NOTES OF C++ LANGUAGE

- **C++ Language:-** C++ is a high-level general-purpose programming language created by Danish computer scientist Bjarne Stroustrup and first released 1985 as an extension of the c programming language, or “c with Classes”.
- **features of C++.**
    1. Object-oriented programming (OOP)
    2. Platform independence:
    3. Rich library support:
    4. Performance:
    5. Memory management:
    6. Generic programming:
    7. Exception handling:
    8. Multithreading:
    9. Dynamic Memory Allocation:
    10. Case-Sensitive:
    11. Simple And User Friendly:
    12. Structured Programming Language:

DATA TYPES

1. **Primitive(pre-defined):** Primitive Data Types: These data types are built-in or predefined data types and can be used directly by the user to declare variables.

   Data types.

   1) Char
   2) Float
   3) Double
   4) Boolean

2. **Non-primitive:** Non-primitive data types are those data types that are derived from primitive data types.
   - Data types(user-define)
   - Line is flow of data in one direction.
   - Array :- array is a collection of same data types.
   - Structure :- structure is a collection of different data types.
   - Union :- structure of good memory management.
   - Stack (LIFO) :- Stack(last in first out) order or approach in which the operations are performed.
   - Queue (FIFO) :- He queue data structure follows the FIFO(First in first out) principle where elements that are added first will be removed first.

3. **Non- linear follow of data in multiple direction.**
   - Tree :- A tree is a type of data structure representing hierarchical data.
   - Graph :- A graph is a type of non-linear data structure that is that used to store data in the form of nodes and edges.

   #include<iostream> :-standard input-output stream.

   Using namespace std; :- He std is a short form of standard, the std namespace contains the built-in classes and declared functions.

Keyword

Asm	**Const**	**Friend**	**Return**	**Dynamic cast**	**Inline**	**Wchart**	**Unsigned**
Auto	**Else**	**New**	**Short**	**This**	**Int**	**Signed**	**Using**
Bool	**Explicit**	**Operator**	**Const cast**	**Throw**	**Long**	**Size of**	**Virtual**
Break	**Export**	**Private**	**Continue**	**True**	**Mutable**	**Static**	**Void**
Case	**Extern**	**Protected**	**Default**	**Try**	**Name space**	**Static cast**	**While**
Catch	**False**	**Public**	**Delete**	**Volatite**	**Type def**	**Struct**	
Char	**Float**	**Register**	**Do**	**Goto**	**Type name**	**Switch**	
Class	**For**	**Reinterpret cast**	**Double**	**If**	**Union**	**Template**	

Comments

Line Lines that are not part of program.

Single Line

//

Multiple

/*

*/

Constants

Values that don't change(fixed)

Types

Integer Constants

1, 2, 3, 0 , -1, -2

Real Constants

1.0, 2.0, 3.14, -24

Character Constants

'a' , 'b' , 'A' , '#' , '

Variables

Variables:- Variable is the name of a memory location which stores some data.

variables

1. Variables are case sensitive.
   Upper case ‘A’ and lower case ‘a’ letter.
2. 1st character is alphabet or '_' .
   Only _age or a character.
3. no comma/blank space.
   Only use for underscore.
4. No other symbol other than '.

Variables:-
Int, char, float and double ko kahate hain.
#include<iostream>
Using namespace std;
Int main()
{
Int a=20;
Cout<<” the value of a is”<<a;
Getch();
}
Cout<< donon se cout and cin ko iostream class ka antargat define kiya gaya hai isliye ham program mein inki header file (iostream.h) ko include karte hain. main statement mein massage ko (“)double quotes mian likha jata hain.
Cin>> mein statement ko kisi statement ki value read karne ke liya double quotes ka use nahin hota hain.
#include<iostream> main header file main hote hain.
Cout<<:- it is use of take display output.
Cin>>:- it is use of take display input.

Operators.

- **Operators:** Operators are the symbols that operators on a value a variable. Operators are used to perform operations on variables and value.
- **Arithmetic operators:** Arithmetic operators are symbols that are used to perform mathematical operations.

Operators	meaning.
+	Addition.
_	Subtraction.
*	Multiplication.
/	Division.
%	Modules division.

- **Relational operators/ Comparison operators:** Comparison operators are used to compare two values:

Operators	meaning.
<	is less than.
<=	is less than or equal.
>	is greater than.
>=	is greater than or equal.
==	is equal.
!=	is not equal.

- **Logical operators: Logical operators are used to combine conditional statements:**

Operators	meaning.
&&	AND
||	OR
!	NOT

- **Bitwise operators:** Bitwise operators are used to compare (binary) numbers:

&	AND
|	OR
^	XOR
~	NOT
<<	Zero fill left shift
>>	Signed right shift

Arithmetic operators

```
// using third variable swapping
#include<iostream>
using namespace std;
int main()
{
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enten the number b";
        cin>>b;
        c=a+b;
        a=c-a;
        b=c-b;
        cout<<"\n after swapping";
        cout<<"\n a number"<<a;
        cout<<"\n b number"<<b;
        getch();
}
```

Relational operators

```
// three smallest number;
#include<iostream>
using namespace std;
int main()
{
        int a,b,c;
        cout<<"\n enter the numbera";
        cin>>a;
        cout<<"\n enter the numberb";
        cin>>b;
        cout<<"\n enter the numberc";
        cin>>c;
        if(a<b&&a<c)
        cout<<"\n a is smallest number";
        if(b<a&&b<c)
        cout<<"\n b is smallest number";
        if(c<a&&c<b)
        cout<<"\n c is smallest number";
        getch();
}
```

Logical operators

```
// even and odd.
#include<iostream>
using namespace std;
int main()
{
        int number,even=0,odd=0,i;
        cout<<"\n enter the number";
        for(i=1;i<=10;i++)
        {
        cin>>number;
        if(i%2==0)
        even=even+number;
        else
        odd=odd+number;
        }
        cout<<"\n even number"<<even;
        cout<<"\n odd number"<<odd;
        getch();
}
```

Bitwise operators

```
// Swapping with using third variable.
#include<iostream>
using namespace std;
int main()
{
        int a,b;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enten the number b";
        cin>>b;
        a=a^b;
        b=a^b;
        a=a^b;
        cout<<"\n after swapping";
        cout<<"\n a number"<<a;
        cout<<"\n b number"<<b;
        getch();
}
```

Loop

- **Loop :-** The purpose of the loop is the except some code number of time.

  There are three types of loop.

  1.for loop.
  2.while loop.
  3.do while loop.

  For loop.

  For loop(initialisation; condition; updation) {

  // do something}
  For loop is a entry control loop.

  While loop.
  While(condition) {
  // do something}
  While loop is a entry control loop.
  While loop is a pre- test loop.
  The condition is specified begin of the loop.

  Do while loop.
  Do { // do something }
  While(condition);
  Do while loop is a exit control loop.
  Do while loop is a past- test loop.
  The condition is specified end of the loop.

Loop.

```
// loop in even and odd;
#include<iostream>
using namespace std;
int main()
{
        int number,even=0,odd=0,i;
        cout<<"\n enter the number";
        for(i=1;i<=10;i++)
        {
        cin>>number;
        if(i%2==0)
        even=even+number;
        else
        odd=odd+number;
        }
        cout<<"\n even number"<<even;
        cout<<"\n odd number"<<odd;
        getch();
}
```

Nested loop.

1.Pattern;

```
#include<iostream>
using namespace std;
int main()   {
        int i,j;
        for(i=1;i<=10;i++)    {
        for(j=1;j<=i;j++)    {
        cout<<"*";
        }
        cout<<"\n";   }
        Getch();
}
```

2. pattern;

```
#include<iostream>
using namespace std;
int main()    {
        int i,j;
        for(i=1;i<=10;i++)    {
        for(j=1;j<=10;j++)   {
        cout<<"*";      }
        cout<<"\n";     }
        getch();
}
```

While loop.

```
// table;
#include<iostream>
using namespace std;
int main()
{
        int table,r,i=1;
        cout<<"\n enter the table";
        cin>>table;
        while(i<=10)
        {
        r=table*i;
        cout<<"\n"<<r;
        i++;
        }
        Getch();
}
```

Do while loop

```
// table;
#include<iostream>
using namespace std;
int main()
{
        int table,i=1,r,number;
        cout<<"\n enter the factorial number";
        cin>>table;
        do
        {
        r=table*i;
        cout<<"\n"<<r;
        i++;
        }
        while(i<=10);
        getch();
}
```

Conditional statement

- **Switch :-** The switch case statement allow variable to be tested again in the list value each value is called case. The switch statement in checked in every value in case.
- If else :- The if-else statement is used to execute both the true part and the false part of a given condition.

Syntax of switch

```
Switch(number) {
Case 1: // do something;
Break;
Case 2: // do something;
Break;
Default: // do something;
}
```

Syntax of if-else

```
If(condition 1) {
// do something if true
}
If(condition 2) {
// do something if 1st is false & 2nd is true
}
```

```
Switch
//Calculator
#include<iostream>
using namespace std;
int main()
{
        int ch;
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        cout<<"\n enter the choice";
        cin>>ch;
        switch(ch)
        {
        case 1:
        c=a+b;
        cout<<"\n addition"<<c;
        break;
```

```
case 2:
c=a-b;
cout<<"\n subtraction"<<c;
break;
case 3:
c=a*b;
cout<<"\n multiplication"<<c;
break;
case 4:
c=a/b;
cout<<"\n division"<<c;
break;
default:
cout<<"\n you enter the wrong choice";
break;
}
Getch();
}
```

Pointer

- **Pointer :-** The pointer variable pointer is a variable that stores address of another variable.

  Syntax

  int age = 22;

  int *ptr = &age;

  int _age = *ptr;

  Declaring Pointers

  int *ptr;

  char *ptr;

  float *ptr;

  Pointer to Pointer

- A variable that stores the memory address of another pointer.

  Pointer to Pointer

  Syntax

  int **pptr;

  char **pptr;

  float **pptr;

  Pointers in Function Call

  Call by Value :- We pass value of variable as argument.

  Call by Reference :- We pass address of variable as argument.

  &= mpersonand.

  *= aestrick.

Pointer.

```
// simple interest
#include<iostream>
using namespace std;
int main()
{
        int principal,rate,time,*p,si;
        cout<<"\n enter the principal";
        cin>>principal;
        cout<<"\n enter the rate";
        cin>>rate;
        cout<<"\n enter the time";
        cin>>time;
        si=principal*rate*time/100;
        p=&si;
        cout<<"\n simple interst"<<si;
        cout<<"\n address"<<p;
        getch();
}
```

Pointer to Pointer.

```
// Swapping
#include<iostream>
using namespace std;
int main()
{
        int a,b,c,*p1;
        int **pp;
        cout<<"\n enter the numbera";
        cin>>a;
        cout<<"\n enter the numberb";
        cin>>b;
        c=a+b;
        a=c-a;
        b=c-b;
        p1=&c;
        pp=&p1;
        cout<<"\n after the swapping";
        cout<<"\n numbera"<<a;
        cout<<"\n numberb"<<b;
        cout<<"\n address"<<c;
        cout<<"\n pointer"<<pp;
        getch();    }
```

Strings

- **Strings:** A string is a variable that stores a sequence of letters or other characters.

A character array terminated by a '\0' (null character).

null character denotes string termination.
EXAMPLE

```
char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
```

Initialising Strings

```
char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
char name[ ] = "SHRADHA";
char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
char name[ ] = "SHRADHA";
```

String using Pointers.
Store string in memory & the assigned.
address is stored in the char pointer 'str'.

function of strings.

1. Strlwr :- string lower case into upper case.
2. Strupr :- string upper case into lower case.
3. Strrev :- reverse string function.
4. Strlen :- string for find length of letter.
5. Strcpy :- copy one string to another string.
6. Strcat :- merge two string.
7. Strcmp :- compare two string.

String enter function.

Printf ,cout :- puts().

Scanf ,cin :- gets().

String.

```
// palindrome.
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
        char str[10];
        int lenth,i,r,s=0,c;
        cout<<"\n enter the string";
        cin>>str;
        c=lenth;
        for(i=0;i<lenth;i++)
        {
        r=lenth&10;
        s=r+(s*10);
        lenth=lenth/10;
        }
        if(c==s)
        cout<<"\n it is a palindome"<<str;
        else
        cout<<"\n it is not a palindome"<<str;
}
```

String.

```
#include <iostream>
#include <string>
using namespace std;
bool isPalindrome(string str) {
  int n = str.length();
  for (int i = 0; i < n / 2; i++) {
    if (str[i] != str[n - i - 1]) {
      return false;
    }
  }
  return true;
}
int main() {
  string str;
  cout << "Enter a string: ";
  cin >> str;
  if (isPalindrome(str)) {
    cout << str << " is a palindrome." << endl;
  } else {
    cout << str << " is not a palindrome." << endl;
  }
  return 0;
}
```

Function

- **Function :-** A function is the blocks of statements which can call by another function is calling by function block of code that performs particular task Functions.

  There are main four type of function.

  1. No argument and no return value.
  2. No argument and a return value.
  3. Argument and a return value.
  4. Argument and a return value.
- **Special function.**
- **Call by value:-** The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.
- **Call by reference :-** passing by reference refers to a method of passing the address of an argument in the calling function to a corresponding parameter in the called function. The corresponding parameter in the called function must be declared as a pointer type.
- **Recursion:-** Recursion is a method in of function which is calls itself directly or indirectly until a suitable condition is met.
- **Function overloading:-** function overloading in defined as the process two or more function with the same name, but different types of parameters called function overloading.
- **Operator overloading: -** To assign more than one operation on a same operator know as operator overloading to achieve operator overloading, we have to write a special function known as operator ().
- **There are two types of operator overloading.**
- **Unary operator overloading: -** A operator which contain only one open and is called unary operator overloading.
- **Binary operator overloading: -** A operator which contain two operands is called binary operator overloading.

1.No argument and no return value.

```
#include<iostream>
using namespace std;
#include<conio.h>
int sum();   //function declaration.
int main()
{
        sum();   //function calling.
        getch();
}
        int sum()  //function difination.
        {
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        c=a+b;
        cout<<"\n addition"<<c;
        }
```

2.argument and a return value.

```
#include<iostream>
using namespace std;
#include<conio.h>
int swap();
int main()
{
        int s;
        s=swap();
        getch();
}
        int swap()
        {
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        c=a+b;
        a=c-a;
        b=c-b;
        cout<<"\n after swapping";
        cout<<"\n a number"<<a;
        cout<<"\n b number"<<b;
        return c;    }
```

3.argument and no return value.

```
#include<iostream>
using namespace std;
#include<conio.h>
int number(int a,int b, int c);
int main()        {
        int g,x,y,z;
        number(x,y,z);
        getch();  }
        int number(int a,int b, int c)        {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        cout<<"\n enter the number c";
        cin>>c;
        if(a>b&&a>c)
        cout<<"\n a is greater";
          else
        if(b>a&&b>c)
        cout<<"\n b is greater";
        else
        if(c>a&&c>b)
        cout<<"\n c is greater";
        return c;     }
```

4.argument and a return value.

```
#include<iostream>
using namespace std;
#include<conio.h>
int swap(int a, int b);
int main()      {
        int s,x,y;
        s=swap(x,y);
        getch();        }
        int swap(int a, int b)     {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        a=a^b;
        b=a^b;
        a=a^b;
        cout<<"\n after swapping";
        cout<<"\n a number"<<a;
        cout<<"\n b number"<<b;
        return a;
        }
```

5.call by value.

```
#include<iostream>
using namespace std;
#include<conio.h>
int number(int a,int b, int c);
int main()      {
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        cout<<"\n enter the number c";
        cin>>c;
        number(a,b,c);
        getch();     }
        int number(int a,int b, int c)     {
        if(a>b&&a>c)
        cout<<"\n a is greater";
        else
        if(b>a&&b>c)
        cout<<"\n b is greater";
        else
        if(c>a&&c>b)
        cout<<"\n c is greater";
        return c;    }
```

6.call by value reference.

```
#include<iostream>
using namespace std;
#include<conio.h>
int number(int *a,int *b, int *c);
int main()      {
        int a,b,c;
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        cout<<"\n enter the number c";
        cin>>c;
        number(&a,&b,&c);
        getch();      }
        int number(int *a,int *b, int *c)    {
        if(*a>*b&&*a>*c)
        cout<<"\n a is greater";
        else
        if(*b>*a&&*b>*c)
        cout<<"\n b is greater";
        else
        if(*c>*a&&*c>*b)
        cout<<"\n c is greater";
        return 0;     }
```

Recursion function.

```
#include<iostream>
using namespace std;
int sum(int a,int b)
{
        if(b!=1)
        return sum(a,b-1);
        else
        return a;
}
int main()
{
        int a,b,c;
        cout<<"\n enter the numbera";
        cin>>a;
        cout<<"\n enter the numberb";
        cin>>b;
        c=a+b;
        cout<<"\n the sum of two
number"<<c;
}
```

Function overloading.

```
#include<iostream>
using namespace std;
int natural()
{
        int a,b,add;
        cout<<"\n enter the numbera";
        cin>>a;
        cout<<"\n enter the numberb";
        cin>>b;
        add=a+b;
        cout<<"\n the sum of two
number"<<add;
}
int natural(double a,double b)
{
        double sub;
        sub=a-b;
        cout<<"\n the sub of two
number"<<sub;
}
int main()
{
natural();
natural(3,2);
return 0;
}
```

Inline function.

```
#include<iostream>
using namespace std;
inline cube(int b)
{
   return (0);
}
int main()
{
   int  b,i,cube;
   cout<<"\n enter the cube";
   cin>>b;
   for(i=1;i<b;i++)
   {
   cube=b*b*b;
        }
   cout<<"\n Cube value for is:"<<cube;
}
```

Friend function.

```
#include<iostream>
using namespace std;
class A
{
        int a,b;
        public:
        void setdata()
        {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        }
        friend void add(A obj);
};
void add(A obj)
{
        int sum;
        sum=obj.a+obj.b;
        cout<<"\n sum"<<sum;    }
int main()     {
        A c;
        c.setdata();
        add(c);
        return 0;      }
```

Array

- **Array:-** The array is a collection of same data types.

  Initialization of array

  Int marks [ ] ={ 97,98,89};

  Int marks [ ] = { 97,98,89};

  Input & Output

  Cin>>marks[0];

  Cout<<marks[0];

  Pointer Arithmetic

  Pointer can be incremented & decremented.

  Int a=2;

  Int *ptr = &age;

  Ptr++;

  Array is a Pointer

  int *ptr = &arr[0];

  int *ptr = arr;

  Traverse an Array

  int aadhar[10];

  int *ptr = &aadhar[0];

Array minimum elements.

```
#include<iostream>
using namespace std;
int main()
{
        int a[5],i,min;
        cout<<"\n enter the array elements";
        for(i=0;i<5;i++)
        {
        cin>>a[i];
        }
        min=a[0];
        for(i=0;i<5;i++)
        {
        if(a[i]<min)
        {
        min=a[i];
        }
        }
        cout<<"\n smallest elements"<<min;
}
```

Array maxmimum elements.

```
#include<iostream>
using namespace std;
int main()
{
        int a[5],i,max;
        cout<<"\n enter the elements";
        for(i=0;i<5;i++)
        {
        cin>>a[i];
        }
        max=a[0];
        for(i=0;i<5;i++)
        {
        if(a[i]>max)
        {
        max=a[i];
        }
        }
        cout<<"\n largest elements"<<max;
}
```

Array inserting elements.

```
#include<iostream>
using namespace std;
int main()
{
        int arr[5],i,pos,number;
        cout<<"\n enter the array elements";
        for(i=0;i<5;i++)
        {
        cin>>arr[i];
        }
        cout<<"\n enter the inserting elements position";
        cin>>pos;
        cout<<"\n enter the inserting elements";
        cin>>number;
        for(i=5-1;i>=pos-1;i--)
   arr[i+1]=arr[i];
   arr[pos-1]=number;
   cout<<"final array after inserting the value is";
   for(i=0;i<=5;i++)
   cout<<"\n"<<arr[i];
}
```

Array deleting elements.

```
#include<iostream>
using namespace std;
int main()
{
   int arr[100],position,i,number;
   cout<<"\n Enter number of elements in array";
   cin>>number;
        cout<<"\n enter the array elements";
   for(i=0;i<number;i++)
   cin>>arr[i];
   cout<<"\n Enter the location where you wish to delete elements";
   cin>>position;
   if ( position>=number+1)
   cout<<"\n Deletion not possible";
   else
   {
      for (i=position-1;i<number-1;i++)
      arr[i]=arr[i+1];
      printf("\n Resultant array is");
      for(i=0;i<number-1;i++)
      cout<<"\n"<<arr[i];
   }
   return 0;
}
```

Array marge in the elements.

```
#include<iostream>
using namespace std;
int main()
{
   int a[5],b[5],c[10],i,j;
   cout<<"\n enter the array elements a";
   for(i=0;i<5;i++)
   {
   cin>>a[i];
   c[i]=a[i];
        }
        j=i;
        cout<<"\n enter the array elements b";
        for(i=0;i<5;i++)
        {
        cin>>b[i];
        c[j]=b[i];
        j++;
        }
        cout<<"\nThe New Array (Merged
Array):\n";
   for(i=0; i<j; i++)
   cout<<"\n"<<c[i]<<" ";

}
```

Array shorting elements.

```
#include<iostream>
using namespace std;
int main()
{
        int a[5],i,j,c;
        cout<<"\n enter the array elements";
        for(i=0;i<5;i++)      {
        cin>>a[i];            }
        for(i=0;i<5;i++)      {
        for(j=i+1;j<5;j++)    {
        if(a[i]>a[j])   {
        c=a[i];
        a[i]=a[j];
        a[j]=c;
        }
        }
        }
        cout<<"\n the ascending order array
elements";
        for(i=0;i<5;i++)
        {
        cout<<"\n"<<a[i];
        }
}
```

Structure

- **Structure:-** A structure is a collection of different data types.

  Syntax.

  Struct student {

  Char name[100];

  Int roll;

  }

  Initializing Structures

  struct student s1 = { "shradha", 1664, 7.9};

  struct student s2 = { "rajat", 1552, 8.3};

  struct student s3 = { 0 };

  Pointers to Structures

  struct student *ptr;

  ptr =&s1;

My number is 1212 767-890

Area, code, exchange 415 555 1212

```
#include<iostream>
using namespace std;
struct   phone
{
        int code;
        int area;
        int exchange;
};
int main()
{
phone         p1,p2;
        p1.code=415;
        p1.area=555;
        p1.exchange=1212;
        cout<<"\n enter the code:";
        cin>>p2.code;
        cout<<"\n enter the area:";
        cin>>p2.area;
        cout<<"\n enter the exchange:";
        cin>>p2.exchange;
        cout<<"\n your area code
exchange:"<<p1.code<<p1.area<<p1.exchan
ge;
cout<<"\n my number is:" <<p2.code
<<p2.area<<p2.exchange;
}
```

// enter the two employee name and age perday salary and working days and calculat total salary.

```
#include<iostream>
using namespace std;
struct employee
{
        char na[10];
        int age;
        float sa,wd,t;
};
        int main()
        {
        int i;
        employee e[2];
        for(i=0;i<2;i++)
        {
        cout<<"\n enter the name"<<1+i;
        cin>>e[i].na;
        cout<<"\n enter the age";
        cin>>e[i].age;
        cout<<"\n enter the perday salary";
        cin>>e[i].sa;
        cout<<"\n enter the total workin
days";
        cin>>e[i].wd;
        e[i].t=e[i].sa*e[i].wd;
        cout<<"\n total salary"<<e[i].t;
        }
}
```

ENUM Datatype.

- **ENUM Datatype.**
- Enumeration, is a user-defined data type that enables you to create a new data type that has a fixed range of possible values, and the variable can select one value from the set of values.
- **Code C++.**

```
#include<iostream>
using namespace std;
int main(){
        enum gender{
                male,female,others
        };
        cout<<male<<endl;
        gender str = others;
        cout<<str;
        return 0;
}
```

Class and object.

- **Introduction of object-oriented programming.**
  Object oriented programming is a based-on class and object. Which is used to decomposed into big problem into small problem also help to make code re-useability.
- **Templates:-**
- **Class=** class is a member variable and member function. Class is created objects.
- **Object=** object is a instance of class. Object is a real word entity such as pen and laptop.
- **Access specifier.**
  1.private :- members are accessible from outside the class.
  2.protected:- members cannot be accessed from outside the class however they can be accessed in inherited classes.
  3.public:- members cannot be accessed (or viewed) from outside the class.
- **The Features of OOPS.**
- **Inheritance:** property of one class can be in inherited into the other class.
- **Encapsulation:** Encapsulation, is to make sure that "sensitive" data is hidden from users.
- **Data Abstraction:** Data Abstraction is a programming technique that focuses on providing only the essential information about the data to the outside world, while hiding the background details or implementation.
- **Polymorphism:** Polymorphism is one of the core concepts of object-oriented programming (OOP). It is the ability of an object to take on many forms. In C++, polymorphism is achieved through function overloading, operator overloading, and virtual functions.
- **Dynamic Binding:** Dynamic binding is the process of binding a function call to a specific function at runtime. This allows the programmer to write code that is more flexible and reusable.
- **Exception Handling:** Exception handling is the process of handling errors that occur during program execution. This allows the programmer to write code that is more robust and reliable.

Syntax.

```
#include<iostream>
using namespace std;
class circle
{
        public:
        float area,radius;
        void areaa()
        {
        cout<<"\n enter the radius";
        cin>>radius;
        area=3.14*radius*radius;
        cout<<"\n area of circle"<<area;
}
};
int main()
{
        circle obj;
        obj.areaa();
}
```

Access specifier.

1.private, 2. Protected, public.

1.private.

```
#include<iostream>
using namespace std;
class a         {
        private:
        int area,l,b;
        void fun()    {
        cout<<"\n enter the length";
        cin>>l;
        cout<<"\n enter the breadth";
        cin>>b;
        area=l*b;
        cout<<"\n area of Area of a
Rectangle"<<area; }
        void fun(int a,int b)         {
        int c;  //a=10,b=20;
        c=a+b; //c=0,a=10+b=20=30,c=30;
        a=c-a; //a=30-10=20;
        b=c-b;//b=30-10=20;
        cout<<"\n after swapping";
        cout<<"\n number a"<<a;
        cout<<"\n number b"<<b;         }        };
int main()    {
        a obj;
        obj.fun();
        obj.fun(10,20);      }
```

Public:

```
#include<iostream>
using namespace std;
class A
{
        float  radius,area;
        public:
        void setdata()
        {
        cout<<"\n enter the radius";
        cin>>radius;
        area=3.14*radius*radius;
        cout<<"\n area of circle"<<radius;
        }
        void getdata()
        {
        cout<<radius<<area;
        }
};
int main()
{
        A obj;
        obj.setdata();
        obj.getdata();
}
```

Protected:

```
#include<iostream>
using  namespace std;
class Base{
        protected:
        int a,b;
        public:
        void show(){
        a = 400; b = 600;
           cout<<"\n"<<a<<"\n"<<b;
        }
};
class Derive:public Base{
        public:
        int a,b;
        void dis(){
        a = 400; b = 600;
        cout<<"\n"<<a<<"\n"<<b;
        }
};
int main(){
        Derive obj;
        obj.dis();
        obj.show();

}
```

Inheritance

- **Inheritance:** property of one class can be in inherited into the other class.
- **There are 5 types of inheritance**
- **Single inheritance.**
- Single inheritance enables a derived class to inherit properties from a single parent class.
- **Code C++.**
- ```
  #include<iostream>
  using namespace std;
  class a
  {
          int a,b,sum;
          public:
          void setdata()
          {
          cout<<"\n enter the number a";
          cin>>a;
          cout<<"\n enter the number b";
          cin>>b;
          sum=a+b;
          cout<<"\n addition"<<sum;
          }
  };
  class b:public a
          {
          };
  int main()
  {
          a obj;
          obj.setdata();
  }
  ```
  ```

- **Multi-level inheritance.**
- In this inheritance we have one parent class and multiple child class.
- **Code C++.**

```
#include<iostream>
using namespace std;
class a
{
        int a,b,sum;
        public:
        void setdata()
        {
        cout<<"\n THE NUMBER OF ADDITION";
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        sum=a+b;
        cout<<"\n addition"<<sum;
        }
};
class b:public a
{
        public:
        int reverse,number;
        void putdata()
        {
        cout<<"\n THE REVERSE NUMBER";
        cout<<"\n enter the number";
        cin>>number;
        while(number>0)
        {
        reverse=number%10;
        cout<<reverse;
        number=number/10;
        }
        }
};
```

```
class c:public b
{
        public:
        int intTypes;
        char charTypes;
        float floatTypes;
        double doubleTypes;
        void getdata()
        {
        cout<<"\n THE SIZE OF DATA TYPES";
        cout<<"\n size of int"<<sizeof(int);
        cout<<"\n size of char"<<sizeof(char);
        cout<<"\n size of float"<<sizeof(float);
        cout<<"\n size of double"<<sizeof(double);
        }
};
class d:public c
{
        public:
};
int main()
{
        d d;
        d.setdata();
        d.putdata();
        d.getdata();
}
```

- **Multiple inheritance.**
- Inheritance which contains more parent classes and only one child class is called multiple inheritance.
- **Code C++.**
- ```
  #include<iostream>
  using namespace std;
  class a
  {
          int a,b,sum;
          public:
          void setdata()
          {
          cout<<"\n THE NUMBER OF ADDITION";
          cout<<"\n enter the number a";
          cin>>a;
          cout<<"\n enter the number b";
          cin>>b;
          sum=a+b;
          cout<<"\n addition"<<sum;
          }
  };
  class b:public a
  {
          public:
          int reverse,number;
          void putdata()
          {
          cout<<"\n THE REVERSE NUMBER";
          cout<<"\n enter the number";
          cin>>number;
          while(number>0)
          {
          reverse=number%10;
          cout<<reverse;
          number=number/10;
          }
          }
  ```
  ```

```
};
class c:public b
{
        public:
        int intTypes;
        char charTypes;
        float floatTypes;
        double doubleTypes;
        void getdata()
        {
        cout<<"\n THE SIZE OF DATA TYPES";
        cout<<"\n size of int"<<sizeof(int);
        cout<<"\n size of char"<<sizeof(char);
        cout<<"\n size of float"<<sizeof(float);
        cout<<"\n size of double"<<sizeof(double);
        }
};
class d:public c
{
        public:
};
int main()
{
        d d;
        d.setdata();
        d.putdata();
        d.getdata();
}
```

- **Hierarchical Inheritance.**
- A class which contains only one base and multiple derive class but each derive class can access base class is called hierarchical inheritance.
- **Code C++.**

```
#include<iostream>
using namespace std;
class base
{
        private:
        int a,b,sum,sub;
        public:
        void setdata()
        {
        cout<<"\n THE NUMBER OF ADDITION";
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        sum = a + b;
        cout<<"\n addition"<<sum<<endl;
        }
        void show(){
        cout<<"\n THE NUMBER OF SUBTRACTION";
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        sub = a - b;
        cout<<"\n Subtraction"<<sub<<endl;
        }
};
```

```
class derive1:public base                    {
        private:
        int reverse,number;
        public:
        void putdata()
        {
        cout<<"\n THE REVERSE NUMBER";
        cout<<"\n enter the number";
        cin>>number;
        while(number>0)
        {
        reverse=number%10;
        cout<<reverse;
        number=number/10;
        }               }                    };
class derive2:public base                    {
        public:
        int intTypes;
        char charTypes;
        float floatTypes;
        double doubleTypes;
        void getdata()
        {
        cout<<"\n THE SIZE OF DATA TYPES";
        cout<<"\n size of int"<<sizeof(int);
        cout<<"\n size of char"<<sizeof(char);
        cout<<"\n size of float"<<sizeof(float);
        cout<<"\n size of double"<<sizeof(double);
        }                       };
int main()
{
        derive1 obj1;
        derive2 obj2;
        obj1.setdata();
        obj2.show();
        obj1.putdata();
        obj2.getdata();                      }
```

- **Hybrid Inheritance.**
- It is the combination of more than one type of inheritance is called hybrid inheritance.
- **Code C++.**

```
#include<iostream>
using namespace std;
class A
{
        private:
        int a,b,sum,sub;
        public:
        void setdata()
        {
        cout<<"\n THE NUMBER OF ADDITION";
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        sum = a + b;
        cout<<"\n addition"<<sum<<endl;
        }
};
class B:virtual public A
{
        private:
        int reverse,number;
        public:
        void putdata()
        {
        cout<<"\n THE REVERSE NUMBER";
        cout<<"\n enter the number";
        cin>>number;
        while(number>0)
        {
        reverse=number%10;
        cout<<reverse;
        number=number/10;
```

```
        }
        }
};
class C:virtual public A
{
        int a,b,sub;
        void show(){
        cout<<"\n THE NUMBER OF SUBTRACTION";
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        sub = a - b;
        cout<<"\n Subtraction"<<sub<<endl;
        }
};
class D:public B, public C
{
};
int main()
{
        A obj1; B obj2; C obj3; D obj4;
        obj1.setdata();
        obj2.setdata();
        obj3.setdata();
        obj4.setdata();
}
```

- **Constructor.**
- A constructor is a special type of member function that is called automatically when an object is created.
- **Code C++.**

```
#include<iostream>
using namespace std;
class construct
{
        public:
        int a,b,c;
        construct()
        {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        c = a + b;
        cout<<"The sum of two number"<<c;
        }
};
int main()
{
        construct c;//Object created
        cout<<"\n value of a"<<c.a;
        cout<<"\n value of b"<<c.b;
}
```

- **There are three types of constructors.**
- Default constructor.
- Parameterized constructor.
- Copy constructor.

- **Default constructor.**
- A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values.
- **Code C++.**

```
#include<iostream>
using namespace std;
class A
{
        int a,b,c;
        public:
        A()
        {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;
        c=a+b;
        cout<<"\n sum"<<c;
        }
};
int main()
{
        A obj;
}
```

- **Parameterized constructor.**
- A constructor that accepts or receive parameters is called Parameterized constructor.
- **Code C++.**

```
#include<iostream>
using namespace std;
class Rectangle
{
        private:
        double lenth;
        double breath;
        public:
        // parameterized constructor
        Rectangle(double len,double be)
        {
        lenth=len;
        breath=be;
        }
        double area()
        {
        return lenth*breath;
        }
};
int main()
{
        // create objects to call constructors
        Rectangle obj1(10,6);
        Rectangle obj2(13,8);
        cout <<"\n Area of Rectangle 1:"<< obj1.area();
        cout <<"\n Area of Rectangle 2:"<< obj2.area();
        return 0;
}
```

- **Copy constructor.**
- A constructor that is used to copy or initialize the value of one object into another objects is called copy constructor.
- **Code C++.**

```
#include <iostream>
using namespace std;
// class name: Rectangle
class Rectangle {
  private:
    double length;
    double breadth;
  public:
    // parameterized constructor
    Rectangle(double l, double b) {
    length = l;
    breadth = b;
    }
  // copy constructor with a Rectangle object as parameter copies data of
the obj parameter
    Rectangle(Rectangle &obj) {
    length = obj.length;
    breadth = obj.breadth;
    }
    double calculateArea() {
    return length * breadth;
    }
};
int main() {
  // create objects to call constructors
Rectangle obj1(10,6);
Rectangle obj2 = obj1;   //copy the content using object
//print areas of rectangles
  cout <<"\n Area of Rectangle 1:"<< obj1.calculateArea();
  cout <<"\n Area of Rectangle 2:"<< obj2.calculateArea();
  return 0;
}
```

- **Constructor overloading.**
- If a class contains multiple constructors where each type of constructor have different parameter constructor list is called Constructor overloading.
- **Code C++.**

```
// C++ program to showing Constructor overloading
#include <iostream>
using namespace std;
class constructor
{
public:
        float area;
        // Constructor with no parameters
        constructor()
        {
                area = 0; //Giving value 0 to variable area
        }
        // Constructor with two parameters a and b
        constructor(int a, int b)
        {
                area = a * b;
        }
        void display() //member function of the class
        {
                cout<< "Area: "<<area<< endl;
        }
};
int main()
{
        // Constructor Overloading
        // with two different constructors
        // of class name
        constructor obj;
        constructor obj2( 22, 40);
        obj.display();
        obj2.display();
        return 1;                               }
```

- **Destructor.**
- Destructor is a special member function that is executed automatically when an object is destroyed that has been created by the constructor.
- Destructor are used to de-allocate the memory that has be allocate for the object by the constructor.
- A destructor declaration should always begin with the tilde (~) symbol as show in the following.
- **Code C++.**

```
#include <iostream>
using namespace std;
class MyClass {
        char name, address;
        int age,salary;
    public:
        MyClass() {
            cout<<"\n Enter the name:";
            cin>>name;
            cout<<"\n Enter the age:";
            cin>>age;
            cout<<"\n Enter the salary:";
            cin>>salary;
            cout<<"\n Enter the address:";
            cin>>address;
    }
    ~MyClass() {
    cout << "Destructor called" << endl;
    cout<<"Details"<<endl;
    cout<<"\n name:"<<name<<"\n age:"<<age<<"\n
salary:"<<salary<<"\n address:"<<address<<endl;
    }
};
int main() {
    MyClass my_object;
  // The destructor will be called automatically when the object goes out
of scope.
  return 0;
}
```

- **SCOPE RESOLUTION OPERATOR(::).**
- The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.
- **Code C++.**

```
#include<iostream>
using namespace std;
class A{
        public:
                int number,even=0,odd=0,i;
                 void display(){
                        cout<<"\n enter the number";
                        for(i=1;i<=10;i++)                    {
                        cin>>number;
                        if(i%2==0)
                        even=even+number;
                        else
                        odd=odd+number;                    }
                        cout<<"\n even number"<<even;
                        cout<<"\n odd number"<<odd;
                }               };
class B:public A{
        public:
                int a,b,c;
                void display(){
                        cout<<"Enter the number a:";
                        cin>>a;
                        cout<<"Enter the number b:";
                        cin>>b;
                        c = a + b;
                        cout<<"The total sum of two number:"<<c;  }  };
int main(){
        B obj;
        obj.display();
        obj.A::display();
        //scope resolution operater.  }
```

- **Static keyword.**
- The static keyword can be used to declare variables and functions at global scope, namespace scope, and class scope.
- **Use of static.**
- **Static Data member.**
- Static data members are class members that are declared using static keywords.
- **Code C++.**

```
#include<iostream>
using namespace std;
class testing{
        static int count;
        int num;
        public:
                void setData(int x){
                        num = x;
                        count++;
                }
                void getStaticValue(){
                        cout<<"count"<<count<<endl;
                }
};
int testing::count;
int main(){
        testing obj1, obj2, obj3;
        obj1.getStaticValue();
        obj2.getStaticValue();
        obj3.getStaticValue();
        obj1.setData(50);
        obj2.setData(58);
        obj3.setData(60);
        obj1.getStaticValue();
        obj2.getStaticValue();
        obj3.getStaticValue();
}
```

- **Static Member function.**
- A static member function can access static data members and static member functions inside or outside of the class.
- A static member function can be called even if no objects of the class exist and the static function are accessed using only the class name and the scope resolution operators.
- **Code C++.**

```
#include <iostream>
using namespace std;
class Box
{
   private:
   static int length;
   static int breadth;
   static int height;
   public:
   static void print()
   {
      cout << "The value of the length is: " << length << endl;
      cout << "The value of the breadth is: " << breadth << endl;
      cout << "The value of the height is: " << height << endl;
   }              };
// initialize the static data members
int Box :: length = 10;
int Box :: breadth = 20;
int Box :: height = 30;
// Driver Code
int main()
{
   Box b;
   cout << "Static member function is called through Object name: \n" <<
endl;
   b.print();
   cout << "\nStatic member function is called through Class name: \n" <<
endl;
   Box::print();
   return 0;
```

- **Encapsulation:** Encapsulation means binding data and methods within a class providing control over the accessibility and it prevents external code from directly modifying the internal data of an object.
- **Code C++.**

```
#include<iostream>
using namespace std;
class Encap
{
	private:
		int age;
		void show(){
			cout<<"Shanelhai";
		}
	public:
		string name;
		void setvalue(int a){
			show();
			age = a;
			cout<<age<<endl;
		}
};
int main(){
	Encap E;
	E.name = "Shanelhai";
	E.setvalue(21);
	cout<<E.name;
}
```

- **Data Abstraction:** Data Abstraction is a programming technique that focuses on providing only the essential information about the data to the outside world, while hiding the background details or implementation.
- **Code C++.**

```
#include<iostream>
using namespace std;
class Car{
        bool startEngine;
        public:
                Car():startEngine(false){
                }
        void start(){
                //startEngine = true;
                cout<<"Engine Started:"<<endl;
        }
        void drive(){
                if (startEngine){
                        cout<<"You are ready to drive:";
                }
                else{
                        cout<<"Can't drive the car:";
                }
        }
};
main(){
        Car c;
        c.start();
        c.drive();
}
```

- **Abstract:** A class which contain at least one pure virtual function we can't declare the object of abstract class.
- **Code C++.**

```
#include<iostream>
using namespace std;
class Animal{
	public:
		virtual void eat()=0;
};
class Dog:Animal{
	public:
		void eat(){
			cout<<"Dog eat Rice:"<<endl;
		}
};
class Cow:Animal{
	public:
		void eat(){
			cout<<"Cow eat grass:"<<endl;
		}
};
int main(){
	Dog D;
	Cow C;
	D.eat();
	C.eat();
}
```

- **Polymorphism:** polymorphism is a concept in which an object can be treated in different ways. It means that objects of a class can be be used as objects of their derived classed.
- **There are two types of polymorphism.**
- **Static Binding (or Compile time) Polymorphism**: Static binding polymorphism is also known as early binding or compile-time polymorphism. It is achieved through function overloading and operator overloading.
- **Function overloading:-** function overloading in defined as the process two or more function with the same name, but different types of parameters called function overloading.
- **Code C++.**

```
#include<iostream>
using namespace std;
int natural()
{
        int a,b,add;
        cout<<"\n enter the numbera";
        cin>>a;
        cout<<"\n enter the numberb";
        cin>>b;
        add=a+b;
        cout<<"\n the sum of two number"<<add;
}
int natural(double a,double b)
{
        double sub;
        sub=a-b;
        cout<<"\n the sub of two number"<<sub;
}
int main()    {
natural();
natural(3,2);
return 0;
}
```

- **Operator overloading: -** To assign more than one operation on a same operator know as operator overloading to achieve operator overloading, we have to write a special function known as operator ().
- **You can write operator () in two ways:-**
- Class function.
- Friend function.
- **Following is the list of operators that can't be overloaded.**
- Dote operator(.).
- Scop resolution operators(::).
- Conditional operators.
- Size of operators.
- **There are two types of operator overloading.**
- **Unary operator overloading: -** A operator which contain only one open and is called unary operator overloading.
- **Binary operator overloading: -** A operator which contain two operands is called binary operator overloading.
- **Code C++.**

```
#include<iostream>
using namespace std;
class A                                 {
        int a,b;
        public:
        void setdata()                  {
        cout<<"\n enter the number a";
        cin>>a;
        cout<<"\n enter the number b";
        cin>>b;                         }
friend void add(A obj);                 };
void add(A obj)                         {
        int sum;
        sum=obj.a+obj.b;
cout<<"\n sum"<<sum;                    }
int main()                              {
        A c;
        c.setdata();
        add(c);
        return 0;                       }
```

- **Unary operator overloading: -** A operator which contain only one open and is called unary operator overloading.
- **Code C++.**

```
#include<iostream>
using namespace std;
class demo{
        int a, b;
        public:
        demo(int x, int y)
        {
                a = x;
                b = y;
        }
        void show()
        {
                cout<<"A"<<a<<"B"<<b<<endl;
        }
        friend void operator -(demo &obj);
};
void operator -(demo &obj)
{
        obj.a = -obj.a;
        obj.b = -obj.b;
}
int main(){
        demo ob(10,-20);
        ob.show();
        -ob;
        ob.show();
}
```

- **Binary operator overloading: -** A operator which contain two operands is called binary operator overloading.
- **Code C++.**

```
#include<iostream>
using namespace std;
class demo{
        int a,b;
        public:
        demo(){
        }
        demo(int x, int y){
                a = x;
                b = y;
        }
        void show(){
                cout<<"\nA"<<a<<"\nB"<<b<<endl;
        }
        friend demo operator +(demo &obj,demo &obj2);
};
demo operator +(demo &obj,demo &obj2){
        demo temp;
        temp.a=obj.a+obj2.a;
        temp.b=obj.b+obj2.b;
        return temp;
}
int main(){
        demo ob(10,20); demo ob1(30,40); demo ob2;
        ob2=ob+ob1;
        ob2.show();
}
```

- **Dynamic Binding(Runtime)polymorphism :** Dynamic binding is the process of binding a function call to a specific function at runtime. This allows the programmer to write code that is more flexible and reusable.
- **Virtual function.**
- A virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword. It is used to tell the compiler to perform dynamic linkage or late binding on the function.
- **Code C++.**

```
#include<iostream>
using namespace std;
class Base              {
        public:
                char name, address;
                int age,salary;
                virtual void display(){
                        cout<<"\n Enter the name:";
                        cin>>name;
                        cout<<"\n Enter the age:";
                        cin>>age;
                        cout<<"\n Enter the salary:";
                        cin>>salary;
                        cout<<"\n Enter the address:";
                        cin>>address;               }               };
class Derived :public Base                          {
        public:
                void display(){
                        cout<<"Detials of employes:";
                        cout<<name<<age<<salary<<address<<endl; }       };
int main(){
        Base b, *bptr;
        Derived d;
        cout<<"bptr points to base\n";
        bptr=&b;
        bptr->display();
        cout<<"bptr points to derived\n";
        bptr=&d;
        bptr->display();                    }
```

- **Function overriding.**
- Whenever we writing function in base and derived class in such a way that function name parameter must be same is called function overriding.
- **Code C++.**

```
#include<iostream>
using namespace std;
class A{
        public:
                int number,even=0,odd=0,i;
                 void display(){
                        cout<<"\n enter the number";
                        for(i=1;i<=10;i++)                  {
                        cin>>number;
                        if(i%2==0)
                        even=even+number;
                        else
                        odd=odd+number;                  }
                        cout<<"\n even number"<<even;
                        cout<<"\n odd number"<<odd;
                }               };
class B:public A{
        public:
                int a,b,c;
                void display(){
                        cout<<"Enter the number a:";
                        cin>>a;
                        cout<<"Enter the number b:";
                        cin>>b;
                        c = a + b;
                        cout<<"The total sum of two number:"<<c;
                }               };
int main(){
        B obj;
        obj.A::display();
        //scope resolution operater.
        obj.display();
}
```

- **Exception Handling:** Exception handling is the process of handling errors that occur during program execution. This allows the programmer to write code that is more robust and reliable.
- An exception is unexpected / unwanted / abnormal situation that occurred at runtime.
- **Code C++.**

```
#include<iostream>
#include<stdexcept>
using namespace std;
main(){
	double bal = 1000.0;
	try{
		double amt;
		//Deposit.
		cout<<"Enter Deposit Amount:";
		cin>>amt;
		if(amt<=0){
			throw invalid_argument("Invalid Deposit Amount:");
		}
		bal = bal + amt;
		cout<<"Available Amout:"<<bal<<endl;
		//withdrown.
		cout<<"Enter Withdrawan Amount:";
		cin>>amt;
		if(amt<=0){
			throw invalid_argument("Invalid Withdrawan
Amount:");
		}
		if(amt>bal){
			throw runtime_error("Insufficient Found:");
		}
		bal = bal - amt;
		cout<<"Available Amout:"<<bal<<endl;
	}
	catch(exception& e){
		cout<<e.what();
	}		}
```

- **Templates.**
- A template in C++ is a blueprint or formula for creating a generic class or function. It allows a function or class to work on different data types without being rewritten.
- **There are two types of templates in C++:**
- **Function templates:**
- A function template is a function that can be used with different data types. For example, you can create a function template that can sort a list of any data type, such as integers, floats, or strings.
- **Code C++.**

```
#include<iostream>
using namespace std;
template<class T>
void show(T a, T b){
        cout<<"A="<<a<<endl<<"B="<<b<<endl;
}
int main(){
        int p = 10, q = 20;
        char name='a', n='b';
        show(p,q);
        show(name,n);
}
```

- **Class templates:**
- A class template is a class that can be used to create objects of different data types. For example, you can create a class template that can be used to create a stack of any data type, such as integers, floats, or strings.
- **Code C++.**

```
#include<iostream>
using namespace std;
template<typename T>
class Example{
        T a,b;
        public:
                Example(T x, T y)
                {
                        a =x;
                        b =y;
                }
                void add()
                {
                        cout<<"\n Addition:"<<a+b;
                }
};
int main(){
        string str1 = "Hello";
        string str2 = "Google";
        Example <int>obj1(100,200);
        Example <float>obj2(20.5,30.9);
        Example <string>obj3(str1,str2);
        obj1.add();
        obj2.add();
        obj3.add();
}
```

- **File handling.**
- File handing is a process of reading and writing data to files allow C++ file handling you to store output of a program in a file and also read file data on console.
- **File handling operations:**
- Create a file.
- Open file.
- Read file.
- Write file.
- Delete file.
- Copy file.
- **Create a file and open file.**
- **Code C++.**

```
#include<iostream>
#include<fstream>
using namespace std;
main(){
        ofstream onfile;
        onfile.open("C:\\Users\\Admin\\Desktop\\New folder\\file.txt");
        onfile<<"Thank U so Much:)";
        cout<<"Date has been written in the file:)";
        onfile.close();
}
```

- **Read file and write file.**
- **Code C++.**

```
#include<iostream>
#include<fstream>
using namespace std;
main(){
        ifstream infile; string str;
        infile.open("C:\\Users\\Admin\\Desktop\\New folder\\file.txt");
        while(getline(infile,str)){
                cout<<str;
        }
        infile.close();
}
```

- **Copy file paste.**
- **Code C++.**

```
#include<iostream>
#include<fstream>
using namespace std;
main(){
	ifstream infile;
	ofstream onfile; char str;
	infile.open("C:\\Users\\Admin\\Desktop\\New folder\\file.txt");
	onfile.open("C:\\Users\\Admin\\Desktop\\New folder\\file.txt");
	while(infile.get(str)){
		onfile.put(str);
	}
	cout<<"Copied !!";
	infile.close();
	onfile.close();
}
```

- **Delete file.**
- **Code C++.**

```
#include<iostream>
#include<fstream>
using namespace std;
main(){
	int value = remove("C:\\Users\\Admin\\Desktop\\New
folder\\file.txt");
	if(value == 0){
		cout<<"file Deleted!";
	}else
	{
		cout<<"file not Deleted!";
	}
}
```