5/11/2023

* string :-

- To access string in c or c++ we have to use character array. Using charactor array makes string operations difficult, because we have to think about base addresses and some inbuilt function.

- So to avoid those difficulties c++ has implemented its own class and taken care of all the dificulties we have to face with char array string we just have to create object of class string to work with strings using class string we can treat our strings as primitive data-type variable because of which we can use operators of primitive datatype with string also. e.g we can use "==" '<', '>' for comparison of string. we don't need to use functions. example

```
int main()
{
    char str[30] = "hello";
    //str = "hello" ;   // Not possible
    strcpy(str, "hello");
    cin.getline(str,30);
    cout<<str;

    string str1;
    str1 = "bye";
    cout<<str1;
}
```

```
int main()
{
    string s1, s2;
    s1 = "Hello";
    s2 = "Bye";

    cout<<"s1 = "<<s1;
    cout<<"s2 = "<<s2;

    cout<<"Entre a string = ";
    getline(cin, s1);
    cout<<"Enter a string = ";
    getline(cin, s2);

    cout<<s1;
    cout<<s2;
}
```

* Comparison of 2 string

```
int main()
{
    string s1, s2;
    s1 = "hello";
    s2 = "bye";

    if (s1 == s2)
    {
        cout<<"Equal String";
    }
    else
    {
        cout<<"Not equal string";
    }
}

int main()
{
    string s1, s2;
    s1 = "cat";
    s2 = "bat";

    if (s1 < s2)
    {
        cout<<s1<<"Is less than "<<s2;
    }
    else
    {
        cout<<s1<<"Is Greater than"<<s2;
    }
}
```

* Some functions for string

1) length(); → Tells length of string

```
int main()
{
    string s1, s2;
    s1 = "bat";
    s2 = "cat";
    int len;

    len = s1.length();
    cout<<" length of s1 ="<<len;   //3

    len = s2.length();
    cout<<" length of s2 ="<<len;   //3
}
```

2) Concatination :-

- To concatinate 2 string we will use '+' operator. '+' operater will concatinate 2 strings and will generate 3rd string without any changes in source string.

```
int main()
{
    string s1, s2;

    s1 = "hello from cpp";
    s2 = "cat";
    string s3;
    s3 = s1 + " and " + s2;

    cout<<"\n s3 = "<<s3;
    cout<<"\n s1 = "<<s1;
    cout<<"\n s2 = "<<s2;
}
```

o/p = hello from cpp and cat
hello from cpp.
cat

* It is possible to access ith character of string by using [ ] bracket.

```
int main()
{
    string s1;
    s1 = "hello";

    while (i<=s1.length())
    {
        cout<<s1[i]<<"\n";
        i++;
    }
}
```

o/p = h
e
l
l
o

3) at ();

- We pass index as a parameter to at()functi on and it returns reference to character at given index in string.

```
int main()
{
    string str;
    str = "hello"
    str.at(0) = 'H';

    cout<<str;      // Hello
    cout<<str.at(2);  // l
}
```

4) append ()

- We use this function for concatination of two string this function makes changes in calling object after concatination.

```
int main()
{
    string S1, S2;
    S1 = "Hello";

    S2 = S1.append("Hii");
    cout<<S2;      // HelloHii
    cout<<S1;      // HelloHii
}
```

5) compare ();

- We use compare function to compare 2 string objects it works exactly same as strcmp() compare() function may return -1, 0, 1 in following condition.

S1 < S2 ⇒ -1
S1 == S2 ⇒ 0
S1 > S2 ⇒ 1

```
int main()
{
    string S1 = "hat";
    string S2 = "bat";

    int res = S1.compare(S2);

    cout<<res;   // 1
}
```

Q Scan six city names. count how many times pune present.

```
int main()
{
    string city[6]

    int i;
    for(i=0; i<6; i++)
    {
        ~~if~~(cout<<"Enter city name =";
        getline (cin, city[i]);
    }

    for (i=0; i<6 ; i++)
    {
        if(city[i].compare("pune") ==0)
        {
            count++;
        }
    }
    cout<<count;
}

also → if(city[i] == "pune");
        {
            count++;
        }
```

```
class Student
{
    pblic :
        string stdname ;
        int age;

        void scan()
        {
            cout<<"Enter Name = ";
            getline(cin, stdname);
            cout << " Enter Age = ";
            cin>>age;
        }
};
int main()
{
    Student std[6];
    int i;
    for( i=0 ; i<6 ; i++)
    {
        std[i].scan();
    }

    for(i=0; i<6; i++)
    {
        if(std[i].stdname == "Sachin");
        {
            cout<<"present";
            break;
        }
    }
}
```

```
    if(i==6)
    {
        cout<<"Not found";
    }
}
```

Q Sort 8 integers array

```
int main()
{
    int arr[8];
    int i;
    for(i=0;i<8;i++)
    {
        cin>>arr[i];
    }

    for(i=0;i<8;i++)
    {
        for(j=i+1;j<8;j++)
        {
            if(arr[i]<arr[j])
            {
                temp = arr[i];
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
    }
    cout
}
```

* Selection Sort

if(num[i] > num[j]

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 7 | 5 | 2 | 1 |
| i | j | | | |
| 10 | 10 | 5 | 2 | 1 |
| i | ~~j~~ | j | | |
| 5 | 10 | 7 | 2 | 1 |
| i | ~~j~~ | ~~j~~ | j | |
| 2 | 10 | 7 | 5 | 1 |
| i | ~~j~~ | ~~j~~ | ~~j~~ | j |
| $\boxed{1}$ | 10 | 7 | 5 | 2 |
| ~~i~~ | i | j | | |
| 1 | 7 | 10 | 5 | 2 |
| | i | ~~j~~ | j | |
| 1 | 5 | 10 | 7 | 2 |
| | i | ~~j~~ | ~~j~~ | j |
| $\boxed{1}$ | $\boxed{2}$ | 10 | 7 | 5 |
| | ~~i~~ | i | j | |
| 1 | 2 | 7 | 10 | 5 |
| | | i | ~~j~~ | j |
| $\boxed{1}$ | $\boxed{2}$ | $\boxed{5}$ | 10 | 7 |
| | | | i | j |
| $\boxed{1}$ | $\boxed{2}$ | $\boxed{5}$ | $\boxed{7}$ | $\boxed{10}$ |

* Sorting Array of objects.

```
class Student
{
    public:
        string name;
        int age;

        void print()
        {
            cout<<name;
            cout<<age;
        }
        void scan()
        {
            cout<<"Enter name =";
            getline(cin,name);
            cin>>age;
        }
};
int main()
{
    student std[6];

    int i,j;

    for(i=0;i<6;i++)
    {
        std[i].scan();
    }

    for(i=0;i<6;i++)
    {
        for(j=i+1;j<6;j++)
        {
            temp
            if(std[i].age > std[j].age)
            {
                temp = std[i];
                std[i] = std[j];
                std[j] = temp;
            }
        }
    }
    for (i=0;i<6;i++)
    {
        std[i].print();
    }
}
```

```
class student
{
    public:
        string name;
        int age;

        void scan()
        {
            getline(cin,name);
            cin>>age;
        }
        void print()
        {
            cout<<name;
            cout<<age;
        }
};
int main()
{
    student std[6];

    int i,j;

    for(i=0; i<6; i++)
    {
        std[i].scan();
    }
    for(i=0; i<6; i++)
    {
        std[i].print();
    }
```

```
for(i=0; i<6; i++)
{
    for (j=i+1; j<6; j++)
    {
        if(std[i].name > std[j].name)
        {
            temp = std[i];
            std[i] = std[j];
            std[j] = std[i];
        }
    }
}

for(i=0; i<6; i++)
{
    std[i].print();
}
}
```

```
class student
{
    private:
        string name;
        int age;

    public:
        void scan()
        {
            getline(cin, name);
            cin>>age;
        }

        int compare (student std)
        {
            if(name > std.name)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }
        void print()
        {
            cout<<name;
            cout<<age;
        }
};
```

```
int main()
{
    student std[6];
    int i, j;

    for(i=0; i<6; i++)
    {
        std[i].scan();
    }

    for(i=0; i<6; i++)
    {
        for(j=i+1; j<6; j++)
        {
            int res = std[i].compare(std[j]);
            if (res==1)
            {
                temp = std[i];
                std[i] = std[j];
                std[j] = temp;
            }
        }
    }

    for(i=0; i<6; i++)
    {
        std[i].print();
    }
}
```

```
class Employee
{
    private:
        string eme;
        int age;
        string hometown;

    public:
        void scan()
        {
            getline(cin, name);
            cin>>age;
            getline(cin, town);
        } ei

        void print()
        {
            cout<<name;
            cout<<age;
        }

        int compare(Employee temp)
        {
            if(town > temp.town)
            {
                return 1;
            }
            else
            {
                retuen 0;
            }
        }
};
```

```
int main()
{
    Employee emp[8],temp;
    int i,j;

    for(i=0 ; i<8 ; i++)
    {
        emp[i].scan();
    }

    for (i=0 ; i<8 ; i++)
    {
        for(j=i+1 ; j<8 ; j++)
        {
            if(emp[i].compare(emp[j]) == 1)
            {
                temp = emp[i];
                emp[i] = emp[j];
                emp[j] = temp;
            }
        }
    }

    for(i=0; i<8 ; i++)
    {
        emp[i].print();
    }
}
```

## * Operator Overloading

- Basic purpose of operator overloading is to treat user defined data-type as a primitive data-type

- Various ~~operation~~ operators in c++ are designed to work only with primitive data-types. To make them to work with user defined datatype also we use operator overloading

Take example to perform addition of 2-obj T1 and T2 we can't write T3 = T1 + T3 because '+' works with only primitive datatype by default. hence to make it work with T1 and T2 also, we have to overload '+' operator.

```
class Number
{
    public:
        int x, y;
};
int main()
{
    Number T1, T2, T3;
    T1.scan();
    T2.scan();
    T3.add(T1, T2);
    T3.show();
```

```
    cin >> T1 >> T2;
    T3 = T1 + T2;
    cout << T3;
```

} Not possible without operator overloading.

```
    Number T1, T2, T3, T4;
    Number res;
    res = T1.multi(T2);
    res = res.multi(T3);
    res = res.multi(T4);

    res = T1.multi(T2.multi(T3.multi(T4)));

    res = T1 * T2 * T3 * T4;   //Not possible
}
```

End next day

7/10/2023

- In operator overloading we assign some extra meaning to our operators, so that they will work for user defined datatype also. while doing this it is programmers responsibility that the basic meaning of operator should not be chang it means, if you overload '+' operator then you should use it to addition only should not perform multiplication using '+' operator.

- Operator overloading is nothing but simple function call only when we overload any operator we just define operator overloading function which gets called automatically, as this function gets called automatically, its name only is predefined.

7/11/2023

The name of operator overloading function must be operatorop where operator is keyword and op is operator to be overloaded
e.g to overload '+' operator, name of operator overloading function is ~~operator+;~~

```
class Number
{
    public:
        int x, y;

        Number ()
        {
        }
        ~~Number~~
        void show()
        {
            cout << x;
            cout << y;
        }
};
Number Operator + (Number temp1, Number temp2)
{
    Number add;

    add.x = temp1.x + temp2.x;
    add.y = temp1.y + temp2.y;

    return add;
}
```

```
int main()
{
    ~~Number T1, T2, T3;~~
    Number T1(2,6), T2(3,6), T3;
    T3 = T1 + T2;                    ①
    // T3 = operator+(T1, T2);
    T3.show()
}
```

- In this program we have overloaded '+' operator by defining operator + function due to this function we can use '+' operator to add 2 objects of class Number [as we did at ①]

- Operator overloading functions gets called only when you are using that operator with at least one user defined datatype variable.

- Following four operator cannot be overload

1) sizeof()
2) : :  scope resolution.
3) ? :  conditional
4) .  dot operator

## Types of operator

| Unary | Binary |
|---|---|
| ++, --, !, ~, - | +, -, *, /, %, <, <=, >, >=, !=, ==, <<, >>, ^, &, \|, &&, \|\| |
| member | idependent |

1) Overloading unary operator using member function :-

- While overloading unary operator using the member function we do not pass any parameter to memberfunction. The only operant on which your operator is working will go as a calling obj to operator overloading function

1) Overloading unary '—' operator.

```
int main()
{
    int a = 5;
    inb b;
    b = -a;
    cout<<a<<b
}
```

o/p 5 -5

```
int
class Number
{
    public:
        int x, y;

        Number()
        {
        }
        Number(int a, int b)
        {
            x = a;
            y = b;
        }
        Number operator-()
        {
            Number ans;
            ans.x = -x;
            ans.y = -y;
            return ans;
        }
};
int main()
{
    Number T1(2,3);
    Number T2;

    T2 = -T1;
    // T2 = T1.operator-();
    cout<<T1.x<<T2.x ;          -2,-3
}
```

```
class Number
{
    public:
        int x, y;

        Number (int a, int b)
        {
            x = a;
            y = b;
        }

        Number operator!()
        {
            Number ans;
            ans.x = x * x * x;
            ans.y = y * y * y;

            return ans;
        }
};
int main()
{
    Number T1(2, 5);
    Number T2;

    T2 = !T1;

    T2.show();        8 , 125
}
```

8/11/2023

2) Overloading Binary operator using member function -

while overloading Binary operator using member function left hand side operand works as a calling object for operator overloading function and right hand side object go as a parameter that is why operator overloading function takes only one parameter while using the OBO you cannot give multiple parameter

```
class Number
{
    public:
        int x, y;

        Number (int a, int b)
        {
            x = a;
            y = b;
        }

        Number operator+ (Number Temp)
        {
            Number Ans;
            Ans.x = x + Temp.x;
            Ans.y = y + Temp.y;

            return Ans;
        }
};
```

```
        void show()
        {
            cout<<x<<y;
        }
};
int main()
{
    Number T1(2,3), T2(1,4), T3;

    T3 = T1 + T2;
    // T3 = T1.operator+(T2);

    T3.show()
}
```

- class Number

```
{
    int x,y;
    public:
        Number (int a, int b)
        {
            x=a;
            y=b;
        }

        Number operator+ (Number Temp)
        {
            Number ans;
            ans.x = x + Temp.x;
            ans.y = y + Temp.y;
            Return Ans;
}
```

```
};
int main()
{
    Number T1(1,3), T2(2,4), T3(2,2);
    Number T4;

    T4 = T1 + T2 + T3;
         Ans + T3;

    T4.show();
}
```

Q

```
int main()
{
    Number T1(1,3), T2(2,4), T3(2,2);
    Number T4;

    T4 = T1 * T2 - T3;
    T4.show()  ans - T3
}

class Number
{
    int x,y;
    public:
        Number (int a, int b)
        {
            x=a, y=b;
        }
        Number operator*(Number Temp)
        {
            Number ans;
```

```
        ans.x = x * temp.x;
        ans.y = y * temp.y;

        return ans;
    }

    Number operator- (Number Temp)
    {
        Number ans;
        ans.x = x - Temp.x;
        ans.y = y - Temp.y;

        return ans;
    }

    void show()
    {
        cout<<x;
        cout<<y;
    }
};

int main()
{
    Number T1(1,3)
    Number T4;

    T4 = T1 + 5;

    T4.show();
}
```

```
class Number
{
    int x,y;
    public:
        Number (int a, int b)
        {
            x = a;
            y = b;
        }
        Number ()
        {
        }

        Number operator+ (int a)
        {
            Number ans;
            ans.x = x + a;
            ans.y = y + a;

            return ans;
        }

        void show()
        {
            cout<<x;
            cout<<y;
        }
};
```

1) Overloading unary using inside independant function -

- while overloading unary op using independant function the only object on which your operator is working will go as a parameter because their is no any calling object for independent function.
- In this case operator overloading function takes only one parameter less than or more than one may generate error.

```
class Number
{ public:
  int x, y;

    Number (int a, int b)
    {
        x = a;
        y = b;
    }

    Number ()
    {
    }
};

int main()
{
   Number T1(2,3);
```

```
   T2 = -T1;
   // T2 = operator - (T1);
   T2.show();
}

Number operator - (Number Temp)
{
    Number Ans;
    Ans.x = - Temp.x;
    Ans.y = - Temp.y;
    return Ans;
}
```

} independent function.

Q

```
int main()
{
   Number T1(2,3);
   Number T2;

   T2 = ~T1;   //independant
   T1.show();  — 2 / 3
   T2.show();  — 4 / 9
}

class Number
{
   public:
      int x, y;

      Number (int a, int b)
      {
```

```
        x = a;
        y = b;
    }

    void show()
    {
        cout << x;
        cout << y;
    }
};

Number operator ~ (Number Temp)
{
    Number Ans;
    Ans.x = Temp.x * Temp.x;
    Ans.y = Temp.y * Temp.y;

    return Ans;
}
```

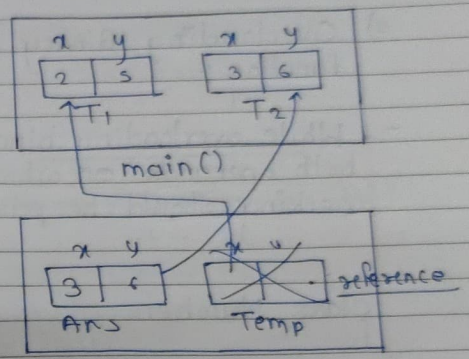
---

Q

```
class Number
{
    int x, y;
    public:
        Number (int a, int b)
        {
            x = a;
            y = b;
        }
        Number ()
        {}
```

++ will increase the value of operand also

```
        void show()
        {
            cout << x;
            cout << y;
        }
};
int main()
{
    Number T1(2, 5);
    Number T2;

    T2 = ++T1;
    T2.show();
    T1.show();
}
```




```
Number operator ++ (Number & temp)
{
    Number ans;
    temp.x = temp.x + 1;
    temp.y = temp.y + 1;

    Ans.x = temp.x;
    Ans.y = temp.y;

    return Ans;
}
```

To make change in both values and to understand about the concept of reference.

9/11/2023

2) Overloading Binary operator using independent function :-

- While overloading binary using independent function both operands on which the your operator is working should be passed as a parameter

```
class Number
{
    public:
        int x, y;

        Number(int a, int b)
        {
            x = a;
            y = b;
        }
        Number()
        {
        }
};
int main()
{
    Number T1(2,3)
    Number T2(4,5);
    ~~T3 = T~~
    // T3 = addition(T1, T2) ⟶ independent
    // T3 = T1.addition(T2) ⟶ member

    T3 = T1 + T2;
    T3 = operator+(T1, T2);
    T3.show()

}

Number operator+(Number T1, Number T2)
{
    Number ans;
    ans.x = T1.x + T2.x;
    ans.y = T1.y + T2.y;

    return ans;
}
```

Next page new point

* Why independent function for operator overloading when we have member function

- while overloading binary operator it is possible that left hand side operand is non object variable in that case we cannot use member function to overload your binary operator because non-object variable on L.H.S cannot work as a calling object.

for example -

```
int main()
{
    Number T1(2,5), T2(3,6);
    Number res;

    res = T1 + 10; —— ① //member function
    res.show();

    res = 10 + T2; —— ② //indipendant function
    res.show();
}
```

we can use member as well as independant function to overload '+' operator at statement ① but at statement ② we have to use independant function for operator overloading because the L.H.S is non-object entity.

Q

```
class Distance
{
    public:
        int km, mtr;

        void operator~()
        {
            cout<<"Enter Kilometer = ";
            cin>>km;
            cout<<"Enter meter = ";
            cin>>mtr;
        }

        void operator!()
        {
            cout<<km;
            cout<<mtr;
        }
};
```

ans →

Q →

```
int main()
{
    Distance D1, D2;

    ~D1;
    ~D2;   } ~ will scan data of object

    !D1;
    !D2;   } ! will print data of object
}
```

```
class Distance
{
    public:
        km, mtr

        Number()
        {
        }
};
```

Q
```
int main()
{
    Distance D1, D2;

    ~D1;
    ~D2;   } scan
    !D1;
    !D2;   } print
}
```

Use independant function.

```
void operator~ (~~Number~~ Distance &temp)
{
    cout<<"Enter km =";
    cin>> temp.km;
    cin>> temp.mtr;
}
void operator! (~~Number~~ Distance temp)
{
    cout<< km;
    cout<< mtr;
}
```

Q
```
int main()
{
    Distance D1, D2;

    D1.scan();
    D2.scan();
```

Do by independ & member seperatly

```
    ~D1;
    ~D2;   } will convert distance in proper format
```

| km | mtr | | km | mtr |
|---|---|---|---|---|
| 2 | 2490 | → | 4 | 490 |

```
    D1.show();
    D2.show();
}

class Distance
{
    private:
        int km, mtr;

        void scan()
        {
            cin>>km;
            cin>>mtr;
        }

        void operator~()
        {
            km = km + mtr/1000;
            mtr = mtr %1000;
        }
```

```
~~D1.show();~~
~~D1.show();~~
void show()
{
  cout<< km;
  cout<< mtr;
};
```

By independant function

```
void operator~(Number & D)
{
    D.km = D.km + D.mtr/1000;

    D.mtr = D.mtr %1000;
}
```

—— × ——

```
int main()
{
  Distance D1, D2; (private km, mtr)

  ~D1;
  ~D2;  } will scan D1, D2 :

  if (D1 > D2)   [Do by both independant and member function also]
  {
    cout<<"D1 > D2";


    cout<<"D1< D2";
```

* By member function

```
class Distance
{
    int km, mtr;
    public:

        void operator~()
        {
            cin>> km;
            cin>> mtr;
        }
        int
        ~~void~~ operator>(Distance Temp)
        {
            ~~km = km + mtr/1000;~~
            ~~mtr = mtr %1000;~~

            ~~Temp.km = Temp.km + Temp.mtr/1000;~~
            ~~Temp.mtr = Temp.mtr + Te %1000;~~

            int co;
            int tp;

            co = km * 1000 + mtr;
            tp = Temp.km * 1000 + Temp.mtr;

            if (co > tp)
            {
                return 1;
            }
            else
            { return 0 }
};
```

* By using independant

```
class Distance
{
    int km, mtr
    public:
        ~~void scan()~~
        ~~{~~
    friend void operator~(Distance & temp);
    friend int operator>(Distance D1, Distance D2);
};

void operator~(Distance &Temp)
{
    cin>> Temp.km;
    cin>> Temp.mtr;
}

int operator>(Distance D1, Distane D2)
{
    int co = D1.km * 1000 + D1.mtr;
    int tm = D2.km * 1000 + D2.mtr;

    if (co > tm)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Q) Scan 2 students and check the name is present or not do not use any type of operator overloading

```
class Student
{
    public:
        char name[20];
        int age;
};

int main()
{
    Student S1, S2;
    char name[20];
    cout<<"Enter name =";
    cin.getline(S1.name, 20);
    cout<<"Enter age =";
    cin>> S1.age;

    cout<<"Enter name =";
    cin.getline(S2.name, 20);
    cout<<"Enter age =";
    cin>> S2.age;

    cout<<"Enter a name =";
    cin.getline(name, 20);

    if(strcmp(S1.name, name) == 0)
    {
        cout<<"Present";
    }
else if ~~ifelse~~ (strcmp(S2.name, name) == 0)
    {
        cout<<"Present";
```

```
        }
        else
        {
            cout<<"Not Present";
        }
    }
```

* Same code by using operator overloading

1) Member function:-

```
class Student
{
    public:
        char name[30];
        int age;

        int operator==(char *p)
        {
            if (strcmp(name,p) == 0)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }
};
```

comparing object with string is illlogical
we are doing it for operator overloading
understanding ↓

or non object

```
int main()
{
    Student S1, S2;
    //scan it 1st
    if (S1 == str)
    {
        cout<<" found";
    }
    else if (S2 == str)
    {
        cout <<" found";
    }
    else
    {
        cout <<" Not found";
    }
}
```

str is string to search
scan it and pass pass
as base address.
also catch by the
pointer.

next page new point

* Overloading increment (++) and decrement (--) operator :-

1) Overloading of pre increment (++)

```
class Number
{
    public:
        int x, y;

        Number()
        {
            x = 0;
            y = 0;
        }
        Number (int a, int b)
        {
            x = a;
            y = b;
        }

        void operator++()
        {
            x = x + 1;
            y = y + 1;
        }
        void sohow()
        {
            cout << x;
            cout << y;
        }
};
```

```
int main()
{
    Number T1 (2,3), T2;
    ++T1;        //preincrement
    T1.show();
}
```

Ouput = 3 4

2) while overloading post increment or decrement we have to catch one integer parameter at operator overloading function. Due to this parameter compilor can differentiate between operator overloading function of pre increment

```
class Number
{
    public:
        int x, y;

        Number()
        {
            x = 0;
            y = 0;
        }
        Number (int a, int b)
        {
            x = a;
            y = b;
        }
```

```
void operator++ (int a)
{
    cout<<"\n Post increment ="<<a;
    x = x+5;
    x = y+5;
}

void operator++()
{
    cout<<"\n Pre increment";
    x = x+2;
    y = y+3;
}

void show()
{
    cout<<x;
    cout<<y;
}
};
int main()
{
    Number T1(2,3), T2;

    T1++;        mapping
    T1.show();

    ++T1;
    T1.show();
}
```

Same code by ~~Ide~~ Independant function.

```
void operator++ (Number&Temp, int a)
{
    cout<<"In post increment ="<<a;
    Temp.x = Temp.x + 1;
    Temp.y = Temp.y + 1;

}

void operator++ (Number&Temp)
{
    cout<<"In pre increment ";
    Temp.x = Temp.x + 1;
    Temp.y = Temp.y + 1;
}
```

10/11/2023

```
Q class Number
  {
    private:
        int x, y;

    public:
        void operator ~ ()
        {
            cout << "Enter 2 Numbers = ";
            cin >> x;
            cin >> y;
        }

        void show()
        {
            cout << x;
            cout << y;
        }

        friend Number + (Number T1, Number T2);

  };
int main()
{
    Number T1, T2, T3;
    Number T4;

    ~T1;
    ~T2;   } member function

    T3 = T1 + T2;
```

```
    T4 = T1 * T2 + T3;
         ans  *T3
    T3.show();
    T4.show();
}

Number operator + (Number T1, Number T2)
{
    Number ans;
    ans.x = T1.x + T2.x;
    ans.y = T1.y + T2.y;

    return ans;
}

Number operator * (Number T1, Number T2)
{
    Number ans;
    ans.x = T1.x * T2.x;
    ans.y = T1.y * T2.y;

    return ans;
}
```

Sort a array of 8 integer in ascending.

Q

```
int main()
{
    int num[8];

    for(int i=0; i<8; i++)
    {
        cin>> num[i];
    }

    for(int i=0; i<8; i++)
    {
        for(int j=i+1; j<8; j++)
        {
            if (num[i] > num[j])
            {
                int temp;
                temp = num[i];
                num[i] = num[j];
                num[j] = temp;
            }
        }
    }

    for(int i=0; i<8; i++)
    {
        cout<<num[i];
    }
}
```

```
class Distance
{
    public:
        int meter;

        Distance ()
        {
            cout<<" Enter Distance in meter =";
            cin >> mtr;
        }
};
int main()
{
    Distance dist[8];

    for(i=0; i<8; i++)
    {
        for (j= i+1; j<8; j++)
        {
            if(dist[i].mtr > dist[j].mtr)
            {
                Distance temp;
                temp = dist[i];
                dist[i] = dist[j];
                dist[j] = temp;
            }
        }
    }
}
```

Q

```
class Distance
{
    private:
        int mtr;

        Distance ()
        {
            cout<<"Enter a distance =";
            cin>> mtr;
        }

        Distance (int a)
        {

        }

        int compare (Distance Temp)
        {
            if(mtr > Temp. mtr)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }

        void print()
        {
            cout<< mtr;
        }
    }
};
int main()
{
    Distance Dist[8];

    for (i=0; i<8 ; i++)
    {
        for(j=i+1 ; j<8 ; j++)
        {
            if (Dist[i].compare(Dist[j])==1)
            {
                Distance temp(0);

                temp = Dist[i];
                Dist[i] = Dist[j];
                Dist[j] = temp;
            }
        }
    }

    for (i=0 ; i<8 ; i++)
    {
        Dist[i]. print();
    }
}
```

Q

```
class Distance
{
  private :
     int mtr ;

     Distance ()
     {
       cout<<"Enter Distance mtr =";
       cin >> mtr ;
     }

     Distance (int a)
     {

     }

     int operator> (Distance Temp)
     {
       if(mtr > Temp.mtr)
       {
         return 1 ;
       }

       else
       {
         return 0 ;
       }
     }
};
```

```
int main()
{
    Distance Dist[8];

    for (i=0; i<8 ; i++)
    {
        for(j=i+1 ; j<8 ; j++)
        {
            if (dist[i] > dist[j])
            {
                Temp = dist[i];

                dist[i] = dist[j];

                dist[j] = Temp;
            }
        }
    }
}

print it
```

Q

```
class Distance
{
  public:
      int km, mtr;

      int operator > (Distance Temp)
      {
         int ans;
         ans = km * 1000 + mtr;
         if (ans > Temp.km * 1000 + Temp.mtr)
         {
            return 1;
         }
         else
         {
            return 0;
         }
      }

      int operator < (Distance Temp)
      {
         int ans;
         ans = km * 1000 + mtr;

         if (ans < Temp.km * 1000 + Temp.mtr)
         {
            return 1;
         }
         else
         {
            return 0;
         }
```

```
};

int main()
{
    Distane D1, D2;

    if (D1 > D2)
    {
        cout << " D1 > D2 ";
    }
    else if (D1 < D2)
    {
        cout << " D1 < D2 ";
    }
    else
    {
        cout << " D1 = D2 ";
    }
}
```

Q Scan 3 distane from user. find the largest one

```
class Distance
{
    public :
        int km, mtr;

        int operator> (Distane Temp)
        {
            int T1, T2;
            T1 = km *1000 + mtr;

            if (T1 > T2)
            {
                return 1;
            }
            else
            {
                return 0;
            }
};

int main()
{
    Distance D1, D2, D3;
    Distance ans = D1;

    if (Dns < D2)
    {
        ~~cout<<~~ ans = D2 ;
    }
    if (D3 > ans)
    {
        ans = D3;
    }

    cout << ans.km << ans.mtr;
}
```

next page new point

## * Overloading '<<' and '>>' operator

- cin and cout both are defined to work with primitive datatypes to make them work with user define datatype we have to overload insertion and extraction operator.
After overloading this operator, you can scan or print object of your class directly by cin and cout

example :-

```
class Number
{
    public:
        int x,y;
};

main()
{
    Number T1, T2;
    cout<<"Enter data =";

    cin>>T1;        ——— (1)
    cout<<T1;       ——— (2)
}
```

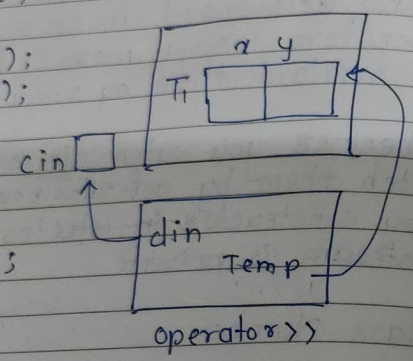
In above example at statement (1) we have to overload extraction operator to overload this operator we have 2 options.

1) Using member function :-
if you overload it using member function. ~~Cin will~~ it means cin will be calling object for operator overloading funtion it means you have to define the function inside class istream but class istream is not accessible to us, Hence we cannot ~~de~~ overload insersion or extraction operators using member function we have to overload them by using independant function.

Note- whenever you catch cin or cout, you have to catch them by reference always because the copy constructor ~~in istr~~ inside istream and ostream is private.

```
class Number
{
    public:
        int x, y;

        void show()
        {
            cout<<x;
            cout<<y;
        }
};
void operator>>(istream &din, Number &Temp)
{
    cout<<"\n Enter data =";
    din>>Temp.x>>Temp.y;
}
```

```
int main()
{
    Number T1, T2;

① —{ cin>> T1;     //Always catch by reference.
   { cin >> T2;
    T1.show();
    T2.show();
}
```




② — cin>>T1 >> T2;

In above example we cannot scan data of T1 and T2 by using one cin, because if we write the statement cin >> T1 >> T2 ~~operator~~ then operator overloading function will get called for cin>>T1 and it will return nothing due to which statement ② will get converted into nothing >> T2 hence compiler generates error.

To run statement ② successfully we have to return cin from operator overloading function. while returning cin and cout from function we have to return by reference because their copy constructor is private.

```
class Number
{
    public:
        int x, y;

        void show()
        {
            cout << x;
            cout << y;
        }
};
istream& operator >> (istream &din, Number &Temp)
{
    cout << —
    din >> Temp.x >> Temp.y;
    return cin; //return din;
}

int main()
{
    Number T1, T2;
    cin>> T1 >> T2;

    T1.show();
    T2.show();
}
```

11/11/202

Scan 8 distances from user sort them in assending

```
class Distance
{
    private:
        int km,mtr;

        void scan()
        {
            cin>>km;
            cin>>mtr;
        }

        int compare(Distance temp)
        {
            if(T1 > T2)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }
};
int main()
{
    Distance Dist[8]
    int i,j;
    for(i=0; i<8; i++)
    {
        Dist[i].scan();
    }
    for(i=0; j<8; i++)
    {
        for(j=i+1; j<8; j++)
        {
            if(Dist[i].compare(Dist[j] == 1)
            {
                temp = Dist[i];
                Dist[i] = Dist[j];
                Dist[j] = temp;
            }
        }
    }
    for(i=0; i<8; i++)
    {
        Dist[i].print()
    }
}
```

Same code by using operator overloading.

```
class Distance
{
    private:
        int km, mtr;

    public:
```

```
Distance()
{
    cin>> km;
    cin>> mtr;
}
Distance (int a)
{
}

int operator> (Distance Temp)
{
    int T1, T2;
    T1 = km * 1000 + mtr;
    T2 = Temp.km * 1000 + Temp.mtr;

    if (T1 > T2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void show()
{
    cout<< km;
    cout<< mtr;
}
```

```
int main()
{
    Distance Dist[8];
    int i, j;
    for(i=0; i<8 ; i++)
    {
        for(j=i+1 ; j<8 ; j++)
        {
            if(Dist[i] > Dist[j])
            {
                Distance Temp(0);

                Temp = Dist[i];
                Dist[j] = Dist[i];
                Dist[j] = Temp;
            }
        }
    }

    for(i=0 ; i<8 ; i++)
    {
        Dist[i].show();
    }
}
```

Q

```
class Time
{
    private:
        int hrs, min;

    public:
        int operator< (Time Temp)
        {
            int T1, T2;

            T1 = hrs *60 + min;
            T2 = Temp.hrs *60 + min ;
            if ( T1 < T2)
            {
                return 1;
            }
            return 0 ;
        }
        friend istream& operator>> (istream &din, Time& Temp);
};

istream & operator >> (istream &din, Time& Temp)
{
    cout<<" Enter hours =";
    cin>> Temp.hrs ;
    cout <<" Enter minutes =";
    cin >> Temp.min ;
    return cin ;
}
```

```
int main()
{
    Time time[8];

    for (i= ; i<=7; i++)
    {
        cin >> time[i];
    }
    Time max;
    max = time[0];

    for (i=0 ; i<8 ; i++)
    {
        if (max < time[i])
        {
            max = time[i];
        }
    }
    cout<<" larges = ";
    max.show();
}   cout << max ;

ostream & operator << (ostream & dout, Time T)
{
    cout << T.hrs ;
    cout << T.min ;
}
```

Why to catch istream and ostream

- copy constructor is the reason for catching istream ostream object by reference

- When we catch object of any class in function then copy constructor of your class will get executed [if you have defined copy constructor then your defined copy constructor will get executed otherwise inbuilt copy constructor provided by the compiler default will get executed] to avoid this execution of copy constructor you have to catch parameter by reference.

if you define copy constructor as private member of your class then you can't catch obj of that class by value you always have to use by reference to catch that object. e.g

```
class Number
{
  public:
     Number()
     {
        cout<<" In default constructor";
     }
private:
    Number(Number & temp)
    {
        cout<<"In copy constructor";
    }
};
void show(Number temp)  ——— ①
{
    cout<<"In show()";
}

void print(Number & temp)  ——— ②
{
    cout<<"In print";
}
int main()
{
    Number T1;
    Number T2;
    show(T2);
    print(T2);
}
```

In case of istream and ostream classes their copy constructor is private hence we can't catch their objects (cin, cout) by value. we have to catch them by reference.

13/11/23

```
class Employee
{
    public:
        string name;
        int salary;
        string hometown;
};
        ~~istream& operator>> (istream~~
        ~~{~~
        ~~}~~

istream& operator>> (istream &din, Employee & temp)
{
    din >> name;
    din >> slary;
    din >> hometown;

    return cin; //return din;
}

ostream& operator<< (ostream& din, Employee temp)
{
    cout << ~~name~~ temp.name;
    cout << temp.salary;
    cout << temp.hometown;
}
int main()
{
    Employee emp[6];
```

Q

```
~~class Dig~~
int main()
{
    int ans;
    Digit obj;
    Number var;
    ans = obj + var;        // member
    cout << ans;

    ans = var * obj;        // member
    cout << ans;

    ans = obj - var;        // independant
    cout << ans;
}

class Digit
{
    public:
        int d;

        int operator+ (Number temp)
        {
            int ans;
            ans = d + temp.n;
            return ans;
        }
}
```

```
class Number
{
    public:
        int n;

        int operator*(Digit temp)
        {
            int ans = n * temp.d;
            return ans;
        }
};

int operator-(Digit T1, Number T2)
{
    int ans;
    ans = T1.d-T2.n;
    return ans;
}
```

—x—x—x—x

Q

```
int main()
{
    Base base;
    Power power;

    cout<< "Enter base";
    cin>> base;

    cout<<"Enter power";
    cin>> power;
    int ans = 1;

    for(i=1; i<= power ; i++)
    {
        ans = ans * base;
    }
    cout<<"Ans =" << ans;
}

class Base
{
    public:
        int b;
}
class Power
{
    public:
        int p;
}

int operator<= (int i, Power Temp)
{
    if (i<= Temp.p)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
int operator * (int ans, Base Temp)
{
        ans = ans * Temp.b;

        return ans;
}

istream& operator >> (istream& din, Base & temp)
{
        din >> temp.b;
        return din;
}

istream& operator << (istream &din, Power &temp)
{
        ~~din << temp.~~
        din >> temp.p;

        return din;  // cin;
}
```

# INHERITANCE

While programing with inheritance, you have to check for following 2 things

1) Who is ready for inheritance from parent class to decide this you have to check access specifier of parent class, only protected and public members ~~members of~~ of parent class will be ready for inheritance

2) After inheritance we have to check that ~~under which access specifier of child class~~ members inheriting from parent class into child class will be stored under which access specifier to decide this we have to check the access specifier used while inheriting the parent class.

```
class Parent
{
  private:
    int x;
    void show();

  protected:
    int y;
    void disp();

  public:
    int z;
    void print()
```

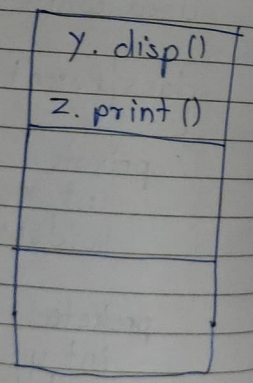
| | |
|---|---|
| x<br>show() | private |
| x<br>disp() | protected |
| z<br>print() | public |

In above class parent only protected and public member are ready for inheritance, Hence y, y.disp(), z.print() will get inherit in child class

1) Inheriting parent using private access specifier.

When we inherit parent using private access specifier all the will be stored under private access specifier of child class.

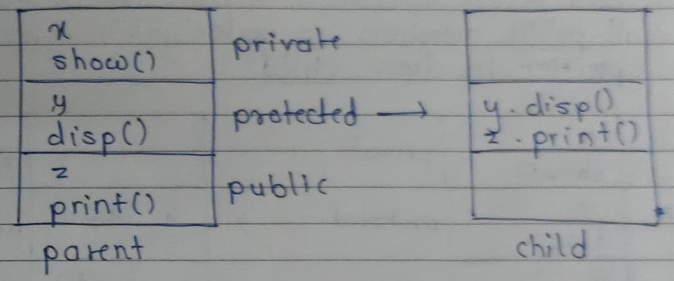
```
class child : private Parent
{
    private :
    protected :
    public :
```




protected y.disp() and public z.print() get inherited.

2) Inheriting parent using protected access specifier.

when you inherit parent class using protected child class, all members coming from parent class will become protected members in child




```
class child : protected Parent
{
    private :
    protected :
    public :
};
```

3) While inherting parent class using public access specifier

While inheriting parent class using public in child class all the members from parent class [which are ready for inheritance] will continue their access specifier in child class it means all public members go under public access specifier in class child

| | | | | |
|---|---|---|---|---|
| x show() | private | | | ~~private~~ |
| y disp() | ~~protected~~ → | | y disp() | ~~protected~~ |
| z print() | public | | z.print() | ~~public~~ |

16/11/2023

* Memory Allocation for Inheritance

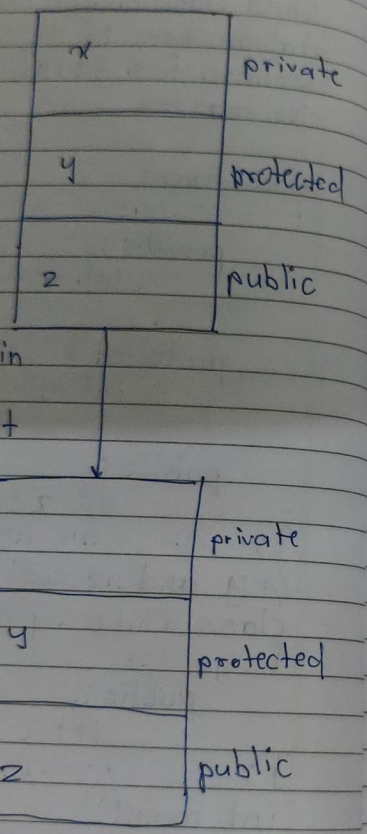
Child class object contains memory for all the datamembers of its own class and it also contains memory for all the datamembers of parent class No matter what is the access specifier

```
class Parent
{
    private :
        int x ;

    protected :
        int y ;

    public :
        int z ;
};
// y and z will go in child
class Child : public Parent
{
    public:
        int c ;
};
int main()
{
    Child obj ;
    cout<<"\nsizeof(obj)="<<sizeof(obj); //16
    //obj contains memory for all datamembers of
     child and parent
}
```

O/P - 16

## * Accessing Members of parent class in Inheritance

```
class Parent
{
    private :
        int x ;
    protected :
        int y ;
    public :
        int z ;
}
// y and z are ready for in

class child : public Parent
{
    private :

    protected :
        //int y
    public :
};
int main()
{
    child obj ;
    obj.c = 10 ;
    obj.z = 20;

    cout<<"\nobj.c = "<<obj.c ;      //10
    cout<<"\nobj.z = "<<obj.z ;      // 20
```




```
class Parent
{
    public :
        int z ;
        void print()
        {
            cout<<"In print()";
        }
    protected :
        int y ;
        void disp()
        {
            cout<<" In disp()";
        }
    private :
        int x ;
};
// y and z are ready for inheritance
class child : protected Parent
{
    private :
    protected :
        //int y ;
        //void disp() ;
        // int z ;
        // void print() ;
    public :
        void myfun()
        {
            cout<<"In myfun";
```

Inherited from parent class

```
        y = 10;
        z = 20;
        disp();  // disp() is working for
                    calling object of myfun()
        print();
    }
    int c;
};
int main()
{
    child obj;
    obj.myfun();
}
```

```
In myfun()
In disp()
In print()
```

* Why protected???

If I want to declare a member in class which must be available for inheritance but it should not be accessible from non-member function, then I have to declare such members using protected access specifier because private and public does not satisfy this requirement.

* Accessing private members of Parent class.

```
class Parent
{
    private:
        int x;
    public:
        void show()      // ready for inheritance
        {
            x = 10;
            cout<<" In show() x = "<<x;
        }
};
class child : public Parent
{
    public:
        // show()
};
int main()
{
    child obj;
    obj.show();
}
```

```
In show() x = 30
```

* class Parent

```
{
    private:
        int x;
    public:
        void show()
        {
            x = 10;
            cout<<" In show() x = "<<x;
        }
};
class child : public Parent
{
    public:
```

```
//void show();        from parent

    void print()
    {
        cout<<"In print()";
        show();
    }
};
int main()
{
    child obj;
    obj.print();
}
```

Output

In print
In show $x = 30$

Different paths

1) obj → show() → $x$

2) obj → print() → show() → $x$

```
class ~~Parent~~ Person
{
    private:
        string name;
        int age;
        string hometown;

    public:
        void scanPerson()
        {
            cout<<"Entee name = ";
            getline(cin, name);
            cout<<"Entee age = ";
            cin>>age;
            cout<<"Entee Hometown = ";
            getline(cin, hometown);
        }

        void showPerson()
        {
            cout<<"Name = "<<name;
            cout<<"Age = "<<age;
            cout<<"Town = " hometown;
        }
};
class Student : private Person
{
    private:
        string clg;
```

```
public :
    void scanstudent()
    {
        scanperson(); //parent class
        cout<<"Enter college =";
        getline(cin, clg);
    }

    void showStudent()
    {
        showperson();
        cout<<"College ="<<clg;
    }
};
class Teacher : private Person
{
    private :
        string name;
        int age;
        string hometown;
        int salary

    public :
        void scanTeacher()
        {
            cout<<"Enter name";
            getline(cin, name);
            cout<<"Enter age";
            cin>>age;
            cout<<"Enter Town =";
            getline(cin, hometown);
            cout<<"Enter salary";
            cin>>salary;
        }

        void showTeacher()
        {
            cout<<name;
            cout<<age;
            cout<<hometown;
            cout<<salary;
        }
};
int main()
{
    Student std;
    std.scanstudent();
    std.showstudent();
}
```

17/11/23

## * Ambiguity in Inheritance

### 1) Single Inheritance

If parent and child both classes contain a member having same name, then when we access that member using child class obj, then priority is always given to copy of child class and not the copy of parent class copy you have to use membership identity label e.g.

```
class Parent
{
    public :
        void show()
        {
            cout<<"In parent show() function";
        }
};
class Child : public Parent
{
    public :
        void show()
        {
            cout<<"In child show() function";
        }
};
int main()
{
    child obj;
    obj.show();   //child show() will get called
    obj.Parent::show();
}
```

output
In child show() function
In parent show() function } First priority is given to the child's func.

```
int main()
{
    Parent obj;
    obj.child::show();   //error
}
```

### 2) Multipule Inheritance

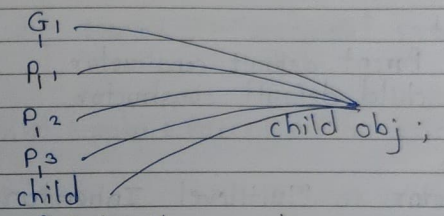
When child class has multiple parents and child is reciving or inheriting multiple members of same name from parent class then compiler may generate ambiguity error while accessing that member. To solve this error you have to use membership identity label

```
class Parent1
{
    public :
        void show()
        {
            cout<<"In parent1 show() function";
        }
```

```
class Parent 2
{
    public:
        void show()
        {
            cout<<"In Parent2 show() function";
        }
};
class child : public Parent1, public Parent 2
{
    public:
};

int main()
{
    child obj;
    obj.show();          // ambiguity

    obj.parent1 :: show();

    obj.parent2 :: show();
}
```

* Constructors in Inheritance.

When we create obj of class child that obj contains memory for all the datamembers of its direct or indirect parent class and contains memory for its own data members




In case of Inheritance, when we create object of child class, constructors of all the parent class of child class will get executed first and then constructor of child class will executed

```
class Parent
{
    public:
        Parent ()
        {
            cout<<"In Parent default constructor";
        }
};
class child : public Parent
{
    public:
        child()
        {
            cout<<" Child of constructor";
        }
};
```

```
int main()
{
    child obj ;
}
```

Output
In Parent default constructor.
In child default constructor.

## * Constructor in Multilevel Inheritance

In case of Multilevel Inheritance constructor executes in sequence of top to bottom. for example if inheritance hirrchical as follows

IMP

Grand
↓
Parent
↓
child

Grand inherited in Parent and Parent in child

Then constructor will execute in this sequence

Grand, Parent then child

```
class Grand
{
    public :
        Grand()
        {
            cout<<" Grand def constructor";
        }
class Parent : Public Grand
{
    public :
        Parent()
        {
            cout<<"\n Parent def constructor";
        }
};
class Child : public Parent
{
    public :
        child()
        {
            cout<<"\n Child def constructor";
        }
};

int main()
{
    child obj ;
}
```

## * Constructors in case of Multiple inheritance

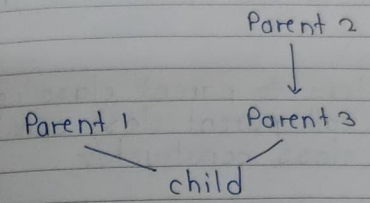
In case of multiple inheritance constructors of parent classes get executed in the sequence of there inheritance it means constructor of 1st inherited class will execute 1st and 2nd for second and so on.

```
class Parent 1
{
    public:
        Parent1()
        {
            cout<<"Parent1 def constructor";
        }
};
class Parent 2
{
    public:
        parent2()
        {
            cout<<"Parent 2 def cons";
        }
};
class child: public Parent2, private Parent1
{
    public()
    {
        child()
        {
            cout<<"In child def cons";
        }
};

int main()
{
    child obj;
}
```

O/P
Parent 2
Parent 1
child

* Sequence of execution

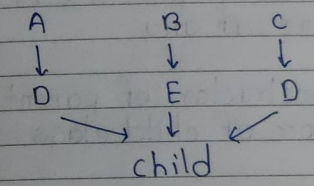
1)




Sequence constructor execution.

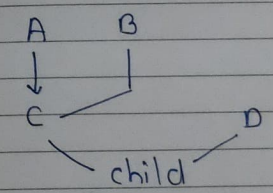
Parent 1
Parent 2
Parent 3
child

2)



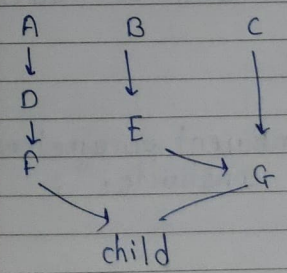

A
D
B
E
C
D
child

3)




A
B
C
D
child

4)




A
D
F
B
E
C
G
child

~~To pass parameter~~

* Parameterised Constructors ~~in Herita~~ in inheritance.

To pass parameters to parent class constructor you have to call parent class constructor using child class constructor.

To call constructor you have to mention the parent class name with parenthesis () containing parameters if any.

You can call any constructor of parent class from any constructor of child class.

```
class Parent
{
    public :
        Parent()
        {
            cout<<"In Parent def constr ";
        }
        Parent(int a)
        {
            cout<<"InParent parameterized
                        constructor";
        }
};
```

```
class child : public Parent
{
    public :
        child() : Parent(5)
        {
            cout<<"\n child def constructructor";
        }

        child(int x)
        {
            cout<<" Child parameterized constru";
        }
};
int main()
{
    child obj;
}
```

Which values can be passed as a parameter to parent class constructor

1) Any constant value ex 3.5, 10

2) Any variable from the formal parameter list of child class constructor.

Remember :- You can't pass datamember of child class as a parameter to parent class constructor if you pass, a garbage value will be passed

```
child (int x, int y) : Parent (a)
{
    a = x ;
    b = y ;
    cout<<"child parameter con";
}
```

In Multiple inheritance parent class constructor get executed in the sequence of inheritance no matter in what sequence you have called their constructor at child class constructor

```
class Parent 1
{
    public:
        int p ;
        Parent1()
        {
            cout<<"Parent 1 def constructor";
        }

        Parent 1 (int x)
        {
            p = x ;
            cout<<"Parent 1 parameter p="<<p;
        }
};
```

```
class Parent 2
{
    public :
        int p ;
        parent 2()
        {

        }
        parent 2 (int x)
        {
            cout<<"Parent 2 parameter p="<<p;
        }
};
class Child : public Parent 1 ; public Parent 2
{
    public :
        int a, b ;
        child()
        {
            a = 0 ;
            b = 0 ;
            cout<<"child def con";
        }
        child (int x, int y) : Parent 2 (y), parent 1 (x);
        {
            a = x ;
            b = y ;
            cout<<"child parameter cons";
        }
};
```

```
child (int x, int y) : Parent (a)
{
    a = x ;
    b = y ;
    cout<<"child parameter con";
}
```

In Multiple inheritance parent class constructor get executed in the sequence of inheritance no matter in what sequence you have called their constructor at child class constructor

```
class Parent 1
{
    public:
        int p ;
        Parent1()
        {
            cout<<"Parent 1 def constructor";
        }

        Parent 1 (int x)
        {
            p = x ;
            cout<<"Parent 1 parameter p="<<p
        }
};
```

```
class Parent 2
{
    public :
        int p ;
        parent 2()
        {

        }
        parent 2 (int x)
        {
            cout<<"Parent 2 parameter p="<<p;
        }
};
class child : public Parent 1 ; public Parent 2
{
    public :
        int a, b;
        child()
        {
            a = 0 ;
            b = 0 ;
            cout<<"child def con";
        }
        child (int x, int y) : Parent 2 (y), parent 1 (x);
        {
            a = x ;
            b = y ;
            cout<<"child parameter cons";
        }
};
```

```
int main()
{
    child obj (3,6)
}
```

* Execution of constructor in case of virtual Parent classes.

1) In case of virtual parent classes constructor of virtual parent will get executed first then non virtual parent classes will get executed

2) If there are multiple virtual parent, there constructor will be executed in the sequence of inheritance then constructor will be executed for all non-virtual parent in sequence of their inheritance.

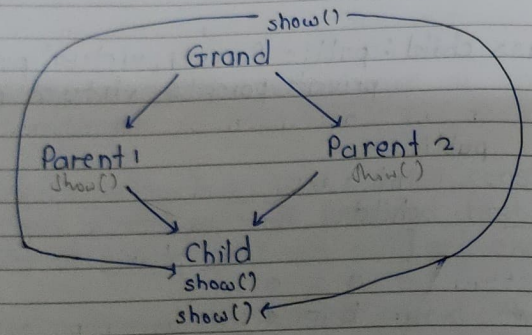
```
class Parent1
{
    public:
        int p;
        parent1()
        {
            cout<<"Parent1 def cons";
        }

        Parent1(int x)
        {
            p = x;
            cout<<"Parent1 parameter con p="<<p
        }
};
```

```
class Parent 2
class Parent 3
class Parent 4
```
} Same as Parent 1

```
class child : public Parent1, virtual public parent2,
              private parent3, virtual parent 4
{
    public:
        int a,b;
        child()
        {
            a = 0;
            b = 0;
            cout<<"Child def constructor";
}

int main()
{
    child obj;
}
```

virtual Parent 2,
virtual Parent 4
Parent 1
Parent 3
Child
} Sequence of execution.

## * Diamond Problem.



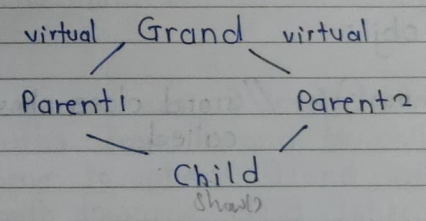

In this example of inheritance Grand class is inherited in child class via Parent 1 and via Parent 2. Hence child class contains two copies of show() fun of class Grand. If we call show using object of child it will generate ambiguity error. To solve this error we can use membership identity label as follows.

```
child obj
obj.show();      // error

obj.Parent1 ::show();
obj.Parent2 ::show();
```
} By using identity label

19/11/2023

Even if we can solve this problem using membership identity label, logically child class should not contain multiple copies of same function. Here child class contains multiple copies of show(). of grand class to avoid these multiple copies, CPP has given us virtual class concept.

By inheriting Grand class using virutal keyword in parent1 and parent2 we can avoid multiple copy problem. When we inherit Grand using virtual.
CPP compiler will take care that only one copy of members of Class Grand will get created in class child even though Grand is inherited in child via multiple paths. eg.



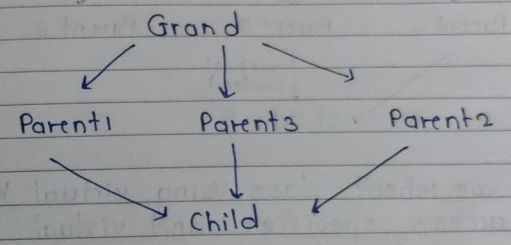

```
class Grand
{
    public:
        void show()
        {
            cout<<"In Grand show()";
        }
};
```

```
class Parent1 : public virtual Grand
{

}

class Parent2 : public virtual Grand
{

};

class Child : public Parent1, public Parent2
{

}
int main()
{
    child obj ;

    obj.show();  //Grand class show() will get
                   called.
}
```

* While using virtual for class you have to follow some rules [we are talking about virtual class not about virtual function]

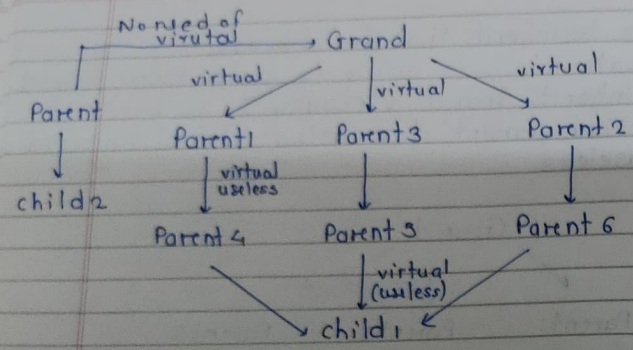
1) We have to inherit that [Grand base class using virtual keyword which is getting inherited via multiple path.

2) We have to inherit that Grand class Every path which is going to child from Grand. must inherit Grand virtually. If you inherit some other parent class using virtual keyword it is useless.




- If you miss to inherit Grand virtually in any path the child will contain multiple copies again.
- If there are multiple levels of inheritance in diff paths from Grand to child then also you have to inherit Grand using virtual, in each path towards child, Inheriting any other parent is useless.

next page diagram.
↓




When we inherit class using virtual keyword then access specifier and virtual keyword can be mentioned in any sequence e.g virtual public grand / public virtual grand

* Inheritance Table :-

| | Private | Protected | Public |
|---|---|---|---|
| Inside Class | ✓ | ✓ | ✓ |
| Inside Derived Class | ✗ | ✓ | ✓ |
| On Object | ✗ | ✗ | ✓ |

21/11/2023

## * Polymorphism.

- Polymorphism means many forms [Poly means many, morphism means forms]

- There are two types of Polymorphism.
  1) Compile time Polymorphism.

  2) Runtime Polymorphism.

- In our program every function call is to connected with correct function defination this job of connecting function call with the correct function defination can be done by compiler or executor. When it is done by compiler we call it compiletime polymorphism and when it is done by executor we call it runtime polymorphism.

- 1) Compiletime Polymorphism

When the job of connecting function call with correct function defination is done by compiler, it is called compiletime polymorphism. To do this, compiler always work with datatypes of variable and not the content of variable

· for example ↓

1) function Overloading
2) Operator Overloading
3) Templete.

```
class Parent
{
    public:
        void show()
        {
            cout<<"In parent show";
        }
};
class Child : public Parent
{
    public:
        void show ()
        {
            cout<<"In Child show()";
        }
};
int main()
{
    Child objchild;
    Parent objparent;

    child *pc;
    Parent *pp;

    pc = &objchild;
    pp = &objparent;
```

```
pc → show();  —— ①
pp → show();  —— ②
}
```

In this program compiler connects the correct function defination at statement ① and ②
To do this compiler checks datatypes of pc and pp and not the conent of pc & pp
at statement ① child class show will be bound because datatype of pc is child
Same thing goes with statement ②

* Rule of Polymorphism -

A pointer of your class can store address of its own object and it can store address of object of all those classes which are directly or indirectly derived from your class

```
Grand    Grand varg;
  ↓
Parent   Parent varp;
  ↓
child    Child varc;
```

∴ Grand *ptr = &varg; ✓
ptr = &varp; ✓
ptr = &varc; ✓

```
Parent *p;
p = &varp; ✓        p = &varg; ✗ → not possible
p = &varc; ✓
```

Class is same as back side page.

```
int main()
{
    Child objchild;
    Parent objparent;

    Parent *pp;

    pp = &objparent;
    pp → show();  —— ①  Parent show()

    pp = &objchild;
    pp → show();  —— ②  child show()
}
```

In above example compiler will decide the function defination which is to be bound at statement ① and statement ②
Compiler will decide this by looking at datatype of pp hence at both ① and ② show() of parent will get called.

2) Runtime Polymorphism :-

* function Overriding

It means defining same function multiple times with same header [do not change the return type also]. To implement function overriding we have to write definations of overridden function in different classes and those classes must implement multilevel inheritance in themselves.

```
class Parent
{
    public :
        void show()
        {
            cout<<"In Parent show()";
        }
};

class child : public Parent
{
    public :
        void show()
        {
            cout<<"Inchild show()";
        }
};
```

* Runtime Polymorphism :-

- Requirement for runtime Polymorphism.

1) Multilevel Inheritance.
2) Over-ridden function.
3) Copy of over-ridden function in topmost base class must be virtual.
4) Pointer to Topmost base class.

all

- In Runtime polymorphism the job of connecting function call with correct function defination will be done by executor. To do this executor will check content of variable and not the datatype of variable.

When executors decides the binding we call it, Runtime Polymorphism.

To implement Runtime Polymorphism we need following things :-

```
class Parent
{
    public:
        virtual void show()
        {
            cout<< "In parent show()";
        }
};

class child : Public Parent
{
    public:
        void show()
        {
            cout<< "In child show()";
        }   Parent :: show()
};

int main()
{
    child objchild;
    Parent objparent;

    Parent *pp;

    pp = &objparent;
    pp -> show();  ——— ①

    pp = &objchild;
    pp -> show();  ——— ②
```

In above example we have implemented all the requirement of runtime polymorphism. Hence at ① and ② executor will decide binding of function by looking at content of pointer pp. At statement ① pp contains address of obj of Parent hence parent class show() will execute. At ② pp contains address of obj of child hence show of child will execute at ②

22/11/23

## * Abstract class.

- When we define pure virtual function inside a class, that class is called Abstract class.

- Pure virtual function -
  - When we assign 0 to declaration of function which is prefixed by virtual keyword we call that function a pure virtual function
  - Pure virtual function do not have any body.
  - You compulsary have to assign 0 to pure virtual function. no other no. is allowed.

- We cannot create object of abstract class. [~~object~~ abstract class.

- Abstract class can contain Normal member function, data members also, but we still can't create object.

- We can use abstract class only for inheritance purpose.

- The Derived class which inherit abstract class must have to define all pure virtual function from abstract class. If derived class does not define pure virtual function from abstract class the derived class also becomes abstract class.

- We use this concept mostly to implement restrictions on derived classes.

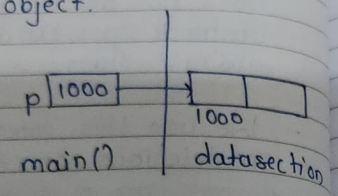
Vehicle

Jeep  bicycle  aeroplane

All jeep, bicycle and aeroplane has their own start methods. if we define start() in vehicle with cout<<"Vehical will start"; and No start functions in derived class then all time vehicle start() will get call which will not specify specific start method so we use abstract class so that all dervied classes have to define their own start method otherwise their obj will not get created.

```
class Vehical
{
    public:
        virtual void start()=0;
}
class Jeep : public vehical
{
    public:
        void start()
        {
            cout<<"Botton start";
        }
```

Q * Write a program to explain constructor and Destructor for Runtime object.

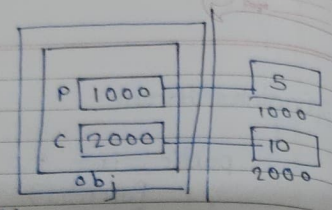



```
class Number
{
    public:
        int x, y;

        Number ()
        {
            x=0, y=0;
            cout<<"In constructor x = "<<x;
        }

        ~Number ()
        {   x++;
            cout<<"In destructor x = "<<x;
        }
};
int main()
{
    Number *p;
    p = new Number[2]; or new number ();

    delete []p;     or delete p;
}
```

* When destructor of any class gets executed destructor of all its parent classes will get executed automatically.

23/11/23

```
* class Parent
  {
      public: int *p;
              Parent ()
              {
                  cout<<"In Parent def constr";
                  p = new int (5);
              }

              ~Parent()
              {
                  cout<<"In Parent Destructor";
                  delete p;
              }
  };
  class Child : public Parent
  {
      public:
              int *c;
              Child ()
              {
                  cout<<"In child def constr";
                  c = new int (10);
              }

              ~Child ()
              {
                  cout<<"In child destructor";
                  delete c;
              }
  };
```



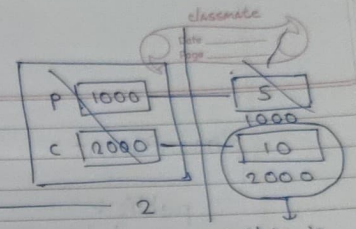

```
int main()
{
    Child *ptr;
    ptr = new child ()
    delete ptr;
}
```

1st main function →

In this main function when compiler compiles statement ① it checks datatype of ptr i.e child to map destructor with this statement. Compiler maps destructor of class child at statement ① hence when executor executes statement ① destructor of class child and then Parent will get executed.
Hence there will be no any memory leak in execution of program.

Output
In Parent def constructor.
In child def constructor
In child destructor
In parent destructor

2nd main() function
↓ next page.




```
int main()
{
    Parent *ptr;    //———— 2
    ptr = new child();
    delete ptr;     //———— 1
}
```

this is memory leakage.

In this main(), parent class pointer ptr will store address of child class obj at statement ①
compiler will map destructor by looking at the datatype of ptr, Hence only parent class destructor ~~is not~~ will get executed at ①. As child class destructor is not executing at ①. Memory at 2000 will not get destroyed, resulting into memory leak. This happen because, destructor is mapped by compiler

```
class Parent
{
    public:
        int *p;
        Parent()
        {
            cout<<"In Parent def constr";
            p = new int(5);
        }
        virtual ~Parent()
        {
            cout<<"In Parent Destructor";
            delete p;
        }
};
class Child : public Parent
{
    public:
        int *c;
        Child()
        {
            cout<<"In child def constructor";
            c = new int(10);
        }
        ~Child()
        {
            cout<<"In child destructor";
            delete c;
        }
};
```

```
int main()
{
    Parent *ptr;
    ptr = new Child();
    delete ptr; ——— ③
}
```

Child and Parent constructor name will be considered as same due to " is a " relation

Output
Parent constructor
Child constructor
Child destructor
Parent destructor

Due to virtual destructor of Parent, at ③ destructor of child is mapped by looking at content at ptr.

26/11/23

* template :-

- We use templates to define generic functions. Generic functions are the function which can work with any datatype to define such function we have to use templates
for example - Following program prints a integer value.
without template

```
void print (int a)
{
    cout<<"a =">>a;
}
void double print (double a)
{
    cout<<"a ="<<a;
}
void print (char a)
{
    cout<<"a ="<<a;
}
int main()
{
    print (2);
    print (3.2);
    print ('A');

}
```

In above program we have used function overloading to work with different datatype. So we have defined same function 3 times so to avoid this repeatative defination. So we will use template so that only one function, so that only one function.

* How to use template.

To do this we have to define our function with template declaration

Syntax - template<class T>

Here T is called template parameter which will work as a datatype in templetized function

* How to define template function.

```
template<class T>
void print (Ta, Tb)
{
    cout<<"a ="<<a<<" b">>b;
}
int main()
{
    print (2, 4);
    print (2.2, 3.2);
    print ('A', 'B');
}
```

In above example 1 defination of print() will work for all three function call in main().

* How does template works.

In case of templates compiler decides the substitute datatype for T. ~~com~~.
compiler can decide this datatype by their own. by looking at the datatype of actual datatype of ~~viraible~~ actual parameters. of programer can mention explicitly the substitute datatype for T. once compiler gets substitute datatype for T, compiler creates a seperate copy of template functions for each decided datatype and maps thats copy with function call. As compiler does this mapping we call template as an example of compiletime polymorphism.

```
~~cla~~ template <class T>
void add (Ta, Tb)
{
    T ans;
    cout<<" Addition ="<<ans;
}
int main()
{
    print (2, 3);
    print (2.3, 4.5);
    print (4.6 f, 5.6 f);
}
```

* How to mention datatype ofor template parameter

To mention datatype explicitly for template parameter we use angular Bractes '< >' containg datatype between name of function and parathesis of function call.

```
template <class T>
void print (Ta, Tb)
{
    cout<< a;
    cout<<b;
}
int main()
{
    print<int> (3, 2.5);
    print<float> (3.2f, 1.5);
    print<char> ('A', 'B');
}
```

* template declaration for each function.

One template declaration can work for only one function if you want to make multiple functions Generic then you have to right ~~multiple~~ template declaration for each generic function
for

```
template<class T>          ——— ①
void print(Ta, Tb)
{
    cout<<a<<b;
}
```

```
template <class T>
void disp (T x, T y)
{
    cout<<x<<y;
}
int main()
{
    print (2,3);
    print (2.3, 3.2);
    print ('A','B');
    disp (1, 2);

}
```

In above example if we skill statement ② compiler will generate error for t at disp () function. template declaration at statement ① can work for print() function only it will not work for disp() function.

* Multiple template paramete rules

-1) Compiler can decide substituet datatype for template parameters only if we have use template parameter at formal parameter.
if we don't compiler cannot find substitute datatype for template parameter

2) Compiler must get substitute datatype for each template parameter other wise compiler will generates error, substituation fails.

```
template <class T, class V>  ——— ①
void print (T a, V b)  ——— ②
{
    cout<< a ;
    cout <<b ;
}
int main()
{
    print (10 ,4);
    print (3.2f, 1.5);

}
```

In this example if we do not use V as a datatype at function defination parameter list [statement ② ] compiler cannot decide ~~datatype~~ substitute datatype for V and will generate error

~~In this case~~
~~In this case~~ To avoid this error we can explicitly mention substitute for each template parameter by using angular bracket '<>' syntax at function call

```
template <class T, class V>
void print (T a, T b)
{
    cout<< a ;
    cout<<b;
}
```

```
int main()
{
    print<int, int>(10, 4);  ——— ①

    print<float, double>(3.2f, 1.5);  ——— ②

    print('A','B');  ——— ③
}
```

In above example we have not used template parameter V as a datatype in print() function parameter list Hence compiler cannot find substitute V on its own, Hence statement ③ will generate error, substitution fails. But in case of statement ① and ② we have explicitly mentioned substitute datatype for T and V, Hence ① and ② will get compile and run successfully.

* If compiler finds exact match and templatized function match for a function call, then priority will be given to templetized function. If their is no any exact match found then priority will be given to templetized function over typecasting.

```
void print(int a, int b)
{
    cout<<"In int fun";
    cout<<a<<b;
}
```

```
template<class T>
void print(Ta, Tb)
{
    cout<<"In template fun";
    cout<<a<<b;
}
int main()
{
    print(10,2);        ——— ①
    print(3.2f, 1.5f);  ——— ①
    print('A','B');     ——— ③
}
```

Output
```
In int fun
a=10 , b=2
In template function.
a=3.2  b=1.5
In template function
a=A  b=B
```

Q Sort 8 integer Array in Assending Order

```
int main()
{
    int num[8];
    int i;

    for (i=0 ; i<8 ; i++)
    {
        cin>>num[i];
    }

    sort (num, 8);

    cout<<"After sorting";
    for (i=0 ; i<8 ; i++)
    {
        cout<<num[i];
    }
}
void sort(int *p, int n)
{
    int i, j, temp;
    for (i=0; i<n; i++)
    {
        for (j=i+1 ; j<n ; j++)
        {
            if(*(p+i) > *(p+j))
            {
                temp = *(p+i);
                *(p+i) = *(p+j);
                *(p+j) = temp;
            }
        }
    }
}
```

* With use of template which will work for all datatype)

```
template <class T>
void start ( T p[], int n)
{   T temp;
    int i, j, ~~temp~~;
    for(i=0; i<n; i++)
    {
        for(j=0 ; j<n ; j++)
        {
            if( p[i] > p[j])
            {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

template <class T>
void print( T p[], int n)
{
    Int i;
    for(i=0; i<n ; i++)
    {
```

```
        cout << p[i];
    }
}
int main()
{
    int num[8] = {10, 20, 30, 60, 2, 4, 3, 1};
    sort (num, 8);
    print (num, 8);

    float num2 [5] = {1.1, 3.2, 2.2, 8.5, 1.2};
    sort (num2, 8);
    print (num2, 8);

}
```

In this example one function is defined which will work for all datatypes because of use of template thats why our function become generic.

But when we pass user defined to generic function we have to over load it that is explained in ~~above~~ below example

→

next page.

```
class Distance
{
    public:
        int mtr;
        friend ostream& operator<< (ostream& dout, );
};
ostream& operator<< (ostream& dout, Distance Temp)
{
    cout << Temp.mtr;
    return cout;
}
template <class T>
void print (T a)
{
    cout << "\n Data = " << a;
}

int main()
{
    Distance d1, d2
    d1.mtr = 200;
    d2.mtr = 300;

    print (d1);
    print (d2);
}
```

Output  Data = 200
        Data = 300.

## * template classes

To create generic classes we have to use template for class. while creating a object of templatized class we have to mention substitute datatype in angular '<>' if we don't compiler generates error. example.

```
template<class T>
class Number
{
  public:
      T x;
      void set(T a)
      {
          x = a;
      }
      void print()
      {
          cout<<"\n x =">>x;
      }
};
int main()
{
    Number <int> T1;
    Number <double> T2;

    T1.set(10);
    T1.print();
    T2.set(3.5);
    T2.print();
}
```

Output $x = 10$
$x = 3.5$

## * How to define member function of template class, outside the class.

```
template<class T>
class Number
{
    public:
        T x;
        void set(T a);
        void print()
        {
            cout<<"\n x =">>x;
        }
};
template<class T>
void Number<T>::set(T,a)
{
    x = a;
}

int main()
{
    Number<int>T1;
    Number<double>T2;

    T1.set(10);
    T2.print();

    T2.set(3.5)
    T2.print();
}
```

```
class Distance
{
    public:
        int mtr

        Distance()
        {
            cout<<"Enter mtr=";
            cin>>mtr;
        }
};
int main()
{
    int num[5]={23,5,41,2,3};
    print(num,5);
    sort(num,5);
    print(num,5);

    Distance dist[5];
    print(dist,5);
    sort(dist,5);
    print(dist,5);
}

template<class T>
void sort(T p[],int n)
{
    int i,j;
    T temp;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(p[i]>p[j])
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
            }
        }
    }
}
template<class T>
void print(T p[],int n)
{
    for(int i=0;i<n;i++)
    {
        cout<<p[i];
    }
}

ostream& operator<<(Ostream &out, Distance Temp)
{
        cout<<Temp.mtr;

        return cout;
}

int operator>(Distance T1, Distance T2)
{
        if(T1.mtr > T2.mtr)
```

```
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}
```

or

```
return mtr > Temp.mtr;
```

another way of defining class Distance

```
class Distance
{
    public :
        int mtr;
        int operator> (Distance Temp)
        {
            return mtr> Temp.mtr;
        }

        friend ostream & opprator<<(ostream& dout,
                                    Distance temp)
        {
            cout<<Temp.mtr;

            return cout;
        }
};
```

27/11/2023

* Delete element

```
int main()
{
    int n, i, ind, x;
    int *p = new int[n];

    cout<<"Enter no. of elements =";
    cin>>n;
    p = new int[n];

    for(i=0 ; i<n; i++)
    {
        cin>> *(p+i);
    }
    ~~ind =~~
    cout<<"Enter x to search";
    cin >>x;

    ind = search (x,p,n);

    if(ind==-1)
    {
        cout<<" Element Not Found";
        return 0;
    }

    ~~for(i=ind; i<n ; i++)~~
    {
        ~~cout<<" "<<*(p+i);~~
    }
}
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 8 | 10 | 5 | 10 | 13 |

1000

```
int search (int *p, int x, int n)
{
    for(int i=0; i

    for(i=ind; i<=n-2 ; i++)
    {
        *(p+i) = *(p+i+1);
    }

    for(i=0 ; i<n ; i++)
    {
        cout<<"  " << *(p+i);
    }
};

int search (int x, int *p, int n)
{
    for(i=0 ; i<n ; i++)
    {
        if(*(p+i)==x)
        {
            return 1;
        }
    }
    return -1;
}
```

Q Compare city names Assignment 14 Question 7

```
int main()
{
    string city[6];
    for(i=0; i<6 ; i++)
    {
        cout<<"Enter city Name =";
        getline(cin,city[i]);
        for(j=0 ; j<=i-1 ; j++)
        {
            if(city[i]==city[j])
            {
                cout<<"Already present";
                i--;
                break;
            }
        }
    }

    for(i=0 ; i<6 ; i++)
    {
        cout<<"city = "<<city[i];
    }
}
```

Q

```
template<class T>
void (T a, T b)
{
    cout<<"Addition =">>a+b;
}
int main()
{
    add (2,2);
    add (3.2f,4.2f);
}
```

Q

```
template<class T>
class Number
{
    public:
        T x, T y;

        Number (T a, T b)
        {
            x = a;
            y = b;
        }
        void show()
        {
            cout<<x;
            cout<<y;
        }
};
```

```
int main()
{
    Number <int> T1(2,3);
    Number <float> T2 (2.3f,2.2f);
    T1.show();
    T2.show();
}
```

28/11/2023

STL [standard

- C++ has define some classes to store data and has defined some functions to access this data. C++ has done this so that we can write time and space efficient codes, using those classes and functions.

- STL is Divided into 3 parts.
  1) Container
  2) Iterator
  3) Algorithm

1) Container :-

This are some inbuilt classes which can be used to store data.
There are 3 types of container.

1) Sequence container :-
   1) Vector
   2) List
   3) deque.

2) Associative container :-
   1) Set
   2) Map
   3) Multimap.

3) Derived container :-
   1) stack
   2) que
   iii) priority-queue

2) Iterator :-

This are also classes. we use object of iterator to access elements of container basically we will treat iterators as a pointer while accessing elements of container.

3) Algorith :-

This is set of predefined function which work with elements of container using iterator for example sorting in data in containr, searching in of data in containor, insertion, deletion of data in containor.

* Sequence container.

1) vector :-

- C++ has implimented vector class using runtime array. when we create a vector, it internally creates an array of some elements. after filling all those elements, if we need more memory vector performs reallocation of memory. If we need more memory then vector will create memory with more elements and copy elements of previous memory in new memory will continue the operation

e.g

```
vector<int> vec;
```