

# THE BIG FOUR, OPERATOR OVERLOADING

---

Problem Solving with Computers-II



Read the syllabus. Know what's required. Know how to get help.

## CLICKERS OUT

## Constructor and De-constructor

**The most profound moments in the life of an object are its birth and death.**

**— Anonymous**

# Constructor and De-constructor

Every class has the following special methods

- **Constructor:** Invoked right AFTER new objects are created in memory
- **De-constructor:** Invoked right BEFORE an object is deleted from memory

The compiler automatically generates default versions

# Constructor

```
void foo(){  
    Player p;  
    Player *q = new Player;  
    Player w( "Jill");  
}
```

default constructor  
is called

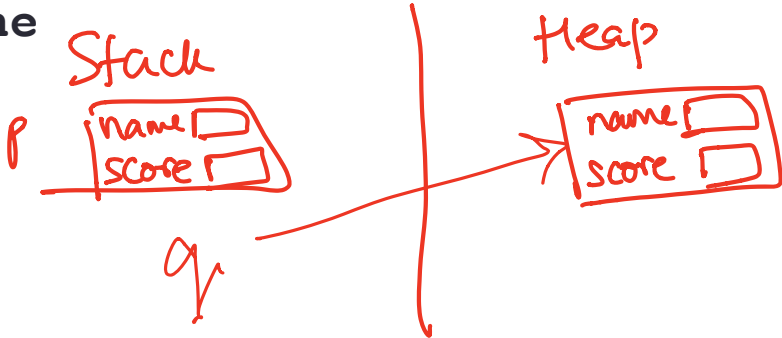
```
1  
2 class Player{  
3 public:  
4     Player();  
5     Player(string playerName);  
6     void setName(string input);  
7     string getName() const;  
8     int playToss();  
9 private:  
10    string name;  
11    int score;  
12  
13 };
```

default

parameterized

How many times is the constructor invoked for the above code?

- A. Never
- B. Once
- C. Twice
- D. Thrice



Destructor: Invoked when an object is removed from memory

```
1
2 class Player{
3 public:
4     Player();
5     ~Player();
6     Player(string playerName);
7     void setName(string input);
8     string getName() const;
9     int playToss();
10 private:
11     string name;
12     int score;
13
14 };
```

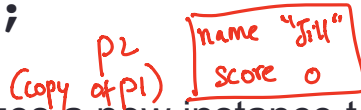
```
void foo(){
    Player p;
    Player *q = new Player;
}
```

The de-constructor of which of the objects is invoked when foo() returns

- ☒ A. p ← p's destructor is called
- B. q
- C. \*q
- D. None of the above

# Copy constructor

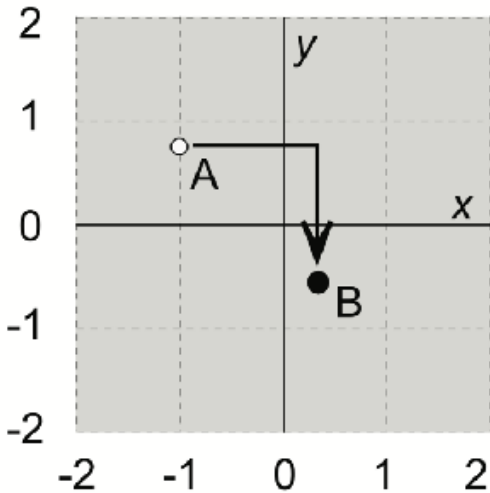
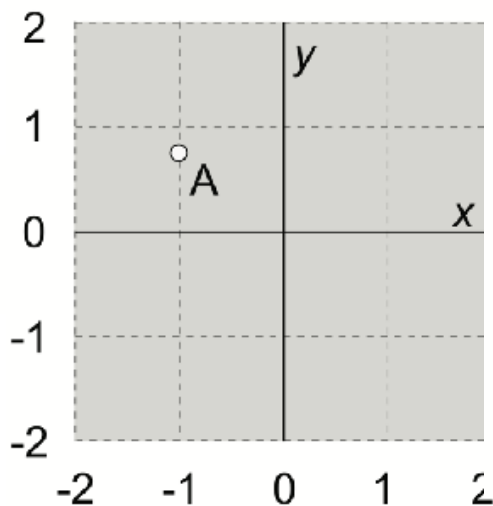
```
Player q1;           //default constructor is invoked
Player p1("Jill");  // Parametrized constructor
//Copy constructor is invoked in all cases below:
Player p2(p1);      Player Object
Player p3 = p1;
Player *p = new Player(p1);
```



- The copy constructor creates and initializes a new instance to be the copy of another instance of the class
- A class always has a default copy constructor that COPIES the values of the input object to the one that is being created

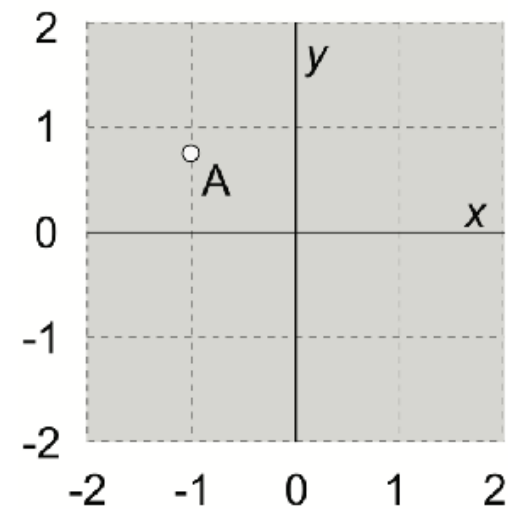
## The point class (Chapter 2, section 2.4)

(a) The white dot labeled A is a point with coordinates  $x = -1.0$  and  $y = 0.8$ .



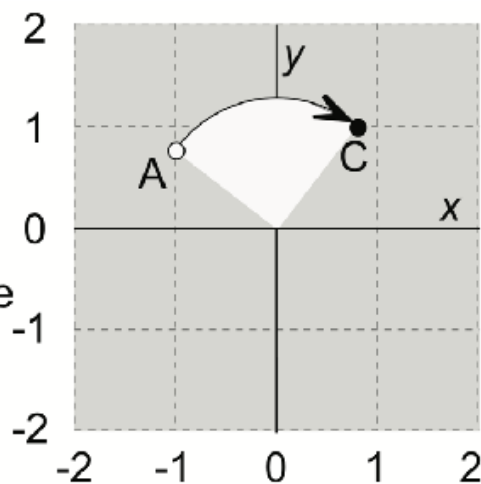
(b) The black dot labeled B was obtained by shifting point A by 1.3 units along the x axis and by  $-1.4$  units along the y axis. The coordinates of point B are  $x = 0.3$  and  $y = -0.6$ .

## The point class (Chapter 2, section 2.4)



(a) The white dot labeled A is a point with coordinates  $x = -1.0$  and  $y = 0.8$ .

(c) The black dot labeled C was obtained by rotating point A  $90^\circ$  in a clockwise direction around the origin. The coordinates of point C are  $x = 0.8$  and  $y = 1.0$ .





# Overloading Binary Comparison Operators

We would like to be able to compare two objects of the class using the following operators

`==`

`!=`

and possibly others

```
double distance(const point & p1, const point &p2){  
    if(p1 == p2)  
        return 0;  
  
}
```

# Overloading Binary Arithmetic Operators

We would like to be able to add two points as follows

```
point p1, p2;  
point p3 = p1 + p2
```

# Overloading input/output stream

- Wouldn't it be convenient if we could do this:

```
point p(10, 10);  
cout<<p;
```

And this....

```
point p;  
cin>>p; //sets the x and y member variables of p based on user input
```

## Copy assignment

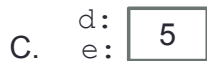
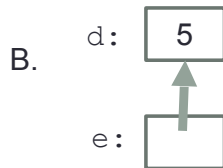
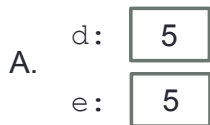
```
Player q;           //default constructor is invoked  
Player p1("Jill"); // Parametrized constructor  
Player p2;  
p2 = p1; // Copy assignment method is invoked
```

- Default behaviour: Member variables of p1 are copied to the members variables of p2

# References in C++

```
int main() {  
    int d = 5;  
    int &e = d;  
}
```

Which diagram below represents the result of the above code?



D. This code causes an error

# References in C++

```
int main() {  
    int d = 5;  
    int &e = d;  
    int f = 10;  
    e = f;  
}
```





How does the diagram change with this code?

A.   
d:   
e: 

f: 

B.   
d: 

e:   
f: 

C.   
d:   
e:   
f: 

D. Other or error

# Passing parameters as references

```
int main() {  
    int d = 5;  
    foo(d);  
    cout<<d;  
}
```

```
void foo(int& e) {  
    e = 10;  
}
```

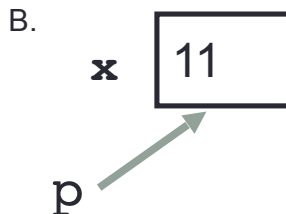
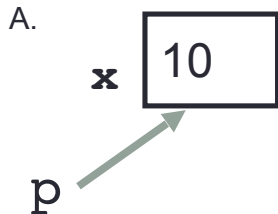
What is the output of this code?

- A. 5
- B. 10
- C. Error
- D. None of the above

# Tracing code involving pointers

```
int *p, x = 10;  
p = &x;  
*p = *p + 1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?



C. Neither, the code is incorrect



# Summary

- ❑ Classes have member variables and member functions (method). An object is a variable where the data type is a class.
- ❑ You should know how to declare a new class type, how to implement its member functions, how to use the class type.
- ❑ Frequently, the member functions of an class type place information in the member variables, or use information that's already in the member variables.
- ❑ New functionality may be added using non-member functions, friend functions, and operator overloading

# Next time

- Linked-lists (Chapter 5)