Counter = 0;}
        Counter() = {};
        void int count;
        };
```

→ Small & objptr:

It is a pointer to the Object of type Small.

Small & sref;

implies that sref is the alias to the object of the type small.

S, ref = &S1.

S ref is used as aliases for S1.

Ques:- Create a time class to display time in Universal & Standard format. Standard time format is followed by AM or PM.

Sol:

```
# include <iostream>
# include <string>
class time
{ private
    int hour;
    int minutes;
    int seconds

public:
    Time();
void settime(int, int, int) → default
  void print universal(); parameter passing
  void print standard();
};

Time:: Time()
    {
     hour = 0
     hmin = 0
     sec = 0;}.
```

```
Time :: Time (int h, int m int s)
    { hour = h ;
      min = m ;
      second = s;
    };
Void time :: Set time (int h, int m , int s)
    {
      hour = (h >= 0 && h < 24) ? h : 0 ;
      minute = (m >= 0 && m < 60) ? m : 0 ;
      Second = (s >= 0 && s < 60) ? s : 0 ;
    };

Void time :: print Universal ()
    { cout << setfill ("0") << setw (2) << hour
      << ":" setw(2) << minute <<
         ":" Set w(2) << second << endl

    };
Void time :: print Stander ()
        { cout << ((hour == 0 || hour == 12) ? 12 : hour % 12)
      << ":" setfill ('0') << setw(2)
      << minute << ":" << setw(2) <<
      Second << hour(12) "Am" : "Pm"
      << endl;
};

int main ()
    { Time t ;
      t. print stander ();
      t. print Universal();
      t. set time (14,36,57);
      t. print Universal();
        ? . stand ();
```

1. 00 : 00 : 00 Am
2. 00 : 00 : 00
3. 14 : 36 : 57
4. 2 : 36 : 57 Pm.

→ Int time:: Set hour (h)
{ hour = h
} x not allowed ;

Utility functions/Utilities :- Sometimes few functions are defined in the private part then each function are called by utility function. They are called utility functions when they are used to access the public part of a program.

In another words, Utility function are the members functions of the private part which assist the member function of the public part of a class.

Eg:- write a c++ program to input sales figure of 12 months & print out the annual sales report.

Sol:- # include <iostream>
Class salesperson
{
Private:: double total annualsale ();
double sale [12];
Public: sales_person();
Void getsalesfromuser ();
Void setsales (int double);
Void print Annual sale ();
};

```
Sales person:: Sale person()
        { for(int i=0; i<12; i++)
          Sales[i] = 0.0;
        }
Void GetSaleFromUser :: SetSaleFromUser()
        {double salesfigure;
    for(i=1, i<=12; i++)
  Cout << " Enter the sales account for month" <<
       i<<"." << endl;
  Cin>> Sales[i];
        OR
  Cin>> salesfigure;
   Set values (i, salesfigure);
  }
Void Set Sales :: Set Sales (int month, double amount)
    { if(month>0 && month <=12 && amount >=0)
    Sales [month-1] = amount;
else
    Cout << " Invalid months or salesfigure";
    }.
Void Sales person :: Print Annual Sale()
  { Cout << set precision(2) << fixed <<
  "Total annual sales are" << Total amount sale() << endl;
```

[Set precision → It is used to set the no of digits we want after decimal.
fixed → shows "." should also be showed].

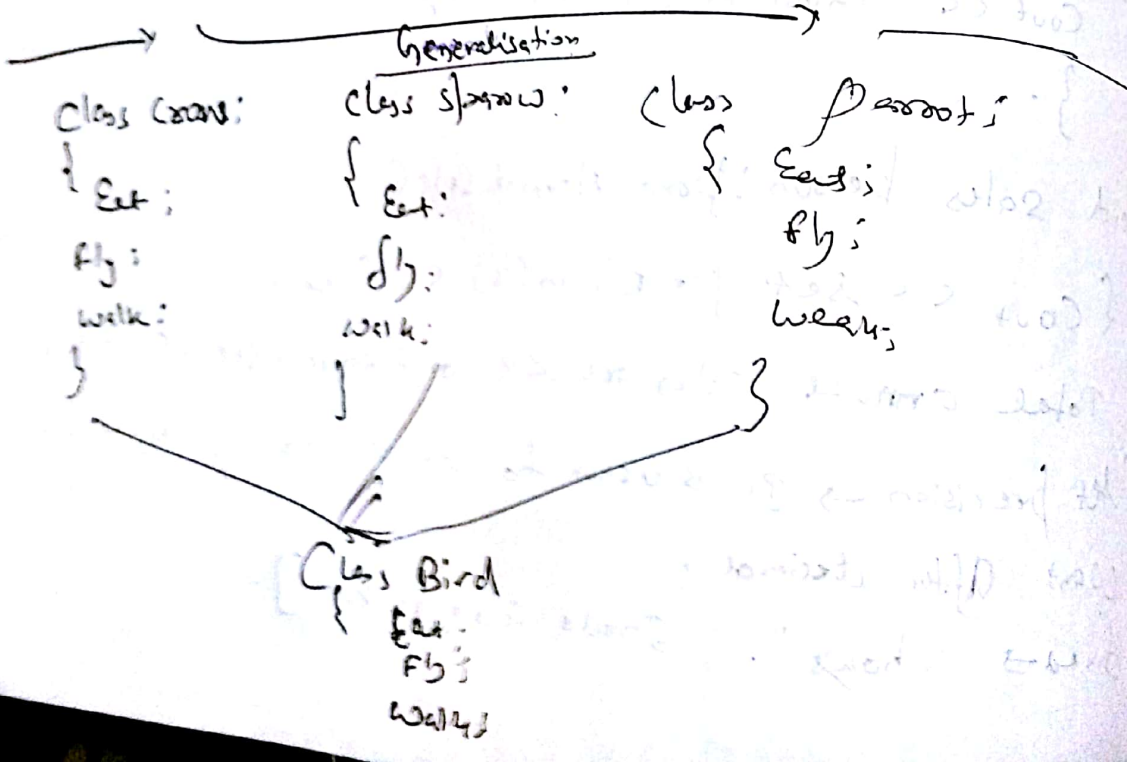
double sales function()
{ double total = 0.0
for (int i = 0; i<12; i++)
total += sales[i];
return total;
}

§ Generalization:- It is process of extracting shared characteristics from two or more classes and combining them into a generalized super class.

§ Aggregation:- It is defined as a group of assembly relationship (or functions) between classes (◇).

Eg

§ Inheritance:- many subclasses can be derived from superclass with the property that the derived class inherits the properties of the super class as well as they can define their own property or overwrite the existing properties of superclass (◇).

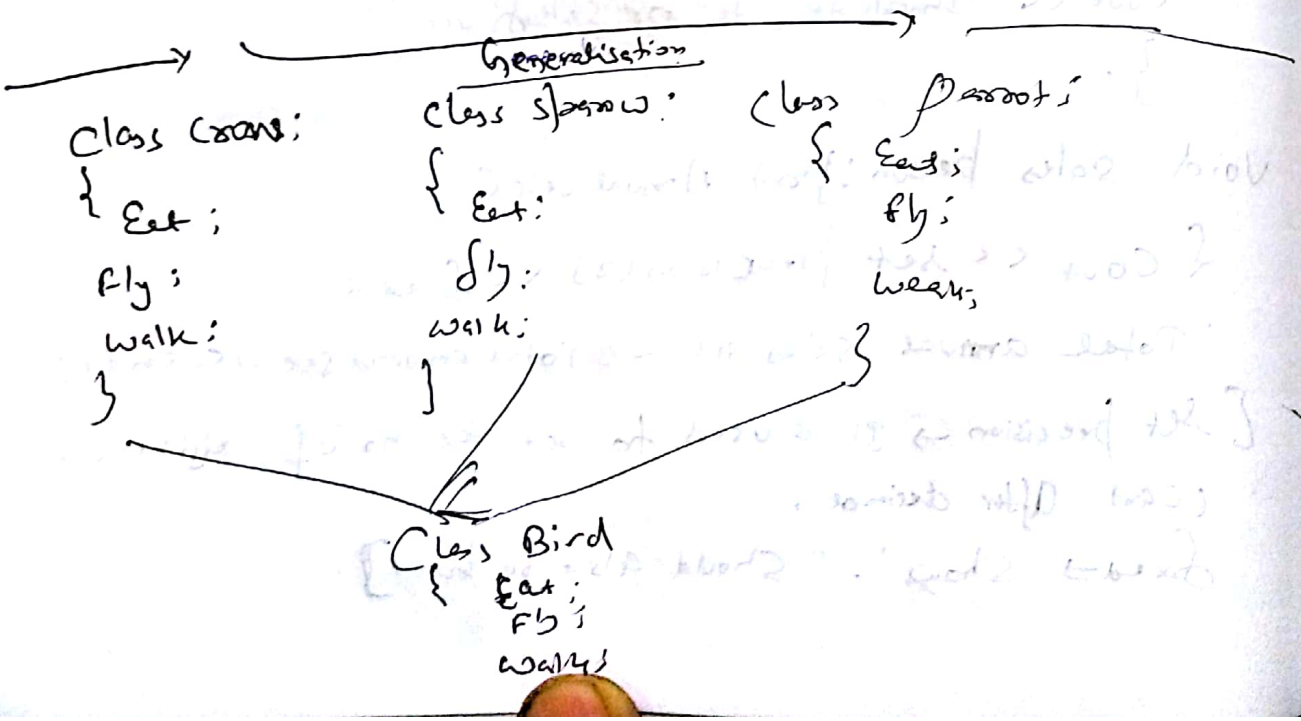


```
double SalesPerson:: TotalAnnualSale()
{ double total = 0.0;
for(int i = 0; i<12; i++)
  total += sales[i];
  return total;
}
```

§ Generalization:- It is process of extracting shared characteristics from two or more classes and combining them into a generalized super class.

§. Aggregation:- It is defined as a group of assembly & relationship (or functions) between classes (◇).

§ :- Inheritance:- many subclasses can be derived from a superclass with the property that the derived class inherits the properties of the super class as well as they can define their own property or overwrite the existing properties of superclass (○).



Inheritence

```
Class Dog
  { walk;
    Eat;
    Play;
    Bark;
  }
Class Germanshherd
  { walk;
    Eat;
    Play;
    Bark;
  }
```

Aggregation:-

```
Class dog
  { walk
    Play;
    Eat
    Bark;
  }
Class Cat:
  { has Dog
    Eat = Dog.Eat;
    Play = Dog.Play;
    walk = Dog.walk;
    purr.
  } end;
```

Aggregation

```
Class pet
  { Eat;
    walk;
    Play;
  }
Class Dog is pet
    Bark;
end;
Class cat is pet
    Purr.
  end;
```

Ques:- write a program for product of two matrice $[A]_{m \times n} \times [B]_{n \times p} = [C]_{m \times p}$.

Sol":-

```
#include<stdio.h>
main()
{ int i,m,n,p,q, a[10][10], b[10][10], c[10][10];
printf("Enter the order of A & B respectively");
scanf("%d %d %d %d", &m, &n, &q, &p)
printf("Ent matrix A is\n");
  for(i=0; i<m; i++)
    for(j=0; j<n; j++)
  Scanf("%d", &a[i][j]);
printf("matrix B is \n");
for(i=0; i<n; i++)
for(j=0; j<p; j++)
  Scanf("%d", &b[i][j]);
```

```
if (n == q)
  { for(i=0;i<m;i++)
    for (j=0 j<p;j++)
  { s=0
    for (k = 0 ;k<n;k++)
      s= s+a[i][k] * b[k][j];
      C[i][j] = s;
    }
  }
else
printf("matrix multiplication is not possible");
printf ("product of A & B is \n");
  for (i=0 ; i<m; i++)
    for(j=0 ; j <p ; j++)
  printf ("%d", c[i][j]);
  }
```

```
Struc date;
 { int month;
   int day;
   int year;
 }
```

```
Structure Acc
    { char acc_type;
      Char acc_no;
```

→ Self-referential structure:

C

25/8/18

Types of Linkages :-

i) External : Global, non-static variables & function.
ii) Internal : Static variables & functions with file scope.
iii) None : local variables.

→

1. Primary data type	2. Secondary or User-defined
1. Int	1. Array
2. Char	2. Pointer
3. float	3. Structure
4. double	4. union
5. void	5. Enum