

# IMPLEMENTING C++ CLASSES: ACCESS SPECIFIERS CONSTRUCTORS

---

Problem Solving with Computers-II

The image shows the C++ logo in a large, blue, 3D-style font. Below the logo is a snippet of C++ code in a monospaced font, with some words highlighted in color. The code is:

```
#include <iostream>
using namespace std;

int main() {
    cout<<"Hola Facebook\n";
    return 0;
}
```

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

## From last lecture...

- Last time we defined a class `Complex` and wrote a main function that created objects of this class
  - We did not implement the member functions of the class.
  - When the code was compiled with `g++`, it resulted in a linker error but when we compiled with the `-c` option, compilation was successful. Why?
- A. The `-c` option suppresses linker errors and produces an executable
- B. The `-c` option does not attempt to link code and no executable is produced
- C. None of the above

In Java:

```
public class DayOfYear {  
    public void setDate(int mon, int day){  
        dd = day;  
        mm = mon  
    }  
    private int dd;  
    private int mm;  
  
}
```

C++, attempt 1:

```
class DayOfYear {  
    public void setDate(int mon, int day);  
    private int dd;  
    private int mm;  
};
```

Which of the following is a problem with the C++ implementation above?

- A. The implementation of the member function setDate should be included in the class
- B. The class DayOfYear should be declared public
- C. The semicolon at the end of the class will cause a compile error
- D. In C++ you specify public and private in regions, not on each variable or function

Which of the following is a problem with the C++ implementation?

- A. In definition of **setDate**, member variables mm and dd should be accessed via objects
- B. Objects declared outside the class cannot access the private member variables
- C. None of the above

C++, attempt 2:

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
    private:
        int dd;
        int mm;

};

void DayOfYear::setDate(int mon, int day){
    mm = mon;
    dd = day;
}

int main(){
    DayOfYear today;
    today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.mm <<"/"<< today.dd;
    return 0;
}
```

How can we make sure that a function doesn't inadvertently change the member variables of the class?

- A. Declare the variables const (as shown)
- B. Declare the function as a const

```
int main() {  
    DayOfYear today;  
    today.setDate(1, 9);  
    cout<<"Today's date is: ";  
    cout<< today.getMonth() <<"/"  
        << today.getDay();  
}
```

```
class DayOfYear {  
  
    public:  
        void setDate(int mon, int day);  
        int getMonth();  
        int getDay();  
    private:  
        const int dd;  
        const int mm;  
};  
void DayOfYear::setDate(int mon, int day)  
    mm = mon;  
    dd = day;  
}  
int DayOfYear::getMonth() {  
    dd = 1;  
    return mm;  
}  
int DayOfYear::getDay() {  
    mm = 12;  
    return dd;  
}
```

How can we make sure that a function doesn't inadvertently change the member variables of the class?

***Declare the function as a const***

***Introduce new terms:***

- ***Accessors (getters)***
- ***Mutators (setters)***

```
int main() {
    DayOfYear today;
    today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;
    private:
        int dd;
        int mm;
};

void DayOfYear::setDate(int mon, int day)
{
    mm = mon;
    dd = day;
}

int DayOfYear::getMonth() const{
    return mm;
}

int DayOfYear::getDay() const{
    return dd;
}
```

- *What is the output of this code?*

```
int main() {
    DayOfYear today;
    // today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;
    private:
        int dd;
        int mm;
};

void DayOfYear::setDate(int mon, int day)
{
    mm = mon;
    dd = day;
}

int DayOfYear::getMonth() const{
    return mm;
}

int DayOfYear::getDay() const{
    return dd;
}
```

# Constructor

**Constructor:** An “initialization” function that is guaranteed to be called when an object of the class is created

- \* If you don't explicitly write a constructor, C++ will generate a default one for you
- \* Member variables are initialized to junk values

```
int main() {
    DayOfYear today;
    today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

C++, attempt 5: We'll now try to improve this `class DayOfYear` {

```
public:
    void setDate(int mon, int day);
    int getMonth() const;
    int getDay() const;
private:
    int dd;
    int mm;
};

void DayOfYear::setDate(int mon, int day)
{
    mm = mon;
    dd = day;
}

int DayOfYear::getMonth() const{
    return mm;
}

int DayOfYear::getDay() const{
    return dd;
}
```



# Constructor:

## Writing your own

- Constructors must have the same name as the class
- Constructors don't have a return type
- Different types of constructors
  1. Constructor with no parameters (default)
  2. Constructor with parameters (parameterized constructor)
  3. Constructor with parameters that have default values

```
int main() {
    DayOfYear today;
    //today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

C++, attempt 6:

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;

    private:
        int dd;
        int mm;

};
```

//Function definitions omitted

# Parametrized Constructor

C++, attempt 7:

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;

    private:
        int dd;
        int mm;

};
```

```
int main() {
    DayOfYear today;
    //today.setDate(1, 9);
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

//Function definitions omitted

# Parametrized Constructor

What is the output of this code?

- A. Compiler error
- B. Junk values (default constructor is called)

```
int main() {
    DayOfYear today;
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
        << today.getDay();
}
```

C++, attempt 7:

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;
        DayOfYear(int mon, int day);

    private:
        int dd;
        int mm;

};

DayOfYear()::DayOfYear(int mon, int day)
{
    mm = mon;
    dd = day;
}

//Function definitions omitted
```

# Parametrized Constructor with default parameters

In the declaration of the parameterized constructor, specify default values for the parameters

Objects can be created in all the following ways:

```
DayOfYear today;
DayOfYear today{1,8};
DayOfYear today{2};
```

```
int main() {
    DayOfYear today;
    cout<<"Today's date is: ";
    cout<< today.getMonth() <<"/"
    << today.getDay();
}
```

C++, attempt 8:

```
class DayOfYear {

    public:
        void setDate(int mon, int day);
        int getMonth() const;
        int getDay() const;
        DayOfYear(int mon=1,int day=1);

    private:
        int dd;
        int mm;

};

DayOfYear()::DayOfYear(int mon, int day)
{
    mm = mon;
    dd = day;
}

//Function definitions omitted
```

# THE BIG FOUR

---

## Problem Solving with Computers-II



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

# The Big Four

1. Constructor
2. Destructor
3. Copy Constructor
4. Copy Assignment

# Constructor and Destructor

Every class has the following special methods:

- Constructor: Called right AFTER new objects are created in memory
- Destructor: Called right BEFORE an object is deleted from memory

The compiler automatically generates default versions, but you can override them

## Constructor (last class)

```
void foo(){  
    Quadratic p;  
    Quadratic* q = new Quadratic;  
    Quadratic w{10, 5, 1};  
}
```

How many times is the constructor called in the above code?

- A. Never
- B. Once
- C. Two times
- D. Three times



# Initializer lists

- \* Used to initialize member variables at the time they are created
- \* Must be used to initialize constant member variables

# Destructor

- Must have the same name as the class preceded by a ~ (tilda)
- Does not have a return type
- Called right BEFORE an object is deleted from memory

# Destructor

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

The destructor of which of the objects is called after foo() returns?

- A. p
- B. q
- C. \*q
- D. None of the above

# Copy constructor

- Creates a new object and initializes it using an existing object

# Copy constructor

- In which of the following cases is the copy constructor called?

**A. Quadratic p1; Quadratic p2{1, 2, 3};**

**B. Quadratic p1{1, 2, 3}; Quadratic p2{p1};**

**C. Quadratic \*p1 = new Quadratic{1, 2, 3};  
Quadratic p2 = \*p1;**

**D. B&C**

**E. A, B & C**

# Copy assignment

- Default behavior: Copies the member variables of one object into another

```
Quadratic p1{1, 2, 3}; // Parametrized constructor  
Quadratic p2;  
p2 = p1; // Copy assignment function is called
```

```
double foo(Quadratic p){  
    return p.evaluate(10);  
}  
  
int main(){  
    Quadratic q{1, 2, 3};  
    foo(q);  
}
```

Which of the following special methods is called as a result of calling foo?

- A. Parameterized constructor
- B. Copy constructor
- C. Copy Assignment
- D. Destructor

# 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 lectures)



# Next time

- Linked Lists and operator overloading

# 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.
- Constructors are used to initialized objects
- In the future we will see more features of OOP.

# Next time

- The big four and operator overloading