7 August 23

C++

* C++ & C Both are Middle level language.

* Middle - Level Language is the combination of both Low - level and high - level language.

Low - level	High - level
→ Develops system software. • Compiler • Assembler.	→ Develops Application Software.
⇒ It converts high-level code into machine code.	→ Code english like Instructions.
→ C / used by both C / C++.	→ Learn easily understand & easily human readable.

* C++ is an **Object - Oriented**
  ↳ It is an Approach or Paradigm.

* Procedural Approach / Structured - Oriented Approach
  ↳ It follows by 'C' language

⇒ Features :-

1) Top - Down ↓
2) Focus on Functions.
3) Used for designing medium-size application.
4) Code - Optimization
5) Less secure.
6) Code Reusability Absent

* Structure of a 'C' Program :-

1.) # Pre-Processor Directives
Macros → Function Declaration Section

2.) Main ( )
→ Variable Declaration
→ Data Instructions / Code.

9/8/23

* Advantages to make a function :-

1.) Easy to understand.
2.) Easy Readable
3.) Easily manageable.
4.) Easily maintable.
5.) Easy to rectify errors.
6.) They are providing the reusability of code.

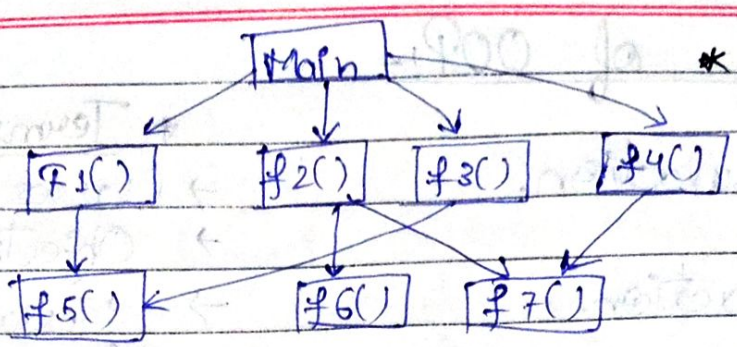
* Limitations of functions :-

Calling overhead ~~of~~ {
If program length is shortest than
1.) Memory and time overhead due to stack frame allocation and transfer of program control.
2.) Cannot return multiple values.

* Procedural - Oriented Approach :-

1.) Follows Top-Down. [Outline [main] then Detailing the function]
2.) Programs are divided into Sub-programs ~~→ Fund~~
3.) Focus is on doing things not on data. (Drawback)
Functions Functions
4.) Less secure data.
5.) Data is moving freely around function.
OOP

manipulate



* A function can call another function.

Variables :-

Local Variable

→ int f1()
{
int d;
}

→ Its scope is within that function

→ Lifetime.

Global Variable

→ int c;
main()
{
int a, b;
f1();
}

→ Its scope is through the whole program.

→ Lifetime (Till the Program exist).

* OOP Approach :-

(Classes are defined first) Detailing First then outline [main]

1) Follow Bottom-up Approach.
2) Programs are divided into classes & objects.
3) Focus is on data.
4) More security of data.
5) Data is binded tightly.

* Global data is less secure. It can be accessed by any function. So, it is easy to manipulate, So it is difficult to track.

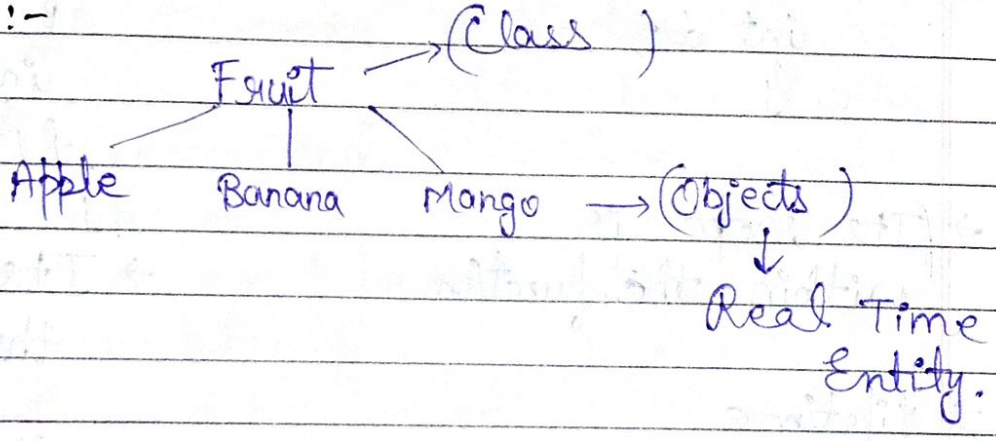
* Pillars of OOP:-

1) Encapsulation.
2) Abstraction.
3) Inheritance.
4) Polymorphism.

* Terms of OOP:-

→ Classes
→ Objects
→ Message Passing
→ Dynamic Binding

⇒ Class :-



10/8/23

Lab Work

Q1. Write a Program to find second largest element in array. [Using Functions]

Q2. WAP to print Fibonacci Series with or without recursion.

```
main ()
 {
  int num[5];
  int size / largest;
printf("Enter size of array");
  scanf ("%d", & size)
  printf("Enter Array Elements");
    for( i=0; i<size ;i++)
    {
    scanf("%d", &num[i]);
    }
   printf("Array");
    for (i=0; i<size ; i++)
    {
    printf("%d\n", num[i]);
    }
   largest =num[0];
    for(i=0; i<size ; i++)
     {
   if (num[i] > largest)
     {
      largest = num[i];
     }
   }
   printf("Largest element in array is %d", largest);
    }
```

10/01/23

* Classes

1.) Group of objects having similar properties & behaviour.

Car →	Properties	Behaviour / action / Operation
	model	- Calculate speed
	Price	- Shift gear
	Milage	- Cal average

2.) It is just a Template [Memory is not allocated.]
~~Blue-Print~~

3.) It is an user-defined data type.

* Objects

1.) Real - Time Entity (Physically exist).

2.) It is an instance of class.

3.) It is a variable of type class.

* POP [Procedural Oriented Programming]

1.) It follows top-down approach.
2.) It divides program into sub programs known as functions.
3.) It ~~follow~~ focus more on functions rather than data.

OOP [Object Oriented Programming]

1.) It follows bottum-up approach.
2.) It divides Programs into classes & objects.
3.) It focus more on data.

4) It is less secure.	4) It is more secure.
5) It is less closer to real-world.	5) It is more closer to real-world.
6) Data moves freely around functions.	6) Data is binded closely within objects.
7) It is more helpful to develop medium size application.	7) More complex application can be developed easily.
8) In this no data hiding.	8) In this data hiding feature is present.
9) Code-reusability is upto less execute.	9) Code-reusability Inheritance.
10) Encapsulation, abstract inheritance & polymorphism features are not there.	10) They have features.
11) There is no concept of function & operator overloading	11) Function & operator overloading is there.
12) Languages that follows procedural approach are C, BASIC, FORTRAN, PASCAL	12) JAVA, C++, PHP.

*	C	C++
1)	It is based on Procedural oriented programming approach	1) Based on OOP.
2)	It is a subset of C++.	2) It is a superset of C.
3)	Follows Top-down approach.	3) Follows Bottom up approach.
4)	Developed by Dennis Ritchie.	4) By Bjarne Stroustroup in 1980's in AT&T Bells labs, US
5)	No concepts of classes & objects.	5) The entire concepts revolve around classes & objects.
6)	No concepts of access specifiers.	6) Concept of access specifiers like private, public, protected.
7)	We cannot define functions inside structure	7) Can define functions inside structure.
8)	It has 32 keywords	8) It has 63 keywords.
9)	No concept of exception handling.	9) There is using try & catch blocks.

10.) No concept to function & operator overloading.	10.) There is concept of operator overloading.
11.) It is less closer to real world.	11.) More closer to real world.
12.) No concept of encapsulation, inheritance, polymorphism (pillars).	12.) There is concept of pillars like encapsulation, inheritance, polymorphism.

11/8/23

* Benefits of OOP :-

1.) Data is more secure. (Unnecessary details are hiding)
2.) Data hiding / Abstraction → Information Details.
   ↳ by diff. access specifiers [Public, Private, Protected]
3.) Closer to real-world
4.) With help of inheritance, [code reusability is achieved.]
   (reduces development & testing time reduces efforts.
5.) With the help of encapsulation (data is binded closed with functions).
   [data is not moving freely].

* Applications of OOP:-

1) We can develop any application software.
2) System software.
3) AI & Expert system.
4) Gaming Applications.
5) CAM / CAO.

★ C++ (It is a case sensitive)

→ Original Name Was "C with classes."
→ It is Extension of C.
→ Developed by Bjarne Stroustroup in 1980's, AT & T Bell labs.
→ It is an OOP but not a pure object oriented. (Because we can declare ~~All contents~~ all primitive datatypes but not completely).
→ Small Talk is a Pure Object oriented.

→ Procedural
Object-oriented
Generic Programming / Templates

} C++ supports these paradigm.

* Structure of C++ Program :-

Include header Files
Declare classes / functions
Define Function Definitions.
Main ().

C++
↳ 1) Character Set
[A-Z, a-z, 0-9]
2) Words / Tokens.
(In English) | (In languages Programming)
↳ Smallest unit of Programs.

Tokens → Constants
→ Variables
→ Operators
→ Keywords

1) Constants :- Value remains same throughout the program and not changed.

5, 10 (Integer Constants), hello (Character Constants), 4.5 (Float Constant), special symbol.

Symbolic Constants :- Throughout the programs values are not changed.

such as :- # define PI 3.14
(Assigned to Value remains same)
(Instead of writing 3.14 as same as We can write PI.)

2) **Variables** :-

↳ User - define Name / ~~Speci~~ Identifiers

- Variable
- Function
- Array

* **Rules** :-

1) Name can be combination of ~~dat~~ 'letters', 'digits', underscore '_'.

2) **Variables** :- Vary Value

↳ User-define Name / ~~Spec~~ Identifiers

- Variable
- Function
- Array

* **Rules** :-

1) Name can be combination of ~~Das~~ 'letters', digits, underscore '_'.
2) Name should be meaningful and should not be lengthy.
3) Can't start with the digits.

16/8/23

* **Character Set**

A - Z,
a - z
0 - 9
Special Characters, $ _ _ .

Words / Tokens

↳ Smallest unit of Individual Programs.

Tokens —— Constants [Can't be changed during execution of program —— Integer, Real, Character

Literal Constants (Assigning to some variable)

Variables

Keywords

Operators

* Integer - 2 bytes } size
* Float - 4 bytes

3) Keywords :- Reserve Words (Approx 63 Keywords)
[Words whose meaning is already known to compiler.]
Such as if, else, try, catch, class, final etc.

4) Operators :- a) Arithemetic [+, -, *, /, %]
Acts/helps to do Operation on the operands
Binary Operator (Requires minimum two Operands).
/ and don't take decimal value
% Works only on integer value, Doesn't work on float

b) Relational Operator (comparison)
<, >, ≤, >=, ==
Used to test condition
Eg:- ⇒ if (age < 18) {
printf("Valid to Vote");

⇒ if (a > b)
{
c = a + b;
}

⇒ if (x == 18)
{
printf("ABC");
}

c) Logical Operator (Logical And) (OR) (NOT)
:- &&, ||, !
Used to combine the condition
Eg:- if ((age > 18) && (gender == 0))
{
printf("can vote");
}

* Associativity:- how it will be evaluate. or direction of evaluation.
L → R or R → L.

NOTE:- Any non-zero value in C/C++ is considered as True.

Truth Table of AND

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

NOT

a	b
0	1
1	0

d) Assignment Operator :- (To assign the value)
=, +=, -=, *=
Associativity (R to L)
Compound assignment Operator

Eg:- x += 5;

e) Conditional / Ternary Operator :-
It takes 3 Operands
Shorthand Notation of if-else.

x = 20;
y = 40;
x > 20 ? x : y

If condition is True → '?' [Value after '?' will print]
If condition is False → ':' [Value after the 'colon' will print]

Eg:-
```
if (x > 20)
  {
  printf("%d", x);
  }
else
  {
  printf("%d", y);
  }.
```

f) Bitwise Operator :- That works on bit-level. [0/1].

Bitwise AND :- & [Convert the into Binary then perform Bitwise AND.]
OR :- |
XOR :- ^
Shift Left :- <<
Shift Right :- >>
NOT :- ~

Eg:- 4 & 2

2	4
2	2 — 0
	1 0

2	2 — 0
	1 0

4 → 100 2 → 010

4 → 0 0 0 0 0 1 0 0
2 → 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 AND
1 1 0 OR

→ Shift left

Eg:- $8 << 2$ → No. of Bits

⇒

2	8
2	4 — 0
2	2 — 0
	1 — 0

Shift 2 ~~Shift~~ Bits

1

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0 → Output

↓ Add

8 → 1000

17/8/23

Lab Work :-

Q4 Factorial of a No. with & without recursion.

Q5 Print table of a No. — 2.

2 x 1 = 2

17/8/23

f) * Bitwise Operator

→ Bitwise shift Right " >> "

Eg:- 8 >> 2

2	8
2	4 — 0
2	2 — 0
	1 — 0

shift Right

0 0 0 0 1 0 0 0

0 0 0 0 0 0 1 0 = o/p.

g) * New & Delete Operator :-

↳ Used for ~~moving~~ memory management.
Used to dynamically allocate memory.
[malloc() , calloc() , free()]

h) * Comma Operator :- Used to separate different variables.

i) * Scope Resolution Operator :- [::]

↳ This function belongs to which class?
↳ Local variable to global variables
↳ to distinguish b/w both
↓
Both Exists

```
int a = 10;
 main()
{
  a = 20;
print → a = 20
        ∴ a → 10
}
```

③ Unary Operator [Requires single operand]

1) Pre / Post Increament ⟶ + 1

2) Pre / Post Decreament ⟶ - 1

Pre (First It will increase then assign).

Post (First It will assign then increase).

Pre

```
int x=10;
y= ++x; — 11
printf("%d", y); — 11
printf("%d", x); — 11
```

Post

```
int x=10;
y=x++; → 10
printf("%d", y); → 10
printf("%d", x) → 11
```

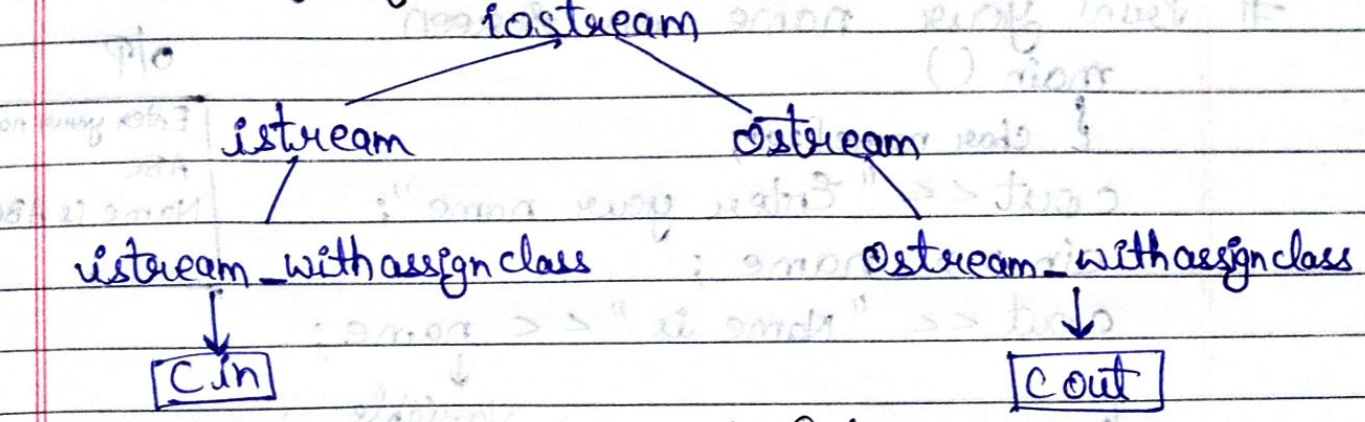
* Precidence :- Priority of an every operand.
  More Priority of any operand → Evaluate First
  low Priority → Evaluate last.

* Sentence / Instruction
  (In English) (In Programming language)

1) Input / Output Instruction

Objects { Cin, Cout

★ Hierarchy of Classes

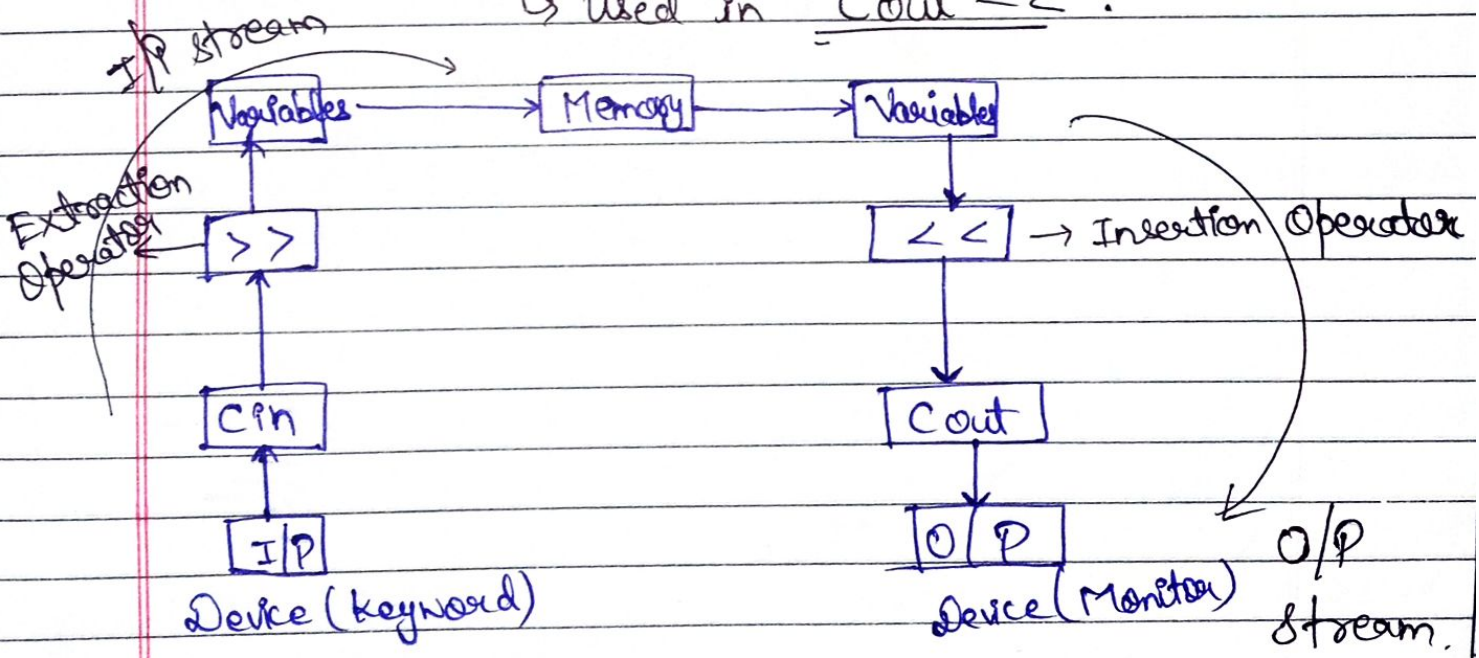


Stream = Flow of Data
It is a sequence of bytes that are flowing.

>> → Extraction Operator
↳ used in Cin >>.

I/P Stream :- Data is moving till memory.

<< → Insertion Operator
↳ Used in Cout <<.



Print your name on screen

```
main ()
{ char name[10];
cout << "Enter your name";
cin >> name;
cout << "Name is" << name;
}
```

name → Variable Name

o/p

```
Enter your name
ABC
Name is ABC
```

18/8/23

Q2 Display Hello

```
#include <iostream>
main()
{
cout << "Hello";
}
```

Output

Hello

Q3 Read 3 variables from user of diff. types. display on screen.

```
main()
{ int num_1; num_2; float num_3;
cout << "Enter First no." << endl;
 cin >> num_1;
cout << "Enter Second no." << endl;
 cin >> num_2;
cout << "Enter Third no." << endl;
 cin >> num_3;
```

* It is a manipulator to move the cursor to the next line.

cout << "First No. is" << num_1 << endl;
cout << "Second No. is" << num_2 << endl;
cout << "Third No. is" << num_3 ;
}

Q4. Write a Program to add 2 No. by initializing number.

```
main ()
{ int ans;
int num1 = 10 , num2 = 20;
cout << "First no. is" << num1 <<endl;
cout << "Second no. is" << num2 <<endl;

ans = num1 + num2 ;
cout << "Sum is" << ans;
      OR
cout << "Sum is" << num1+num2;
}
```

```
main ()
{
int num1, num2;
CO
```

Q5. Find area of Circle

```
#define PI 3.14
main ()
{
int radius , Float area ;
cout << "Enter Radius of Circle" << endl;
cin >> radius ; (OR PI)
area = 3.14 * radius * radius;
cout << "Area of Circle is" << area;
}
```

5! = 5x4x3x2x1

*CPP stands for C++.

Dynamic Initialization

(Declaring at the ~~need~~ time of need).

float area;

Or

float area = 3.14 * radius * radius.

Q6 Write a Program to find the factorial of a number.

⇒

```
#include <iostream>
int main()
    {
    int num, fact =1, i;
cout << " Enter a number";
cin >> num;
for(i=1; i<=num; i++)
    {
    fact = fact *i;
}
cout << "factorial of Number is" << fact;
}.
```

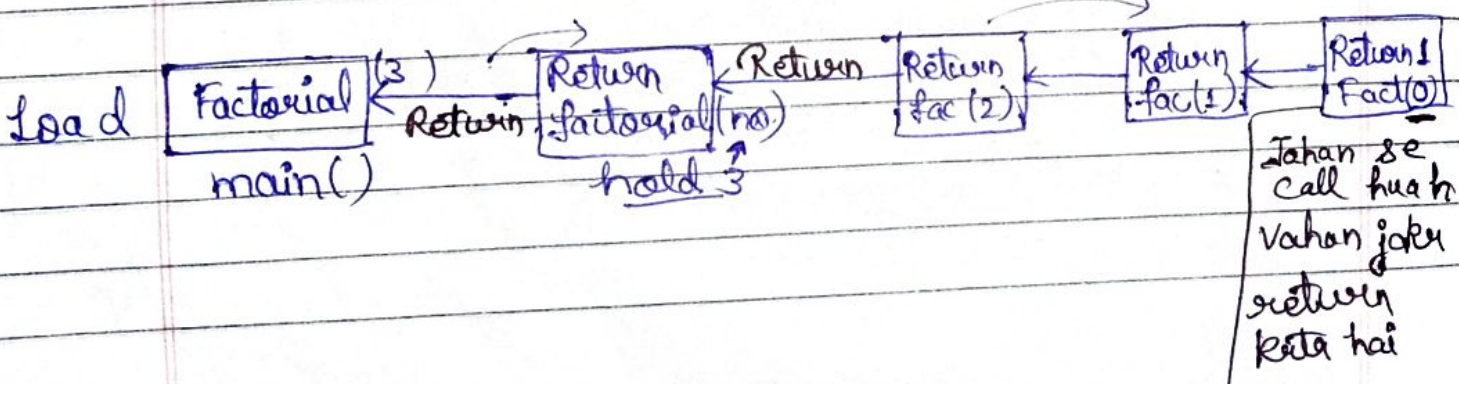
21/8/23

Q. WAP to find factorial of no. using functions.

```
int factorial (int)
main ()
{
    int ans;
    int num;
cout << " Enter a no.";
Cin << num;
ans = factorial (num); // Calling.
                       ↑ Actual Arguments
cout << " factorial of a no. is " << ans;
}
int factorial (int no)   // Function Definition
   ↑ Input Data Type [Why we have to receive]
return type   local variable of fact. function   (Formal Arguments)
{
 int i, fact = 1;
for (i = 1; i <= no; i++)
{
 fact = fact * i;
}
return fact;
}
```

fact = fact * i
1 = 1 * 1
2 = 1 * 2
6 = 2 * 3
24 = 6 * 4
120 = 24 * 5

Return only one value



Recursion:- [Use Stack DS].

⇒ When function calls itself again & again.
⇒ It is important to give terminating condition.

Code:-

```
#include <iostream>
main()
{
int num; int ans;
cout << "Enter a No";
cin >> num;

ans = factorial (num);
cout << "Factorial of a no. is" << ans;
    }
int factorial (int no);
    {
if (no==0)
    {
    return 1;
    }
else {
    return (no * factorial (no-1));
    }
}.
```

Terminating Point

Factorial (0)
Factorial (1)
Factorial (2)
Factorial (3)
main ()

22/8/23

Lab Work

Q. WAP to find string is palindrome or not.

⇒

```
#include <iostream.h>
#include <string.h>
using namespace std;
int main()
{
  char inp[10];
  int length;
  int flag=0;
  int i=;
  cout << "Enter a string" <<endl;
  cin >>inp;
  length = strlen(inp);
  for(i=0; i<length; i++)
{
if(inp[i] != inp[length-i-1])
  {
  flag = 1;
  break;
  }
}
if(flag==1)
{
cout<<"String is not a palindrome";
}
else {
    cout<<" string is a palindrome";
  }
}
```

23/8/23

Q2 Fibonacci Series :-

O/P Screen :- Enter the no. of terms - 6

→ 0 1 1 2 3 5

⇒

```
#include <iostream>

int main()
  {
  int no. of terms;
   int t1 = 0 ; int t2 = 1;
  cout << "Enter no. of terms";
   cin >> no. of terms;
  cout << t1 << t2;
  For( i=0 ; i< no. of terms-2; i++)
    {
  next term = t1 + t2;
  cout << "next term";
   t1 = t2;
    t2 = next term;
   }
}
```

Function

```
void fib ( int);   // declaration
 main ()
 {
 int no. of terms;
cout<< "Enter no. of terms";
 cin>> no. of terms;
```

```
fib(no.of terms);  // calling
                   → Actual Arguments
}
void

void fib(int no.)  // definition
                   → Formal Arguments
  {
int t1=0; int t2=1;
cout<<t1<<t2;
for(i=0; i<no.of terms-2; i++)
 {
next term = t1+t2;
cout<<"next term";
t1=t2;
 t2=next term;
 }
}
```

Using Recursion

```
main()
{
int no.of terms=5;
for(i=0; i<no.of terms; i++)
 {
  fib(i);
}

  fib(int n)
   {
if(n==0||n==1)
 { return 1;
 else
return(fib(n-1)+fib(n-2));
   }
```

Q11 WAP to implement call by value, call by Address and call by reference.

Functions :-

1) Call by Value
2) Call by Address
3) Call by Reference.

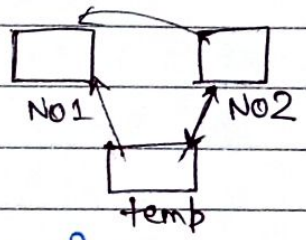
1) Call by Value :- [a=10, b=20 (Swap 2 No.)]

[• When we call a function by passing a value.]

Q11.

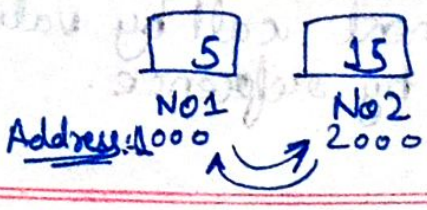
```
void swap(int);
main()
{
~~cout~~ int no1 , no2;

cout<<"Enter 2 No ";
cin >> no1 >> no2;
cout<<"Original Values are<<no1<<no2;
 swap ( no1 , no2); //Actual Arguments
cout <<"Numbers are" << no1<<no2; {
int swap(int no1; int no2) //Formal Arguments
  {
   temp = no1;
   no1 = no2;
    no2 = temp;
cout<<"Numbers are"<<no1<<no2;
  }
```



2) Call by Address :- [When we call a function by passing an address].

```
main()
 {
    int no1; no2; int p1, p2;
Cout <<"Enter 2 No ";
 Cin >>no1>>no2;
```




```
cout << "Original No are: " << no1 << no2;

swap (&no1, &no2); // Actual Arguments

cout << " After swap No. are: " << no1 << no2;
}
```

&no1 → *p1 , &no2 → *p2

Return Type: `void swap( int *no1, int *no2)` // Formal Arguments

```
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
    cout << *p1 << *p2;
}
```

Value at Address p1 ← `*p1 = *p2;`

- Declare a Pointer variable
  - `int *p;` (`int x;`)
  - `p = &x;`
- It holds the address of the
- Special variable
- In this '*' is used to declare a variable.

* Address से Value को fetch करना है

Dereferencing of a pointer Variable

`*p1` ↓ Value at p1

Value at &no1

Value at 1000

int& no 1 = It is a reference variable of no 1.

24/8/23

[Combination of Call by value & Call by Address]

3) Call by Reference :-

```
void swap(int&, int&);
main ()
{
  int no1, no2;

cout<<"Enter 2 No";
cin>>no1>>no2;
cout<<"Actual No are"<<no1<<no2;
 swap(no1, no2);
cout<<"Number are"<<no1<<no2;

}

void swap(int&a, int&b)
  {
  int temp;
  temp = a;
   a=b;
   b =temp;
cout<< a<< b;

 }
```

Reference Variable

↳ It is an alias name.

While declaring any variable

```
int a;
int &x = a;
// x is a reference to a.
// x is a reference variable.
// x is an alias name for a.
```

Call by Value	Call by Address	Call by Reference
→ Calling the function by passing value.	→ Calling the function by passing address.	→ Calling the function by passing value.
→ Changes are not permanent.	→ Changes are permanent.	→ Changes are permanent.
→ Actual Value = Value.	→ Actual value = Address.	→ Actual value = value.
→ Formal parameter = value.	→ Formal parameter = Pointer.	→ Formal parameter = variable.

* Reference have power of both call by value as well as call by address.

* Actual value passing are values but the formal arguments are the reference variable (their address.)

Default Arguments -

```
int sum(int, int);
int sum(int, int, int);
main()
{
  int no1, no2;
  int ans, ans1;
cout<<"Enter 2 No";
cin>>{no1}>>no2;
```

```
ans = sum( no1, no2);
 cout << "Sum is" << ans;
 cin >> no  cout << "Enter third no";
    Cin >> no3;
ans1 = Sum1( no1, no2, no3);
  cout << "Sum is" << ans1;
   }
int Sum( int no1, int no2)
   {
   int no3;
 no3 = no1 + no2;
   return no3;
   }
int sum1( int no1, int no2, int no3)
   {
   int no4;
  no4 = no1 + no2 + no3;
   return no4;
  }
```

Problem :- For doing same task, here is the example of addition.
We need to define multiple function to perform the same task as no. of arguments vary.

#. To solve this problem we will use Default Arguments.

* Default Arguments are those arguments that are automatically assigned by compiler. But If you are providing value then default value will be overwrite

Code :-

```
int main ()
  {

cout << "Sum(10,20)";
cout << "Sum(10,20,30)";
cout << Sum(70,80,90,100);

}
int Sum( int x, int y, int z=0, int w=0);
  {
  return (x+y+z+w);
}
```

Default Argument :-

* Values are always copies from left to Right.

* You can only set trailing arguments (R → L) (पीछे वाला) as defaultt argumente.

```
Sum(int x, int y=0, int w, int z);
```
{ This is wrong way }

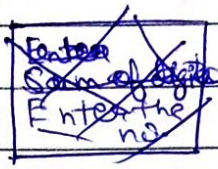
* Reduces the lines of code and increase the readability.

25/8/23

Q13. WAP to find sum of digits of a number.

⇒
```
#include <iostream>

int main()
{
  int num ; int sum=0;
int rem;
  cout << "Sum of digits" << endl;
  cin >> num ;
  cout << "Sum of digits" << num ;
  while(num >0)
  {
    rem = no % 10 ;
    sum = sum + rem ;
      num = num / 10;
  }
 cout << "Sum is" << sum;
}
```



Enter the no.
0 1 2 3 4
120

12 % 10 = 2

0+2 = 2

Q14. WAP to reverse a number.

⇒
```
int main()
{
int num; int rem; int rev=0;
cout << "Enter a number" << endl;
  cin >> num;
  while (num >0)
  {
  rem = no % 10;
    rev = (rev * 10) + rem;
      no = no / 10;
  }
cout << "Reverse no. is " << rev;
}
```

Q15 WAP to check no. is palindrome or not.

121 = 121
1456 = 6541

Q15 WAP to check no. is palindrome or not.

```
#include<iostream>
using namespace std;
int main ()
{
int n, temp, sum, rem;
cout << "Enter a no." <<endl;
cin >>n;
temp=n;
while (n>0)
{
rem = n%10;
sum = (sum*10)+ rem;
n = n/10;
}
if (temp==sum)
{
cout<<"No. is Palindrome";
}
else{
cout << "No. is not a Palindrome";
}
}
```

* Datatype of every real no. is "Double".

28/8/23

Function Overloading :-

Compile - Time polymorphism
↓ Multiple ↓ Forms

→ If any function definition is combines with function calling. /binds

→ Multiple function with same name & same task
↳ No. of arguments
↳ Types of arguments
differ in

* Function return type does not play any role in function overloading.

→ No. & type both can vary.

Q1 WAP to display value using F.D.

```
main()
{
f1(10);
f1(10,20);
f1(10,20,30);
f1(10,3.5);
}
int f1(int a)
{
return a;
}
void f1(int a, int b)
{
cout<<a<<b;
}
void f1(int a, int b, int c)
{
```

```
    cout<<a<<b<<c;
}
void f1(int a, double b)
{
    cout<<a<<b;
}
```

Q2. WAP to find :-

Area of Square = $(side)^2$ → int

Rectangle = $(L \times B)$ → int, int

Sphere = $\frac{4}{3}\pi r^2$ → float

Triangle = $\frac{1}{2} \times b \times h$ (b: int, h: float)

```
void area(int);
void area(int, int);
double ~~int~~ area(double);
double area(int, double);
```

⇒

```
int main()
{
    ~~int~~ int side; int base
    int length, Breadth;
double ~~float~~ r;
double ~~float~~ h;
    int ans;

int side(int a) ~~{~~ cout<<"Entere the side of square";
                      cin >> side;
    return a;
}
                  cout<<"Enter the length & Breadth
                      of rectangle";
                  cin >> l >> b;

    cout<<"Entere the radius of sphere";
    cin >> r;
```

```
                                    base
cout<<" Enter the breadth & height of triangle";
  cin >> b >> h;

  area (side);
  area( l, b);
   area( r);
   area(base, h);  ---> cout << Area is:<< ans;
}
                          5
void area ( int side)
  {
  cout << &*s;    = 25 :-O/P
  }
void area( int l , int b)
  {
  cout << l * b;
  }
void area( double r)
  {
  cout << r;
  }
```

or

```
double area (double r)
  {
  return ( 4/3 * 3.14 * radius);
  }
```

```
void area( int base , ~~int~~ double h)
  {
  cout << base * h;
  }
```

or or

```
~~double area(int~~
double area (int base, double h
  {
  return (0.5 * b * h);
  }
```

Test on 11 Sept (Unit-1)

3/8/23

Q. WAP to calculate the power of a number take number and power from the user and if power is not given then calculate the square of a number by default.

Q. WAP to add two numbers by taking two integers, two float, two double, one float one integer / one integer, one float.

4/9/23

Classes and Objects:- [Class Variable]

1) Class is an user-defined data-type.
2) It is a template.
3) Blue-print of an object.

int, float, char → Inbuilt / Primitive

It is a keyword. Compiler ~~have~~ know the meaning.

```
Class Student          → Class Name
{
 public:               → keyword / Access specifier
 int rno;
 float marks;          // Data Members
 char grade;
 void display()        // Function
  {
 cout << "~~Roll~~ R No is " << rno;
 cout << " Marks" << marks;
 cout << "Grade" << grade;
 }
```

// Member Functions (Inside the function) values.

By Default class have 3 ~~function~~ modifiers
→ Public
→ Private
→ Protected

NOTE:- ~~So Class Members are declare~~ Either class members or data members or ~~data~~ member function they will declare outside the dot operator with the help of

```
main()
{
Student S1;
```

Object of student or class

Variable of type Class (Object)

Student

Class/ Userdefined data type

Dot Operator / Member Access Operator

```
S1.rno = 1;
S1.marks = 90.5;
S1.grade = A;
S1.display();
}
```

OR

```
cout << "RNo is:" << S1.rno;
cout << "marks is:" << S1.marks;
cout << "Grade is:" << S1.grades;
```

(When we do not display the function.)

Q. WAP to create a class employee to store the information Employee ID, EName, EDesignation, Eexperience by storing this information and display it inside the main.

⇒

```
Class Employee
{
Public: → Access Specifier
int EID;
char Ename[10];
char Edesignation[20];
float Eexperience;
};
main()
{
Employee E1;

E1.EID = 10;
strcpy(E1.Ename, "ABC");
```

Data Members

Declaration

* Members of class access direct itself.

strcpy(E1.Edesignation, "XYZ");
E1.Eexperience = 4.5;

cout << "EID is:" << E1.Eid;
cout << "Employee Name is:" << E1.Ename;
cout << "Employee Designation is:" << E1.Edesignation;
cout << "Employee Experience is:" << E1.Eexperience;

}

6/9/23

Private :- Either it can be Data Members or member function. They can't be access outside the class. Can be accessed only inside the class.

Public :- Data Members } can be accessed
Member Function } outside the class

Q. Class student
{
private: / public:
int rno;
char name[10]; } DM
float marks;

public:
void get()

NOTE:- Every copy has its own data members and jitne member ko call kiya wahi access hoenge.

```
{
cout << "Enter R No";
cin >> rno;
cout << "Enter name";
cin >> name;
cout << "Enter marks";
cin >> marks;
}
void display()
{
cout << "Roll No. is:" << rno;
cout << "Name is:" << name;
cout << "Marks is:" << marks;
}
};
main()
{
Student S1;
S1 get();
S1 display();
}
```

```
Student S1;
S1.rno=;
Strcpy(S1.name, "");
```

} Can't be accessed in private Compiler will not run this

Array of Objects multiple

⇒ Instead of making ~~more~~ number of objects. We can make array of objects.

```
⇒ main()
{
Student S[60]; // Array of objects
for (i=0; i<60; i++)
```

* :: = Scope Resolution Operator
(Agar class ke bahar define kiya toh yeh operator ka use karenge function set pehle)

```
{
  s[i].get();
  s[i].display();
}
```

How to define member function outside the class ?

⇒
```
class student
{
private:
int rno;
char name[10];
float marks;
public:
  void get();
  void display();
};
void student :: get()
{
cout<<"Enter RNo:";
cin>>rno;
cout<<"Enter name:";
cin>>name;
cout<<"Enter marks:";
cin>>marks;
}
void student :: display()
{
cout<<"RNO is :"<<rno;
```

DM

MF

```
cout << " Name is : " << name;
cout << " Marks is : " << name; marks;
}
main ()
{
Student S1;
  S1.get ();
  S1.display();
}
```

Q. WAP to create a class location input- states & capitals by declaring them in private section. Define member function. get data & display data. Define them outside the class.