Object Oriented Programming
C++ Class and Object

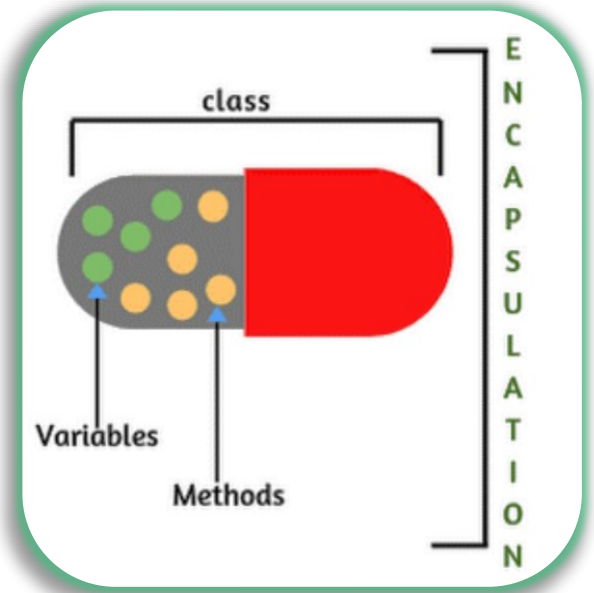
CS(217) Object Oriented Programming

Abeeda Akram

Object Oriented Programming

Implementation of **Abstract Data Type (ADT)**

1. Select a concrete data representation **using built-in data types**
2. Implement all relevant functions

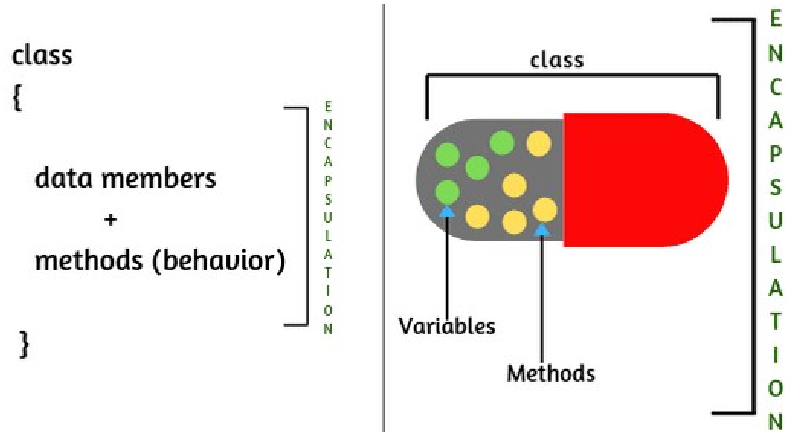


C++ Class (class is reserve word in C++)

- Class is used to only ***define*** new data types.
- It is collection of
  - Data called **data members** or **attributes**
  - Functions called **member functions**, **methods** or **behaviors**

C++ Class

```
class class-name
{
   //declaration statements here
   //data members defined only not initialized
   //add member functions prototype or complete implementation
};
   // Class is simply definition no memory is reserved
```



```
class Point
{
    int x;
    int y;
};
```

```
class myTime
{
    int sec;
    int min;
    int hour;
};
```

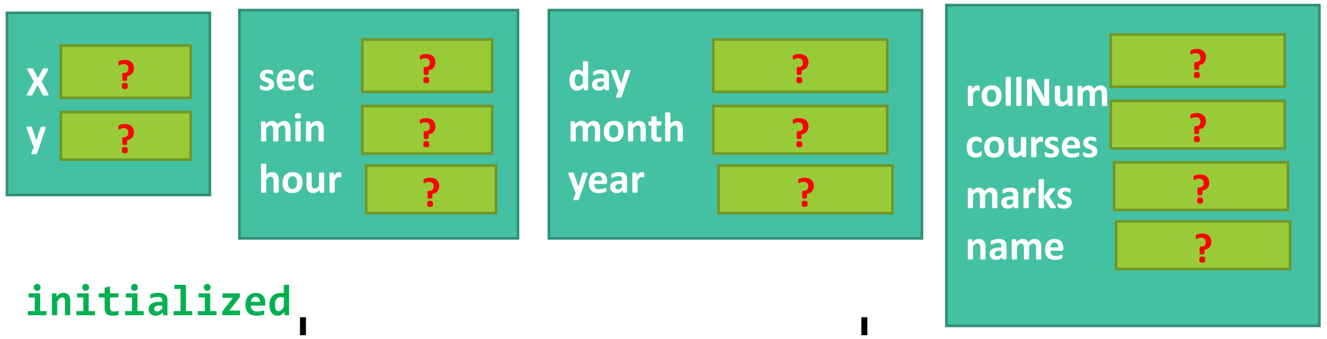
```
class myDate
{
    int day;
    int month;
    int year;
};
```

```
class Student
{
    int rollNum;
    int courses;
    float marks;
    char name[20];
};
```

Creating Class Objects

- Objects are variables of class
  - **Separate** data members memory is allocated only when object is created
  - For member functions only **one copy** is created that is used by all objects

```
void main(){
  Point p;
  myTime t;
  myDate d;
  Student s;
} //Objects created but not initialized
```



```
class Point
{
    int x;
    int y;
};
```

```
class myTime
{
    int sec;
    int min;
    int hour;
};
```

```
class myDate
{
    int day;
    int month;
    int year;
};
```

```
class Student
{
    int rollNum;
    int courses;
    float marks;
    char name[20];
};
```

Class Member access specifiers

`private:` (reserve word in C++)

- Class members accessible only to member functions of class
- Not accessible outside class (user defined functions)

`public:` (reserve word in C++)

- Class members accessible to member functions of class
- Also accessible outside class (user defined functions)

`protected:` (reserve word in C++)

- Class members Accessible to member functions and derived classes (will use and discuss later on)

By default class member access is private, if no access specifier is mentioned

```
class Point
{
    private:
        int x;
        int y;
};
```

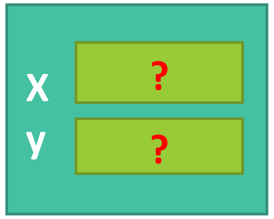
```
class Point
{
    public:
        int x;
        int y;
};
```

```
class Point
{
        int x;
        int y;
};
```

Object Member access Operator (.)

- Class members are accessed outside class by **name** using **dot operator**
- Only if access specifier is **public**.

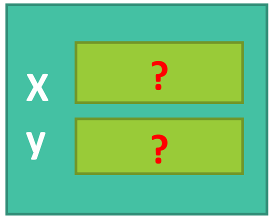
```
void main(){
    Point p;

    cout << p.x;
    //object variable name dot member name
    cout << p.y;
    //object variable name dot member name

    p.x = 100;
    cin>>p.y;
    //initialize members
}
```



```
class Point
{
    public:
        int x;
        int y;
};
```

Object Member access Operator (.)

- Class members are accessed outside by **name** using **dot operator**
- Only if access specifier is **public**.

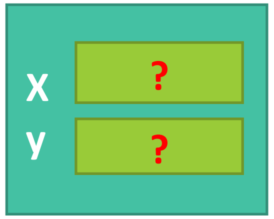


```
void main(){
    Point p;
    cout << p.x;
    //Compiler Error cannot access private member
    outside
    cout << p.y;
    //Compiler Error cannot access private member
    outside

    p.x = 100;
    cin>>p.y;
    //Compiler Error cannot access private member
    outside
}
```

```
class Point
{
  private:
     int x;
     int y;
};
```

Object Member access Operator (.)

- Class members are accessed outside by **name** using **dot operator**
- Only if access specifier is **public**.



```
void main(){
    Point p;
    cout << p.x;
    //Compiler Error cannot access private member outside
    cout << p.y;
    //Compiler Error cannot access private member outside

    p.x = 100;
    cin>>p.y;
    //Compiler Error cannot access private member outside
}
```

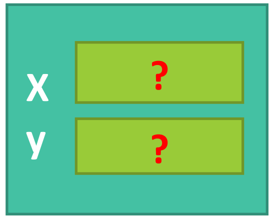
```
class Point
{
        int x;
        int y;
};
```

Object Member access Operator (->)

Dynamic objects and Pointers

- Class members are accessed outside by **name** using **arrow operator**
- Only if access specifier is **public**.

```
void main(){
    Point * p = new Point;
    //Allocate memory

    //Dereference pointer dot member name
    cout << (*p).x ;

    //pointer name arrow member name
    cout << p->y;
    p->x = 100;
    cin >> p->y;

    delete p; //Deallocate memory
}
```



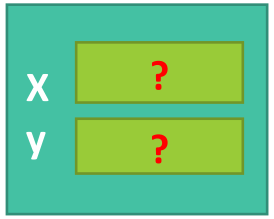
```
class Point
{
    public:
        int x;
        int y;
};
```

Object Member access Operator (->)

Dynamic objects and Pointers

- Class members are accessed outside by **name** using **arrow operator**
- Only if access specifier is **public**.

```
void main(){
    Point * p = new Point;
    //Allocate memory

    //Compiler Error cannot access private members
    cout << (*p).x ;
    cout << p->y;
    p->x = 100;
    cin >> p->y;

    delete p; //Deallocate memory
}
```

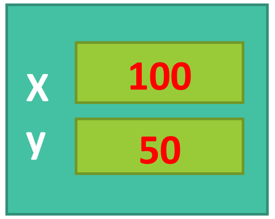


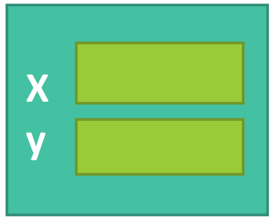
```
class Point
{
   private:
      int x;
      int y;
};
```

Object Assignment Operator (=)

- Member wise assignment of data.
- Only if member access specifier is **public**.

```
void main(){
    Point p;
    p.x = 100;  p.y = 50;

    Point p2;
    p2.x = p.x;
    p2.y = p.y;
    //Member wise data assignment

    //Aggregate data assignment
    p2 = p; //Same as member wise data assignment

}
```





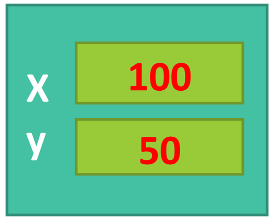
```
class Point
{
    public:
        int x;
        int y;
};
```

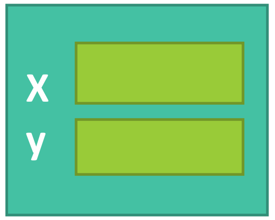
Object Assignment Operator (=)

Dynamic objects and Pointers

- Member wise assignment of data.
- Only if member access specifier is **public**.

```
void main(){
    Point p;
    p.x = 100;  p.y = 50;

    Point * p2 = new Point;
    p2->x = p.x;
    p2->y = p.y;
    //Member wise data assignment

    //Aggregate data assignment
    *(p2) = p; //Same as member wise data assignment
    delete p2; //Deallocate memory
}
```



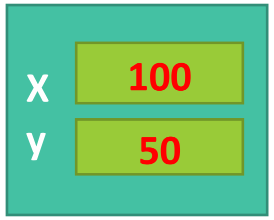


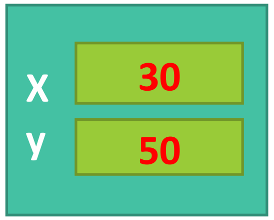
```
class Point
{
    public:
        int x;
        int y;
};
```

Object Relational Operators (==, !=, <=, >=, <, >)

- Always compare data member wise.
- Only if member access specifier is **public**.

```
void main(){
    Point p;
    p.x = 100;  p.y = 50;

    Point * p2 = new Point;
    p2->x = 30; p2->y = 50;

    cout << p2->x != p.x;
    cout << p2->y < p.y;
    //Compare member wise

    cout << *(p2) == p;
    //Compile time error Operation not defined
    delete p2; //Deallocate memory
}
```



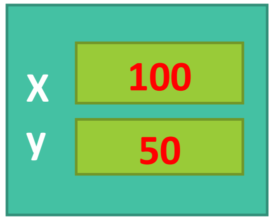


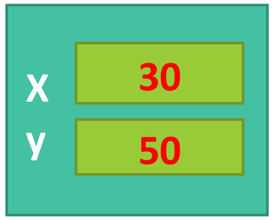
```
class Point
{
    public:
        int x;
        int y;
};
```

Object Arithmetic Operators (+,-, /, *, %)

- Operations depends on data members built in data type operation.
- Only if member access specifier is **public**.

```
void main(){
    Point p; p.x = 100; p.y = 50;

    Point * p2 = new Point;
    p2->x = 30; p2->y = 50;

    p.x = p2->x + 100;
    p2->y = p2->y + p.y;
    //member wise

    p = *(p2) + p;
    //Compile time error Operation not defined
    delete p2; //Deallocate memory
}
```



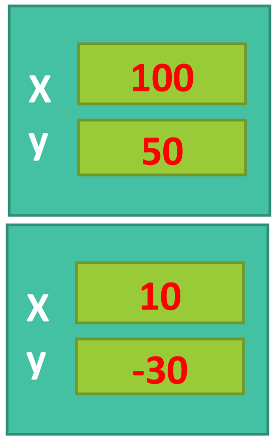


```
class Point
{
    public:
        int x;
        int y;
};
```

Objects and functions (Pass by value)

- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point p1; p1.x = 100; p1.y = 50;
    Point p2; p2.x = 10;  p2.y = -30;

    cout << equal (p1, p2);
}

//objects pass by value or copy
bool equal(Point p, Point q){
    if((p.x == q.x)&&(p.y == q.y))
        return true;
    else
        return false;
}
```

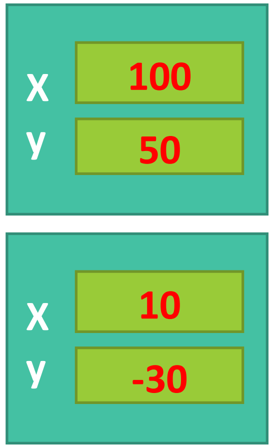


```
class Point
{
    public:
        int x;
        int y;
};
```

Objects and functions (Pass by Reference)

- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point p1; p1.x = 100; p1.y = 50;
    Point p2; p2.x = 10;  p2.y = -30;

    update(p1, p2);
}

//object pass by reference
void update(Point & p, Point q){
     p.x  = p.x + q.x;
     p.y  = p.y + q.y;
}
```

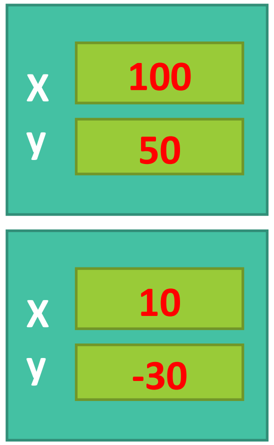


```
class Point
{
   public:
      int x;
      int y;
};
```

Object and functions (Return by value)

- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point p1; p1.x = 100; p1.y = 50;
    Point p2; p2.x = 10;  p2.y = -30;

    Point n = create(p1, p2);
}

//returns object’s copy
Point create(Point p, Point q){
    Point n;
    n.x = p.x + q.x;
    n.y = p.y + q.y;

    return n;
}
```

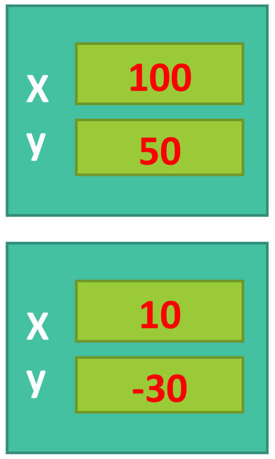


```
class Point
{
    public:
        int x;
        int y;
};
```

Dynamic Objects and functions

- Pass objects pointer in functions by **value**
- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point * p1 = new Point;
    p1->x = 100; p1->y = 50;
    Point * p2 = new Point;
    p2->x = 10; p2->y = -30;

    cout << equal (p1, p2);
    delete p1; delete p2;
}
//Object pointer pass by value or copy
bool equal(Point * p, Point * q){
    if((p->x == q->x)&&(p->y == q->y))
        return true;
    else
        return false;
}
```



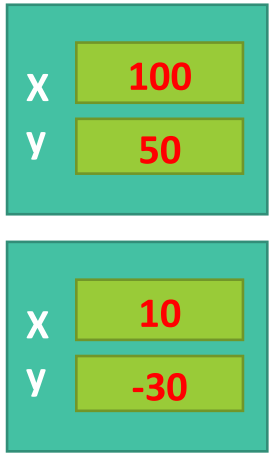
```
class Point
{
    public:
        int x;
        int y;
};
```

```
//Object's data is always
pass by reference through
pointers
```

Dynamic Objects and functions

- Pass objects pointer in functions by **value**
- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point * p1 = new Point;
    p1->x = 100; p1->y = 50;
    Point * p2 = new Point;
    p2->x = 10; p2->y = -30;

    cout << update (p1, p2);
    delete p1; delete p2;
}
//Object pointer pass by value or copy
void update(Point * p, Point * q){
     p->x  = p->x + q->x;
     p->y  = p->y + q->y;
}
```



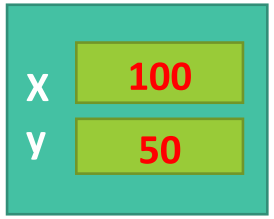
```
class Point
{
   public:
      int x;
      int y;
};
```

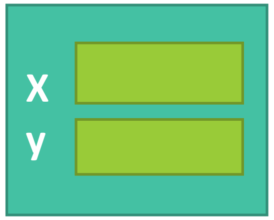
```
//Object's data is always
pass by reference through
pointers
```

Dynamic Objects and functions

- Pass objects pointer in functions by **reference**
- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point * p1 = new Point;
    p1->x = 100; p1->y = 50;

    Point * p2;
    createCopy(p2, p1);

    delete p1; delete p2;
}
// Object pointer pass by reference
void createCopy(Point *& n, const Point * q){
    n = new Point;
    n->x = q->x + 100;
    n->y = q->y + 50;
}
```





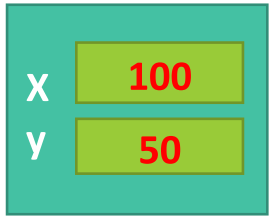
```
class Point
{
    public:
        int x;
        int y;
};
```

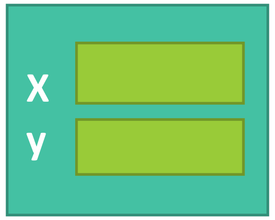
```
//Object's data is always
pass by reference through
pointers
```

Dynamic Objects and functions

- Return objects pointer from function
- Functions can access class members only if member access specifier is **public**.

```
void main(){
    Point * p1 = new Point;
    p1->x = 100; p1->y = 50;

    Point * p2 = createCopy(p1);

    delete p1; delete p2;
}
// Object pointer return from function
Point * createCopy(const Point * q){
    Point * n = new Point;
    n->x = q->x + 100;
    n->y = q->y + 50;
    return n;
}
```





```
class Point
{
    public:
        int x;
        int y;
};
```

//Object's data is always pass by reference through pointers