IMPLEMENTING C++ CLASSES: ACCESS SPECIFIERS CONSTRUCTORS

Problem Solving with Computers-II

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

CLICKERS OUT – FREQUENCY AB

From last lecture…

- Last time we defined a class DayOfYear 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 and executable

B. The -c option does not attempt to link code and no executable is produced

C. None of the above

The -c option only compiles the cpp file to an object file (.o)
It does attempt to link the code with other object files. So, no executable is produced

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;
}
```

mm is private, cannot be accessed here

C++, attempt 3:

What will be printed by this code?

A. 1/9
B. 1/1
C. 12/1
D. Compiler error
E. None of the above

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

```
class DayOfYear {
 public:
      void setDate(int mon, int day);
      int getMonth();
      int getDay();
  private:
      int dd;
      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?

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();
}
```

C++, attempt 4:

```
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();
}
```

C++, attempt 5: this version is correct!!!

```
class DayOfYear {

  public:
       void setDate(int mon, int day);
       int getMonth()const; //Accessor/Getter
       int getDay()const;   //Accessor/Getter
  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?***

Junk values

C++ does not automatically initialize member variables

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

C++, attempt 5: this version is correct!!!

```
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;
      DayOfYear();

  private:
      int dd;
      int mm;
};
DayOfYear :: DayOfYear () {
        dd = 1;
        m = 1;
}
//Function definitions omitted
```

Parametrized Constructor

Constructor that takes parameters.

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
```

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

Parametrized Constructor

What is the output of this code?

A. Compiler error

B. Junk values (default constructor is called)

If a parameterized constructor is defined, C++ does not generate a default constructor.

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

call to default

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();
}
```

This works now!

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
```

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

- Linked Lists (with classes)