Lecture 7.md

Lecture 7 - Classes and Objects

Access Control

C++ classes can have **public** and **private** members.

DayOfYear.h

```
class DayOfYear{
  private:
    int day;
    int month;
  public:
    void setDay(int d);
    void setMonth(int m);
    void print();
};
```

Note:

1. `int day` and `int month` are **private** members
   - They can only be accessed by **function members** in the class
2. `setDay(int d)` , `setMonth(int m)` , and `print()` are **public** members
   - They can be accessed anywhere in the source code via an object

main.cpp

```
int main(){
  DayOfYear FirstOfJuly;
  DayOfYear Christmas;
  FirstOfJuly.day = 1;
  FirstOfJuly.month = 7;
}
```

Note:

1. `FirstOfJuly.day = 1;` and `FirstOfJuly.month = 1;` are errors
   - **private members** cannot be accessed outside the FirstOfJuly object

Access Control works by class, not by object. If we define another **member function** called AddOne:

```
void DayOfYear::AddOne(){
  DayOfYear temp;
  temp.day = 1;
  temp.month = 1;
  day = day + temp.day;
  month = month + temp.month;
}
```

This member function can access any private member of any object of the same type

- In this case, the type is `DayOfYear`

- temp.day and temp.month are related to the object declared in `AddOne` , which is temp
- day and month are related to the object that `AddOne` is called on

For example, in main:

```
DayOfYear birthday;
birthday.setDay(18);
birthday.setMonth(6);
birthday.AddOne();
```

The day and month fields noted above are related to the object birthday.

Code Organization (Header Files)

Good practice and convention is to place class definitions in **header files** and to place member function implementations in **.cpp files**.

Organizing our `DayOfYear` class into separate `.h` and `.cpp` files:

DayOfYear.h

```
#ifndef _dayofyear_h
#define _dayofyear_h

class DayOfYear{
  private:
    int day;
    int month;
  public:
    int getDay();
    int getMonth();
    void setDay(int d);
    void setMonth(int m);
    void print();
};
#endif
```

Notes:

1. #ifndef checks if the given **preprocessor directive** is not defined
2. #define defines the given **preprocessor directive**
3. `getDay()` and `getMonth()` functions are considered **accessor**, or **getter** methods
   - they return/output some data (usually private) from the object
4. `setDay()` and `setMonth()` functions are considered **mutator**, or **setter** methods
   - they change some data (usually private) inside the object

DayOfYear.cpp

```
#include "DayOfYear.h"
#include <iostream>

int DayOfYear::getDay(){
  return day;
}
int DayOfYear::getMonth(){
  return month;
}
void DayOfYear::setDay(int d){
  day = d;
}
```

```
void DayOfYear::setMonth(int m){
  month = m;
}
void DayOfYear::print(){
  cout << day << "/" << month << endl;
}
```

Notes:

1. Need to include `DayOfYear.h` **header file**
   - The **member function** declarations are located inside `DayOfYear.h`
   - Will throw a **compile time error** if the header is not included

Compiling

Compiling code with class definitions in .cc files is the same as normal cpp compiling:

compiling main.o

> g++ -c main.cc

compiling DayOfYear.o

> g++ -c DayOfYear.cc

linking all (main.o and DayOfYear.o)

> g++ main.o DayOfYear.o -o myprog.exe

Need for Initialization - Constructors

Sometimes we would like to create variables and assign them initial values. For example, `int x = 0;`

- But how do we do that for objects?

Constructors are functions that you write and are automatically called upon creation of an object

- The **constructor** is used to initialize objects easily
  - You can pass in initial parameters to the argument with the **constructor**
- **Constructors *must*** have the same name as the class
  - Constructors are members of the class
  - Constructors have **no** return type
  - Constructors are usually public (although they can be private)
- Constructor selection (which constructor the compile chooses) happens **at runtime**
- C++ 2011 standard has a natural mechanism that allows for **default member initializers**
  - This simplifies initialization

One special case: The default constructor

- It has all the same properties as the constructor listed above
  - The default constructor *takes no argument*

DayOfYear.h

```
#ifndef _dayofyear_h
#define _dayofyear_h
```

```
class DayOfYear{
  private:
    int day;
    int month;
  public:
    DayOfYear();
    DayOfYear(int d,int m);
    DayOfYear(string s);
    int getDay();
    int getMonth();
    void setDay(int d);
    void setMonth(int m);
    void print();
};
#endif
```

main.cpp

```
int main(){
  DayOfYear birthday;
  DayOfYear christmas(25,12);
  DayOfYear mybirthday("12 16");
}
```

Notes:

1. `DayOfYear()` is the default constructor
2. `DayOfYear(int d,int m)` and `DayOfYear(string s)` are additional constructors
    - You can define as many constructors as you would like
        - However, every constructor ***must have different types or amounts of arguments***
3. `christmas(25,12)` is both object creation and initialization (via a constructor)

Properties of Default Constructor

Every class must contain at least one constructor

- If you *define no constructor*, the compiler will define the **default constructor** for you
- If you *do define a constructor*, the compiler will ***not*** define a **default constructor** for you

For example:

DayOfYear.h

```
#ifndef _dayofyear_h
#define _dayofyear_h

class DayOfYear{
  private:
    int day;
    int month;
  public:
    DayOfYear(int d,int m);
    ...
    void print();
};
#endif
```

main.cpp

```
int main(){
  DayOfYear birthday;
}
```

The above code will return a **compile time error**

- **default constructor** is no longer being generated for you

Options to fix this **compile time error**

- You must either define the default constructor
- You can change the object initialization to `DayOfYear birthday(16,12);`

Beep Boop