10/8/2020 ①

Compiling process:- of C & C++

C/C++
⇓ (Source code of c/c++ lang goes to compiler)
compiler
⇓ (Compiler makes an object code (compiled))
object code
⇓ (object code goes to OS)
OS
⇓ (OS transfer it to Hardware)
Hardware
⇓ (Hardware transfers it to ALU)
ALU

Qu. What is platform Dependency & platform Independency.

foreg:- Web server → Web Application is placed at web server
Windows Unix Android etc

- Whenever a web application can be opened on any of the OS then it is known as platform Independency
- Cpp is extension of C++ file
- C & C++ lang. is platform dependent because object code made by compiler is specific for hardware & OS.

11/8/2020

Data types:-
- int
- float
- char

Q Why size of char is 1 byte in C?
Ans. • It works on ASCII
• It supports capital as well as small characters
• It can store 8 bits i.e, 1 byte

operator
① Unary operator
(i) ++ → It increases value of variable by 1
(ii) -- → It decreases value of variable by 1

for eg: int a = 2;
{
printf("%d %d %d %d", a++, ++a, ++a, a++);
getch();
}

a++ → post Increment: first process then increases.
++a → pre Increment: first Increases then process

Ans a++ , ++a , ++a , a++
23 ← 23 ← 22 ← 21 ← 20

O/P → 23, 23, 22, 21

(ii) Binary operators
: The operators which use 2 operands

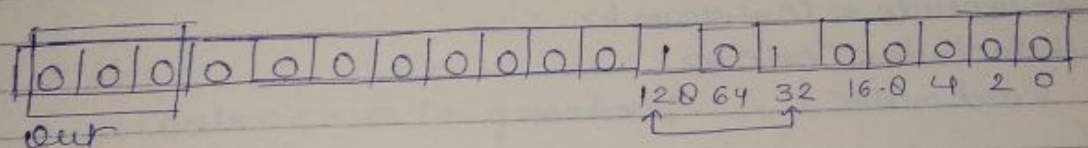
13/08/2020

Arithmetic Operators:-
⇒ +, −, *, /, % → Checks Remainder

Relational operator
<, <=, >, >=, !=

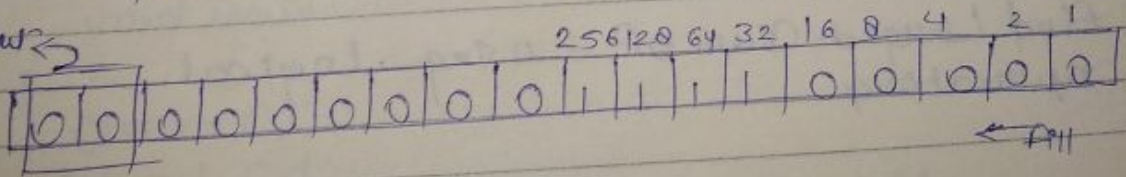
SHIFT OPERATOR:-
: It works on bits (010101 ...)
(1) Left Shift operator (<<)
for eg:-
→ int x = 20; [cout is printf in C++]
cout<<x(<<3); [cin is scanf in C++]

binary of 20: 1 0 1 0 0
16 8 4 2 1



128 + 32 = 160
(x<<3) = 160
for eg:- int x = 120;
cout<<(x<<2);

64 32 16 8 4 2 1
binary of 120:- x = 1 1 1 1 0 0 0



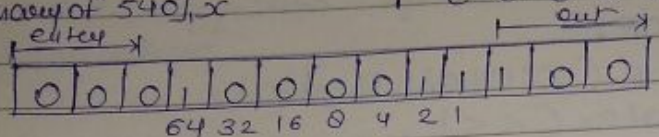
(x<<2) = 256 + 128 + 64 + 32
(x<<2) = 480

13/8/2020

(ii) Right Shift Operator (>>)

for eg: int x = 540;
cout << (x >> 3);

	512	256	128	64	32	16	8	4	2	1
[binary of 540], x	1	0	0	0	0	1	1	1	0	0



(x >> 3) = 64 + 2 + 1 = 67
(x >> 3) = 67

Relational operator
<, >, <=, >=, !=

```
# include <iostream.h>
# include<conio.h>
  void main()
{
  clrscr();
  cout<<(10>20);
  getch();
}
```

Logical operator
&&, ||, !

Q. find Largest Num of 3 using logical operators.

14/08/2020

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int a,b,c;
// Input
 cout<<" Enter the value of a,b,c:";
 cin>>a>>b>>c;
 if(a>b && a>c){
   cout<<" A is greater";
 }
 else if(b>c){
    cout<<" B is greater";
 }
 else{
    cout<<" c is greater";
 }
  getch();
}
```

Conditional operator (?,:)

```
#include <iostream.h>
#include <conio.h>
  void main()
{
 clrscr();
 int a, b, c, d;
 cout<<" Enter values of a, b, c: ";
```

```
cin>> a>>b>> c;
d= (a>b && a>c)? a: (b>c) ? b: c;
cout<<"D= "<<d;
getch ();
}
```

Difference B/w Signed & Unsigned Operator.

SIGNED	UNSIGNED
• We can store both +ve & -ve values	• We can store the +ve values only.

Types	Byte	Range
1. Char	1	-128 to 127
2. Unsigned char	1	0 to 255
3. Signed char	1	-128 to 128
4. int	2	-32768 to 32767
5. Unsigned int	2	0 to 65535
6. signed int	2	-32768 to 32767
7. short int	2	-32768 to 32767
8. Unsigned short int	2	0 to 65535
9. Signed short int	2	-32768 to 32767
10. Long int	4	-2147483648 to 47
11. Signed long int	4	-2147483648 to 47
12 Unsigned long int	4	0 to 4294967295
13 float	4	3.4E-38 to 3.4E+38
14. Double	8	1.7E-308 to 1.7E+308
15. Long Double	10	3.4E-4932 to 1.1E+4932

17/08/2020

operators in C++

① (::) Scope Resolution operators

: It is used to uncover / get hidden names due to variable & allow access to global variable

```
#include<iostream.h>
#include<conio.h>
 int a = 3000;
 void main()
{
 int a=2000, b;
{
  b=a;
int a = 1000;
cout<<"B = "<<b <<endl;
cout<<"A = "<<a<<endl;
}
 cout<<"A = "<<a<<endl;
 cout<<" ::A = "<<a<< endl;
  getch();
}
```

O/P→

```
B =  2000
A =  1000
::A= 3000
  A= 2000
::A= 2000
```

LOOPS:-

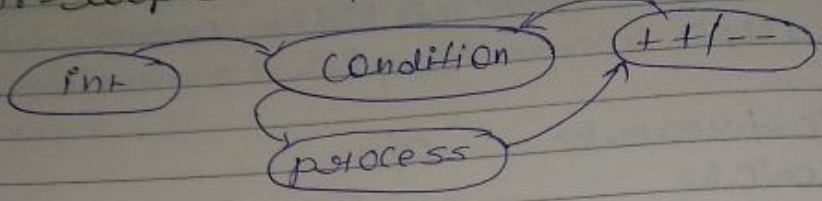
1) for-loop

syntax: for(intialization; condition; -/++)

{ process

}

• for-loop life cycle:-



• for-loop program

```
# include <iostream.h>
# include <conio.h>
  void main()
{
  int i;
for(i=1; i<=10; i++){
 cout<< i<< endl;
}
  getch();
}
```

O/P →
1
2
3
4
5
6
7
8
9
10

Q. To print table of 2.

```
# include <iostream.h>
# include <conio.h>
  void main()
{
  int i;
for(i=1; i<=10; i++){
  cout<<(i*2)<<"\n";
}
  getch();
}
```

O/P →
2
4
6
8
10
12
14
16
18
20

18/08/2020

Q. To print table of any no.

```
#include<iostream.h>
#include<conio.h>
  void main()
{
 int i,n;
 cout<<"N=";
 cin>>n;
 for(i=1; i<=10; i++)
{
  cout<<(i*n)<<"\n";
}
 getch();
}
```

O/P → N=6

6
12
18
24
30
36
42
48
54
60

Q. To print table of any no. (3*1=3) → Like this

```
  void main()
{
 int i,n;
 cout<<"N=";
 cin>>n;
for(i=1; i<=10; i++)
{
 cout<<n<<"*"<<i<<"="<<n*i<<endl;
}
 getch();
}
```

O/P → 4*1=4
4*2=8
4*3=12
4*4=16

4*5 = 20
4*6 = 24
4*7 = 28
4*8 = 32
4*9 = 36
4*10 = 40

Q. To print n=5!, 5! = 1x2x3x4x5 using for loop.

```
# include <iostream.h>
# include <conio.h>
 void main()
{
int i, f=1, n;
cout<<"N=";
cin>>n;
for(i=n; i<=n; i++)
{
 f= f*i;
}
cout<<"I fact="<<f;
getch();
}
```

O/P → n=5
fact=120

Q. To print 5! = 5x4x3x2x1 order

```
 void main()
{
int n, i, f=1;
cout<<"N=";
cin>>n;
```

O/P → n=5
fact = 120

19/8/2020

```
for(i=n; i<0; i--)
{
 f= f*i;
}
 cout<<"fact="<<f;
 getch();
}
```

• find Individual process

n= 7868

① 7868 % 10 = 3
0+3 = 3
7868/10 = 786

② 786 % 10 = 6
3+6 = 9
786/10 = 78

③ 78 % 10 = 8
9+8 = 17
78/10 = 7

④ 7 % 10 = 7
17+7 = 24
7/10 = 0

Now, n = 0 & Sum = 24

2) While-Loop.

Q. find the Sum of a given no. using While loop

```
void main()
{
int n, p, s=0;
cout<<"N=";
cin>>n;
while(n!=0)
```

o/p →
n = 78
p = 78 % 10 = 8
s = 0+8 = 8
n = 78/10 = 7

```
{
 p = n % 10;
 s = s+p;
 n = n/10;
}
cout<<"sum="<<s;
getch();
}
```

Now, n=7
p = 7 % 10 = 7
s = 0+7 = 15
n = 7/10 = 0

Now, n=0 & sum=15

O/P → n=70, sum=15

Q. To print Reverse of a Number

```
void main()
{
 int n, p, s=0;
 cout<<"N=";
 cin>>n;
 while(n!=0)
 {
 p = n % 10;
 s = s*10+p;
 n = n/10;
 }
 cout<<"Reverse="<<s;
 getch();
}
```

O/P → n=54
p = 54 % 10 = 4
s = 0*10+4 = 4
n = 54/10 = 5

n = 5
p = 5 % 10 = 5
s = 4*10+5 = 45
n = 5/10 = 0

O/P → n=54
Reverse = 45

Q. Check whether a No is palindrome or Not ;-

```
void main()
{
int n,p, s=0 q;
cout<<" Enter any NO.\n";
cin>>n;
q=n;
while(n!=0)
{
p=n%10;
s= s*10+p;
n= n/10;
}
if(s=q){
cout<<" pallindrome";
}
else{
  cout<<"Not pallindrome";
}
getch();
}
```

O/P → Enter any NO.
707
pallindrome
(OR)
Enter any NO.
123
Not pallindrome

Q. Check whether a no is Armstrong no or Not.

```
void main()
{ int n,p, s=0, q;
cout<<" Enter any no.";
cin>>n;
q=n;
while(n!=0)
{ p=n%10;
s= s+(p*p*p);
```

O/P → Enter any NO.
153
NO. is Armstrong.

(OR)

Enter any NO.
141
NO. is not Armstrong.

20/8/2020

```
    n=n/10;
}
if(q==5){
cout<<"Number is Armstrong";
}
else{
cout<<"No is not Armstrong";
}
getch();
}
```

#Nested loop

```
void main()
{int i,j;
for(i=0;i<3;i++){
for(j=1;j<=5;j++){
cout<<"J="<<j;
}//j loop closed
cout<<"\n";
}//i loop closed
getch();
}
```

O/P →

i=0 j= 1 2 3 4 5
i=1 j= 1 2 3 4 5
i=2 j= 1 2 3 4 5

② print values of j in order to values of i.

```
void main()
{
int i,j;
for(i=0;i<10;i++)
{
```

O/P →

i=0, j= 1 2 3 4 5 6 7 8 9 10
i=1, j= 1 2 3 4 5 6 7 8 9 10
i=2, j= 1 2 3 4 5 6 7 8 9 10
i=3, j= 1 2 3 4 5 6 7 8 9 10
i=4, j= 1 2 3 4 5 6 7 8 9 10

20/08/2020

```
for(j=1;j<=10;j++)
{
cout<<"j = "<<j;
}
cout<<"\n";
}
getch();
}
```

i = 5 j = 1 2 3 4 5 6 7 8 9 10
i = 6 j = " " " " " "
i = 7 j = " " " " " "
i = 8 j = " " " " " "
i = 9 j = " " " " " "
~~i = 10~~ j = " " " " " "

Q. To print value of j in descending order by 1

```
void main()
{
int i,j;
for(i=0;i<10;i++){
for(j=1;j<=10-i;j++){
cout<<"J = "<<j;
}
cout<<"\n";
}
getch();
}
```

i=0, j = 1 2 3 4 5 6 7 8 9 10
i=1, j = 1 2 3 4 5 6 7 8 9
i=2, j = 1 2 3 4 5 6 7 8
i=3, j = 1 2 3 4 5 6 7
i=4, j = 1 2 3 4 5 6
i=5, j = 1 2 3 4 5
i=6, j = 1 2 3 4
i=7, j = 1 2 3
i=8, j = 1 2
i=9, j = 1

Q. To print values of j in ascending order by 1

O/P

```
void main()
{
int i,j;
for(i=0;i<10;i++){
for(j=1;j<=i+1;j++)
{
cout<<j;
```

i=0, j=1
i=1, j=1 2
i=2, j=1 2 3
i=3, j=1 2 3 4
i=4, j=1 2 3 4 5
i=5, j=1 2 3 4 5 6

```
}
cout<<"\n";
}
getch();
}
```

```
i=6, j= 1 2 3 4 5 6 7
i=7, j= 1 2 3 4 5 6 7 8
i=8, j= 1 2 3 4 5 6 7 8 9
i=9, j= 1 2 3 4 5 6 7 8 9 10
```

Q. To print * value of j in ascending order

```
void main()
{
int i,j;
for(i=0;i<10;i++){
 for(j=1;j<=i+1;j++){
 cout<<"*";
}
cout<<"\n";
}
getch();
}
```

```
i=0, j= *
i=1, j= **
i=2, j= ***
i=3, j= ****
i=4, j= *****
i=5, j= ******
i=6, j= *******
i=7, j= ********
i=8, j= *********
i=9, j= **********
```

Q. To print Alphabets

```
void main()
{
int i,j;
char t='A';
for(i=0;i<10;i++){
for(j=1;j<=i+1;j++)
{
 cout<<t;
  t++;
}
```

O/P →
A
BC
DEF
GHIJ
KLMNO
PQRSTU
VWXYZ.

20/08/20

```
{
 cout<<"\n";
}
 getch();
}
```

Q. To print blankspaces in descending order with *

```
 void main()
{ int i,j,k;
for(i=0; i<10;i++){
for(k=0; k<10-i; k++){
 cout<<" ";
}
for(j=1;j=i+1;j++)
{
 cout<<"*";
}
 cout<<"\n";
}
 getch();
}
```

O/P→

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
```

Qw: to print Alphabets:-

```
  void main()
{
 int i,j,k,n;
 char t='A';
for(i=0; i<10; i++){
 for(j=0; j<=i; j++){
  cout<<t;
  t++;
```

O/P→

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
ABCDEFG
ABCDEFGH
```

```
    t='A';
    cout<<"\n";
  }
  getch();
}
```

ABCDEFGHI
ABCDEFGHIJ

Q. to print a pyramid of stars with blank space

```
void main()
{
int i,j,k,n;
for(i=0;i<10;i++){
for(k=0;k<=10;k++)
{
 cout<<" ";
}
for(j=0;j<=i*i;j++){
cout<<"*";
}
 cout<<"\n";
}
getch();
}
```

O/P :-

```
          *
        ***
      *****
    *******
  *********
***********
 *********
  *******
   *****
    ***
     *
```

Q. to

Q. to print no. of lines of pyramid of stars:-

```
void main()
{
 int i,j,k,n;
 cout<<"N=";
 cin >>n;
 for (i=0; i<n; i++)
 {
 for(k=0; k<=n-i; k++){
  cout <<" ";
 }
 for(j=0; j<=i+i; j++
 cout<<"*";
 }
 cout<<"\n";
 }
 getch();
 }
```

O/P→

n=5

```
    *
   ***
  *****
 *******
*********
```

FUNCTIONS:-

- A function is a self-contained ~~block~~ program segment that carries out some specific, well-defined task. The use of programmer defined functions allows a large program to be broken down into a no. of smaller, self contained components, each of which has some unique, identifiable purpose.
- Simply, A self-contained block is known as function.
- When a code is divided into smaller modules then this process is known as structured

programming these modules are known as function

Category of functions:-
: Depending upon whether an argument is present or not and value is returned or not function can categorized as;-

* function with no arguments & no return value
: When a function has no arguments then, it does not receive any data from the calling function similarly when it does not return any value, the calling function does not receive any data from the called function. Such function can only be used as an independant statement

• Arguments but no return values
: The calling funct reads the data from the terminal and passes it on to the called function. The this case the actual arguments (a & b) and formal arguments (x & y) should match in number, type & order

→ The value of actual argument is assigned to the formal arguments one one-to-one basis, starting with first argument. Here,

When a function call is made, only a copy of the values of actual parameters is passed into the called functions. What occurs inside the function has not effect on variables used in the actual argument list. The variables declared inside a function are known as local variables & therefore their values are local to function & cannot be accessed by any other function. Here, function does not return any value to calling function.

- Arguments with return values:-

Very often it is required to passed the data into the called function, for function further processing, and processed data is sent back to calling function as a result. In such cases called function are required to have their return type. A type-specifier is present at the function header which is the datatype of the result in function

Eg:- #include<iostream.h>
#include<conio.h>
int sum(int x, int y)
{
return (x+y);
}
void main()
{
int a, b;
cout<<"Enter the value of a & b";
cin>>a>>b;

cout<<"sum="<<sum(a,b); // Calling sum funct

}

O/P → Enter the value of a & b: 10 20

sum = 30

- STRUCTURES:-
- A collection of dissimilar element is called structure

for ex, it can be used to represent a set of attributes such as student no, student name, address & marks

- The general syntax of a structure is given as follows;

struct tag-name

{

data type variable name 1;

data type variable name 2;

data type variable name 3;

};

- Tag name is optional. It is a name of the structure. Using the tag name we can declare structure variable anywhere in the program

22/8/2020

All variable which are declared inside the structure are called member of the element structure.

- Suppose a student consisting of student name, roll no. we define a structure to hold the information as follows:-

```
struct stud
{
  int r-no;
  char name[20];
};
struct stud s1, s2, s3;
```

- Here struct declares a structure to hold the details of two fields r-no (roll no.), and name. These fields are called member of structure. Each member may belong to a diff. type of data. stud is the name of the structure and is called structure tag.

- Declares s1, s2, s3 are variables of struct stud. Each one of these variables has two members and 22 bytes of memory

- There are no. of ways for assigning values to the members of structure. Since we know that members themselves are not variables, therefore they should be linked to the

Structure variables. But using the member operator :. The link b/w a member and a variable is established. This member operator is also known as "Dot operator" or "period operator".

```
#include<iostream.h>
#include<conio.h>
 Structure Demo
{
 int rno;
 Char nameenm[20];
 } dl[100];

void main()
{
 int i, ch;
for(i=0;; i++)
{
Cout<<"Enter Roll No:";
 cin>> dl[i].rno;
 Cout << "Name:";
 cin >> dl [i] nm;

 Cout <<" Cont -- press1:";
 cin >>ch;
if(ch != 1){
 p= i;
 break;
}
```

O/P→
Enter Roll No:21
Name- sh-
Cont -- press1

24/08/2020

```
for(i=0; i<=p; i++){
    cout<<"\n Roll No. = "<<dm[i].rno;
    cout<<"\t Name = "<<dm[i].nm;
}
getch();
}
```

CLASSES

for eg:

```
class point                    → header part
{
 int xVal, yval;               → body part
 public:
 void getpt(int, int);
 void offsetpt(int, int);
```

- A class definition consists of two parts: header & body.
→ The class header specifies the class name & its base classes.
→ The class body defines the class members
- Two types of members are supported: Data Members and member functions.
- Data Members have the syntax of variable definitions and specify the representation of class objects
- Member functions have the syntax of function prototypes & specify the class operations, also called the class interface.

25/08/2020

• class member fall under one of three diff. access permission Categories:
→ public members are accessible by all class users
→ private members are only accessible by the class members.
→ protected members are only accessible by the class members and the members of a derived class.

INSTANCE

once can have an instance ~~if the actual object~~ of a class or a particular object. The instance is the actual object created at runtime.
• The memory at which object works, that memory is known as instance.

PROPERTY OF CLASS:
• class is a user-defined property

OBJECTS:
• In object-oriented programming Language C++, the data and functions are bundled together as a self-contained unit called an object.

26/08/
Cr
• Once can inst
• Jus int
. O
. T
fo

26/08/2020

Creation of objects:-

- Once the class is created, one or more objects can be created from the class as objects are instance of the class
- Just as we declare a variable of data type int as:

```
int x;
```

- objects are also declared as:
  class name followed by object name;

```
exforsys e1;
```

- This declares e1 to be an object of class exforsys.

for eg. a complete class & object declaration:

```
class exforsys
{
private:
int x, y;
public:
void sum()
{
  ___
  ___
}
};
main()
{
  exforsys e1;
{  ==
```

program for class & objects:-

```
#include<iostream.h>
#include<conio.h>
  class Demo
{
 private:
int a, b, c;
public:
void get()
{
 cout<<" Enter the value of A & B";
 cin >> a>>b;
}
void show()
{
 c= a+b;
 cout<<" c=" << c;
}
};
 void main()
 {
  clrscr();
Demo d1; d2;
 d1. get();
 d1. show();

 d2. get();
 d2. show();
 getch();
}
```

CODE REUSABILITY:-

: Once a class has been written & tested, it can be adopted by other programmers to make their requirement fulfill. This is done by creating new class reusing the properties of the existing ones is called Reusability.

eg:

d1 → (a=70, b=80) → get → (c=a+b, c=150)

d2 → show (garbage) → without any val there is garbage

but if, d2 → (a=9, b=5) → (c=9)

Use of Scope Resolution:-

- To access global variable → It is completed

Sum of an array using class:-

```
# include <conio.h>
# include <iostream.h>
  class Demo
{
  int a[10], i, s;   // To run loop for sum
  public:
    void get();
    void put();
    void show();
};
```

To create objects of class

If we do not give this in public so our program will not run.

⇒ [If we want to provide body (definition) of function out of the class]

```
void Demo :: get()
{
for(i=0; i<10; i++){
cout<<"value=";
cin>>a[i];
}  // get function close
// Array is a linear DS coz addressing of array in sequence
// Array always start from 0.

// Data will be stored in a sequence like
// If a[0]=2020, a[1]=2022, a[2]=2024

void Demo :: pros()
{
s=0
for(i=0; i<10; i++){
    s= s+a[i];
}
}// pros function closed

void Demo :: show()
{
cout<<"\n sum="<<s;
}// show funct closed.

void main()
{
Demo d1;
```

[we cannot call get() funct directly coz it no. of class]

```
    d1.get();
    d1.pros();
    d1.show();
}
```

The finish largest element of an array using class

```
#include<iostream.h>
  class Demo
{
  int a[10], i, f, l;
  public
  void get()
{
  for(i=0; i<10; i++){
    cout<<"value=";
    cin>>a[i];
  }
}
  void pros()
{ Cursor should be the process when we will show
        f1)
}
  void show()
{
  cout<<"fL="<<f

void main(){
    Demo d1;
   d1.get();
```

```
    dl.pros();
    dl.show();
}
cout<<"first largest = "<<f<<endl;
}
```

Second largest Element of an array

```
#include<stdio.h>
#include<conio.h>
  void main()
{
 int a[10],i,f,s;
 for(i=0;i<10;i++){
   cout<<"value#";
   cin>>a[i]
}
  f=a[0];
  s=a[1];
for(i=1;i<10;i++){
if(f<a[i]){
   s=f;
   f=a[i];
} else{
  if(s<a[i]&& a[i]!=f)
{
  s=a[i];
}
}
}
cout<<"first largest= "<<f<<endl;
  cout<<" second largest ="<<s;
}
```

a[0] = 67
a[1] = 70
a[2] = 45
a[3] = 03
a[4] = 65
a[5] = 02
a[6] = 03

f= 67, s=70
f= 70, s= 67
f= 03, s=02

Bubble Sort

```
#include <iostream.h>
#include <conio.h
 void main()
{
 int a[10], i, j, s;
 for(i=0; i<10; i++){
  cout<<" values=";
  cin>>a[i];
 }
for(i=0; i<10; i++){
for(j=0; j<9; j++){
if(a[j]>a[j+1]){
   temp = a[j]
   a[j] = a[j+1];
   a[j+1] = temp;
  }
}
}
   for(i=0; i<10; i++){
    cout<<"\n value="<<a[i];
}
 getch();
}
```

Selection sort

```
#include <iostream.h>
#include <conio.h>
 void main()
```

```
int a[10], i, f, s, temp;
for(i=0; i<10; i++){
 cout<<"values=";
 cin>>a[i];
}
for(i=0; i<10; i++){
for(j=0; j<9; j++){
if(a[i] >a[j+1]){
temp= a[i];
a[i]= a[j+1];
a[j+1]= temp;
}
}
}
for(i=0; i<10; i++){
 cout<<"\n value="<<a[i];
}
getch();
}
```

Call by Reference

```
#include<iostream.h>
#include<conio.h>
 void change(int x[]) //x will work on address of
{
 int i;
 for(i=0; i<10; i++){
 x[i]= x[i]+10;
}
}
```

```
void main()
{ clrscr;
 int a[10], i;
 for(i=0; i<10; i++){
cout<<"values="; 10,20,30,40,50,60,70,80,90,100
 cin >> a[i];
}
  change(a);      //We are passing address of a
  for(i=0; i<10; i++){
  cout<<"\n values="<<a[i];
}
  getch();
}
```

Output → 20 30 40 50 60 70 80 90 100 110

```
// pointer
 int a= 5000;
```

// properties of
// a → 1) address → 2020
// 2) values → 5000

```
int *p;
```

// properties of
// *p → 1) address → 3030
// 2) value → add. of a[2020]
// 3) → 5000+3000

ress of a

```
*p = *p+3000;
```

```
cout<<"A="<<a<<endl;
cout<<*p="<<*p;
a= a+5000;
```

// 8000
// 8000

```
cout<<"A = "<<a;
cout<<"*p = "<<*p;

eg:-
int a = 2000;

int *p;
p = &a;
*p = *p + 4000;

int *t;
t = p;
*t = *t + 5000;

int **k;
k = &t;
**k = **k + 5000;
```

```
// properties of
// a → address → 2020
//     value → 2000

//*p → address → 3020
//     value → add. of a (2020)
//     2000 + 4000

//*t → add. → 4040
//     value → add. of a (2020)
//     6000 + 5000

//**k → add. → 5050
//      value → add. of t (4040)
//              add. of a (2020)
//              11000 + 5000
```

```
cout<<"\n A = "<<a;      16000
cout<<"\n*p = "<<*p;     16000
cout<<"\n*t = "<<*t;     16000
cout<<"\n**k = "<<**k;   16000
```

Q Exchange value of two pointers using call by reference using call.

```
#include <iostream.h>
#include <conio.h>
  Class swap
{
  public;
   int x;
  void swap (int*x, int*y)
  {
5000 z = *x;     5000
4444 *x = *y;    4444
5000 *y = z;     5000
};
  void main()
  {
   int a, b;
  cout<<" Enter the value of A & B";
   cin>>a>>b;      // a= 5000, b= 4444
   swap d1;
   d1. swap(&a, &b)
  cout<<" A= "<<a<<" \n B= "<<b;
   getch();
}
```

Q. Edit function

```
#include <iostream.h>
#include <conio.h>
  void edit1 (int *p, int size, int end); //function
  prototype.
  void main()
{
  Class r();
```

```
int ar[100], n, edi, i;
cout<<" Enter the size of an array ";
cin>>n;
for(i=0; i<n; i++){
cout<<" value = ";
cin>>ar[i];
}
cout<<" Editi ";
cin>>edi;
edit1(ar, n, edi);
cout<<" After editing output\n";
for(i=0; i<n; i++){
cout<<"\n value; "<<ar[i];
}
getch();
}
void edit1(int *p, int size, int ed)
{
int i;
for(i=0; i<size; i++){
if(*p==ed){
*p=*p+40;
}
p++;
}
}
```

// If there p is written instead of *p then, it will increase by address

```
#include<iostream.h>
#include<conio.h>
void edit(int* p, int size, int ed)  //funct prototype
{ char c;
int ar[100], n, edi, i;
cout<<"Enter the size of an array:");
cin>>n;
for(i=0; i<n; i++){
  cout<<"value=";
  cin>>ar[i];
}
  s= edit(ar, n);
cout<<"After function output\n");
cout<<"\n sum ="<<s;
  getch();
}
int edit(int* p, int size){
  int i, sm=0;
for(i=0; i<size; i++){
  sm = sm + *p;
  p++;    // it will increment the address
}
return sm;
}

#FUNCTION OVERLOADING
#include<iostream.h>
#include<conio.h>
{
```

```
int sum(int a, int b){
    return (a+b);
}
float sum(int a, float b){
    return (a+b);
}
float sum(float a, float b){
    return(a+b);
}
double sum(double a, double b){
    return(a+b);
}
void main(){
cout<<sum(344, 655.999f);
cout<<sum(4545, 5666);
cout<<sum(33344f, 777, 999f);
getch();
}
```

FRIEND CLASS AND FUNCTION;-

① friend class:- A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class

② friend function: A friend func of a class is defined outside that class scope but it has the right to access all private & protected members of class Even though the prototypes for friend functions appear in the class.

definition, friends are not member functions. A friends can be a function template, or member function or a class template

Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:

→ A method of a another class
→ A global function

following are some points about friend functions & classes:

1] friends should be used only for limited purpose. Too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in OOP.

2] friendship is not mutual. If class A is a friend of B, then B doesn't become a friend of A automatically.

3] friendship is not inherited.

4] The concept of friend is not there in Java

⇒ friend class in C++

· We can also use a friend class in C++ using friend keyword

eg → class classB;
Class classA {
// classB is a friend class of classA
friend class classB;
}
Class classB

- When a class is declared a friend class, all the member class, call the member function of a friend class become friend function

- Since class B is a friend class, we can access all member of class A from inside class B

- However, we cannot access members of class B from inside class A it is because friend relation in C++ is only granted, not taken

```
#swap function
#include<iostream.h>
#include<conio.h>
  class B;
  class A;
{
private;
int a;
friend void swap(A&t1, B&t2);
public;
void get(){
 cout<<"A=";
  cin>>a;
}
  void show(){
  cout<<"A="<<a;
}
;// A class close
  class B
{
```

```
    private;
    int b;
    private void swap(A &t1, B &t2);
public;
    void get(){
    cout << "B = ";
     cin >> b;
    }
    void show(){
    cout << "B = " << c;
    }
}; // B class close
    void swap(A &t1, B &t2){
     int x;
    x = t1.a;
    t1.a = t2.b;
     t2.b = x;
    }
     void main(){
     A P1;
     B P2;
P1.get(); // calling get funct of A class
P2.get(); // calling get funct of B class
     swap(P1, P2);
      P1.show();
      P2.show();
    }
```

Constructor & Destructor

1. Constructor
   - It always defined in public section

- It donot have any return type
- It constructs the memory of object

eg→
```
#include<iostream.h>
#include<conio.h>
class Demo{
int a;          // By default private
public;
Demo(){
a=9000;
}
void show(){
cout<<"A="<<a<<endl;
}
};// class close
void main(){
Demo d1, d2, d3;
d1.show();
d2.show();
d3.show();
getch();
}
```

- They should be declared in public section
- They are invoked automatically when objects are created
- They do not have any return type not even void therefore they cannot return values
- They cannot be inherited, though a derived class can call the base class constructor

→ The other C++ functn, they can have default arguments

- Constructors can be virtual.
- We cannot refer to their addresses.

```
#include <iostream.h>
#include <conio.h>
    int a=0;
      class Demo
{
  public:
  // Constructor
   Demo(){
  cout<<"\n Constructor Working"<<++a;
}
  // Destructor
  ~ Demo(){
    cout<<"\n Destructor Working "<<a--;
  }
}; // class close
    void main(){
  cout<<"\n Main function start\n");
   Demo d1, d2, d3, d4;
  { cout<<"\n first Block start \n";
   Demo d5, d6;
  cout<<"\n first Block close \n");
}
  cout<<"\n Main function close\n";
}
  getch();
```

STATIC VARIABLES & FUNCTIONS

:- They can work on their memory, they do not need any object memory for execution but when we call them by the help of an object then they can share a copy of their memory with object, static function can use only static variables.

```
eg→ class Demo
    {
    public;
    int a;
    void get(){
      cout<<"A=";
      cin>>a;
    }
    void show(){
    cout<<"A="<<a<<endl;
    };
    void main(){
    Demo d1, d2, d3;
     d1. get();
     d2. get();
     d3. get();
     d1. show();
     d2. show();
     d3. show();
    getch();
    }
```

O/P

// A is working on memory of object then O/P will be;

if A=1000
A=2000
A=3000
then A=1000
B=2000
A=3000

← // If we insert int Demo::a, here then we are giving memory to A

A=1000
A=2000
A=3000
⇒ A=3000
A=3000
A=3000

s
they do not
tion but
an object
ist memory
e only

q on memory of
n o/p will be;
00
00
00
000
2000
000

no::a; here
to A

① show that static function can work on their own memory.

```
class Demo
{
public;
static int a;
void get(){
cout<<"A=";
 cin>>a;
}
 void show(){
cout<<"A="<<a<<endl;
};
int Demo::a;
 void main(){
 Demo d1, d2, d3;
d1.get();
d2.get();
d3.get();
d1.show();
d2.show();
d3.show();
cout<<"Main A="<<Demo::a; // Using :: calling static variable
getch();
}
```

#REFERENCE VARIABLE

Once reference is initialized with a variable, either the variable's name may be used to

Or The variable which refer to the memory location, is called Reference variable

```
Ex→ int x = 9000;
    int &f = x;   // ref is refrence to x
// value of x is not changed to 2000
    f = 2000;
    cout<<" x=" <<x<<endl;
// value of x is now changed to 30,000
    x = 30,000
    cout<<" ref = "<<f<<endl;

// Declare Simple Variables
int i;
double d;
int &r = i;        // declare reference Variables
Double &s = d;
   i = 5;
cout<<value of i:"<<i<<endl;
cout<<"value of i reference:"<<r<<endl;
   d = 11.7
cout<<"value of d:"<<d<<endl;
cout<<" value of d reference:"<<s<<endl;

eg → void swap(int &first, int &second)
{
   int temp = first;
   first = second;
   second = temp;
}
```

```
int main(
{
 int a = 2
 swap(
 cout<<
  retur
}
// copy
 Copy c
 is u
 objec
 x(x&
 provi
 the
 synt
  cl
 {
 }
And
 ca
 ob
 Me

 eq
 #
```

```
int main( )
{
 int a=2, b=3;
 swap (a, b);
 cout<< a<<" "<<b;
 return 0;
}
```

COPY CONSTRUCTOR:-

Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type it is usually of the form x(x&), where x is the class name. The compiler provides a default copy constructor to all the classes.

syntax;

```
classname(const classname &objectname){
{---
}
}
```

And it is used to create an object, Hence, it is called a constructor. And, it creates a new object, which is exact copy of the existing copy. Hence it is called copy constructor

eg:-
```
#include <iostream.h>
#include <conio.h>
class Democopy
{
 private;
 int x, y; // data members
public;
Dis_
```

Kalwati & ...

```
    x = x1;
    y = y1;
}
// Copy constructor
DemoCopy(const DemoCopy &sam){
    x = sam.x;
    y = sam.y;
}
void display(){
    cout<<" x = "<<x<<" y = "<<y<<endl;
}
};
// Main function
void main(){
DemoCopy obj1(10, 15); // Normal Constructor
DemoCopy obj2 = obj1;  // Copy Constructor
cout<<" Normal constructor:";
obj1.display();
cout<<" Copy constructor:";
obj2.display()
getch();
}
```

⇒ String function:-

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<string.h>
    void main()
{   char nm[10];
    int n;
    cout<<"name=";
```

```
    gets(nm);
    n = strlen(nm);
    cout<<"Length = "<<n;
    getch();
}

#Without inbuilt function:-
#include<stdio.h>
#include<iostream.h>
#include<string.h>
  int length(char nm[])
  {
   int t=0;
for(i=0; name[i]; = '\0'; i++)
   { t++;
  }
return t;
}

  void main()
{
  char nm[10];
   int n;
cout<<"Name = ";
   gets(nm);
n = length(nm);
  cout<<"Length = "<<n;
  getch();
}
```

=> Header files:-

```
class strfun
{
public:
int length (char name[])
{
int i, t = 0;
for (i=0; nam[i] != '\0'; i++)
{
    t++;
{
return t;
}
};
```

```
#include <strfun.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main()
{
 char nm[20];
  int n;
cout<<"Name=";
 gets(nm);
strfun d1;
n=d1.length(nm);
cout<<"length="<<n;
}
```

Using class making strcmpi function:-

STRFUN.H

```
class strfun
{ public:
  int strcmp(char st1[], char st2[])
{
  int i, n=0;
for(i=0; st1[i]!='\0'||st2[i]!='\0'; i++)
  {
   n= st1[i]= st2[i];
    if(n!=0 && n!=32 && n!=-32)
   { break;
  }
}
if(n==32||n==-32)
  {
    n=0;
  {
  return n;
};
```

New file

```
# include<string.h>
# include<conio.h>
# include<stdio.h>
# include<iostream.h>
# include<STRFUN.H>
  void main()
{
  char nm[20] sn[20];
    int n;
  cout<<"Name=";
```

```
gets(nm);
cout<<"SN=";
get(sn);

//n = strcmpi(nm, sn);
  strfunc' d1;
n= d1.strcmpi(nm, sn);
  cout<<"N="<<n;
  getch();
}
```

O/P;-
Name = Amit
SN = Amit
n = 0
or
Name = AMIT
SN = Amit
n = 0.

⇒

```
#include<string.h>
#include<stdio.h>
#include<iostream.h>
void main()
{
  char nm[20], sn[20], ca[40];
  int n;
  cout<<"Name =";
   gets(nm);
   cout<<"SN=";
    gets(sn);
 strcpy(ca, strcat(nm sn));
   cout<<"CA = "<<ca;
  getch();
}
```

⇒ Operator overloading in binary operator;-

```
#include<iostream.h>
#include<conio.h>
  class demo
{
 public:
 int sal[10],i;
{ coun<<"salary";
    cin>>sal[i];
 {
{
 void operator++()
{
 for(i=0;i<10;i++)
{ sal[i]=sal[i]+2000;
 }
}
 void show()
{
 for(i=0;i<10;i++)
 {
  coun<<"\n sal="<<sal[i];
  {
}
};
  //class close
```

```
void main()
{
  demo d1;
  d1.get()
  d1++;
  d1.show()
  getch();
{
```

=> We have written '++' on the instance of demo class so it will shift at operator (

```
#include<iostream.h>
class demo
{
 public;
int sal[10],i;
void get();
{
for(i=0; i<10; i++);
{ cout<<" salary=";
  cin>>sal[i];
 }
{ demo operator++(demo&p1)
  {
   demo d1;
for(i=0; i<10; i++)
{
t1.sa[i]=sal[i]+p1.sal[i];
  return 1;
 }
void show()
{ for(i=0; i<10; i++)
 { cout<<"\n sal =" << sal[i];
  }
}
};  //class end.
```

```
void main()
{
demo d1,d2,d3;
 d1.get();
 d2.get();
 d3=d1+d2;
 d3=show();
 getch();
{
```

O/P:-

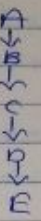
salary=1 2 3 4 5 6 7 8 9 10

salary = 11 11 11 11 11 11 11 11 11 11.

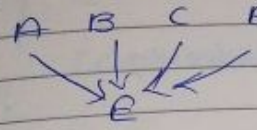
=) Types of Inheritance:-

1) single Inheritance:-

(2) Multiple-level Inhi~



(3) Multiple Inheritance;-



# class A: public B
{ // class A now inherits the members of class B
// with No changes in the "access specifier" for
{ // the inherited member.

public Base class(B) derived class(A)

public members ⟶ public
protected members ⟶ protected
private members ⟶ inherited but not accessible

# class A: protected B
{ // class A now inherits the members of class B
// with public members promoted" to protected
// but no other changes to the inherited members,
{

protected Base class (B)
public members ⟶ protected
protected members ⟶ protected
private members ⟶ inherited but, not accessible

derived class

# class A: private B
{ // class now inherits the members of class B
// with public & protected members "promoted to private".
}

private Base class (B)
public members ⟶ private
protected members ⟶ private
private members ⟶ inherited but not accessible

derived class (B)

include <iostream.h>
class Base
{ public:
int q[10], i, j;
void get()
{
for (i=0; i<10; i++)
{
cout << "value=";
cin >> q[i];
}
}
};

```
class subi public Base
{
public:
 void show()
{  j=0
for(i=0; i<10; i++)
 { j= j + a[i];
 {
  cout <<"value=" <<j;
 }
};
 void main()
 {
  sub s1;
  s1. get();
  s1. show();
   getch();
 {
```

⇒ How to pass value from derive class constructor to base class constructor;-

```
#include<iostream.h>
  class Base
{
  public:
  int j;
 Base(int x)
 {  j = x;
{;
```

```
class sub: public Base
{  public:
    int p;
  sub(int x, int y): Base(x)
    // passing value from sub class const to
    Base class constructor
  {  p = y;
  }
   void show()
{  cout<<"\n sum ="<<j+p;
}
   void main()
{  sub s1(340,670);
     s1.show();
     getch();
   }
```

passing values from sub-class constructor to base class constructor;-

```
Class Base
{
 Public;
  int x;
 Base(int a)
  {  x = a;
  }
};
   Class sub; pubic Base
    { public:
      int y;
```

• Constructor always work top to down

```
Sub(int a, int b): Base(a)   //passing value
{ y= b;
}
void show()
{
  cout<<"sum="<<x+y;
}
};
  void main()
{ Sub d1(200,300
    d1.show();
  getch();
}
```

(2) Multi-level Inheritance:

```
class Upper
{ public:
```

```
class Mid: public Upper
{ public;
  void pros()
{ f1 = a[0];
  for(i=1; i<10; i++)
{
  if(f1 < a[i])
    { f1 = a[i];
}
}
};
```

```
class lowers: public mid
{ public:
  void show ()
{
  cout << "fl = " << fl;
  }
};
  void main ()
{ lower dl;
  dl.geh ();
  dl.pros();
  dl.show ();
}
```

(3) Multiple Inheritance

When multiple classes will inherit in a single class, this type of inheritance is called multiple Inheritance

```
#include <iostream.h>
  class A
{ public:
    void m1 ()
  { cout << "I am M1" << end;
  }
};
  class B
{ public;
    void m2()
  { cout << "I am M2" << end;
}
};
```

```
class C
{
public:
 void m3()
{
 cout<<"I am M3"<<endl;
}
};
class MDemo: public A, public B, public C
{ public:
  void show(int x)
{
 x==1)?m1():(x==2)?m2():m3();
}
};
void main()
{
MDemo d1;
d1.show(2);
getch();
}
```

```
void main()
{ int p;
 cout<<"Enter Values";
    cin>>p;
 MDemo d1;
    d1.show(p);
 {
```

single
ed

Hierarchical Inheritance -

```
#include<iostream.h>
   class A
{
 public:
  int a,b;
  void get()
{
```

```
    cout<<" Enter the value of A and B"<<
    cin>>a>>b;
  }
};
class sum; public A
{ public:
   void sum ()
  {
   cout <<" sum = "<<a+b<<endl;
  }
};
  class Multi publicA
{ public:
  void mult ()
  { cout <<" Mult ="<<a*b<<endl;
  };
};
  Class Div public A
{ public:
   void div()
  { cout<<" Div ="<<a/b<<endl;
};
};
  void main ()
{


  getch ();
{
```

Virtual function:-

We just discovered that a base pointer, even when it is made to contain the address of a derived class. The compiler simply always executes the function in the base class. The compiler simply ignores the content of the pointer and chooses and the member function that matches the type of pointer.

O/P:- We must access virtual fn through the use of a pointer declared as a pointer to the base class. Why can't we use the object name with dot operator the same way as any other member function to call the virtual function? but we can remember, runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class

```
#include<iostream.h>
  class Base
{ public:
  virtual void m()
{ cout<<"Base class fn"<<endl;
}
};
  class sub: public Base
{ public:
    void m()
{ cout<<"Sub class fn"<<endl;
}
};
```

```
void main()
{  Sub s1;
   Base b1;
  Base * P[2];
   P[0] = & b1;
   P[1] = & s1;
   P[0] -> m1();
   P[1] -> m1();
  getch();
{
```

#Pure virtual function:-
If you have any pure virtual function then your class is abstract and you can not create its object instance we can not give definition to pure virtual function.

```
#include<iostream.h>
#include<conio.h>
   class Base
  {
   public:
  virtual void abc()=0;  // Pure virtual fundia
   virtual void m1()
  {
   cout<<"Base class funct1"<<endl;
   }
};
  void main()
{
   clrscr();
     Base b1;
```

```
b1.m1();
 getch();
}
```

// Now your class is abstract

#Temp
#Indu
tem
T
{
out
{
int
{ in
f
ch
c
ci
Co
cou
c
c
Co
c
v
e
{

Template function: -

```
#include<iostream.h>
template<class T>
T large(Tn1, Tn2)
{
return (n1>n2)? n1: n2;
}
int main()
{ int i1, i2;
  float f1, f2;
  char c1, c2;
  cout<<" Enter two Integers:\n";
cin>>i1>>i2;
cout<<large(i1, i2)<<" is large."<<endl;

cout<<"\n Enter two floating-point numbers:\n";
cin>>f1>>f2;
cout<<large(f1, f2)<<" is large."<<endl;

cout<<"\n Enter two characters:\n";
cin>>c1>>c2;
cout<<large(c1, c2)<<" has large ASCII
value";

return 0;

}
```

```
#include<iostream.h>
template<typename T>
void swap(T& n1, T& n2)
{
  Temp;
  temp=n1;
   n1 = n2;
   n2 = temp;
  }
int main()
{
  Int i1=1, i2=2;
  float f1=1.1, f2=2.2;
  char c1='a', c2='b';

cout<<"Before passing data
to fundu template.\n";
cout<<"i1="<<i1<<"\ni2="<<i2;
cout<<"\nf1="<<f1<<"\nf2="<<f2;
cout<<"\nc1="<<c1<<"\nc2="<<a;

  swap(i1, i2);
  swap(f1, f2);
  swap(c1, c2);

cout<<"\n\nAfter
passing data to fu. tem-
plate\n";
```

```
cout<<"i1="<<i1<<"\ni2
 "<<i2;
cout<<"f1="f1="\nf2
"<<f2;
cout<<"\nc1="<<c1<<"\n
 c2="<<c2;

return 0;
}
```

Template class

```
#include<iostream.h>
using namespace std;
template<class T>
class Calculator
{
private:
T num1, num2;
public:
Calculator(T n1, T n2)
{
num1 = n1;
num2 = n2;
}
void displayResult()
{
cout<<" NO. are:"<<num1<<" and "
<<num2<<" : "<<endl;
cout<<" Addition is:"<<add()<<endl;
cout<<" subt is:"<<subt()<<endl;
cout<<" product is:"<<multiply()<<endl;
cout<<" Division is:"<<divide()<<endl;
}
T add(){return num1+num2;}
T subtract(){return num1-num2;}
T multiply(){return num1*num2;}
T divide(){return num1/num2;}
};
```

```
int main()
{
Calculator<int> int calc(2,1);
 Calculator<float> float. Cal
          (2.4, 1.2);
Cout<<"int results:"<<endl;
 int calc display Result();

Cout<<endl<<"float results" <<endl;

float calc. Display Result();

return 0;
}
```

Exceptions, using Handling Devices;-
Exceptions are abnormal conditions which comes during runtime.

```
#include<iostream.h>
 void main()
{
 clrscr();
int a=40, b=3, c;
 try{
for(i=0; i<5; i++)
{
 c=a/b;
 Cout<<"c=" <<c<<endl;
 b--;
 if(b==0)
 {
  throw 'a';
```

```
}
}
}
  catch (char a)
{
cout<<" catch working for exception "<<endl;
{
cout<<" Hello";
getch();
}
```

Exceptions →
Exceptions are those abnormal conditions that Happens occurs on runtime

comes
```
#include <fostream.h>
#include <conio.h>
  void main()
{
  clrscr();
int a=40; b=3, c;
  for(i=0; i<5; i++)
{
  c= a/b;
  cout<<" c="<<c<<endl;
    b--;
}
  getch();
{
```

o/p→ c = 19 40/3
c = 20 40/2
c = 40 40/1
Divide error
→ this is arithmetic exceptions.

→ to handle such type of exceptions, we use exceptions handling devices

Exceptions Handling ~~function~~ Devices →

① Try → The code in which there is a problem

② Catch → the code which contains the solution of the problem (Handle the exceptions)

③ Throw → using this we can generate the exception and put it into the catch block.