Ch 13: Introduction to Classes
Part 3

CS 2308
Fall 2011

Jill Seaman

Lecture 12

Review: The Class

- A class contains members:
  - variables AND
  - functions
- Members can be:
  - private: unaccessible outside the class
  - public: accessible outside the class.
- Goals:
  - information hiding: isolate details
  - interface: define limited access to data

Example class declaration

- Example: time.h

```
// file time.h
#include <string>
using namespace std;

  class Time         //new data type
  {
    // models a 12 hour clock
    private:
      int hour;
      int minute;
      void addHour();

    public:
      void setHour(int);
      void setMinute(int);

      string display();
      void addMinute();

  };
```


Defining (some) Member Functions

- the Time.cpp file:

```
// file time.cpp
#include <sstream>
#include <iomanip>
using namespace std;

#include "time.h"

void Time::setHour(int hr) {
  hour = hr;            // hour is a member var
}
void Time::setMinute(int min) {
  minute = min;         // minute is a member var
}

void Time::addHour() {  // a private member func
  if (hour == 12)
     hour = 1;
  else
     hour++;
}
```

A progam that uses Time

- File driver.cpp:

```
//using Time class (driver.cpp)
#include<iostream>
#include "time.h"
using namespace std;

int main() {
  Time t;
  t.setHour(12);
  t.setMinute(58);
  cout << t.display() <<endl;
  t.addMinute();
  cout << t.display() << endl;
  t.addMinute();
  cout << t.display() << endl;
  return 0;
}
```


Constructors

- A constructor is a member function with the same name as the class.
- It is called automatically when an object is created
- It performs initialization of the new object
- It has no return type

```
class Time
{
    private:
      int hour;
      int minute;
      void addHour();
    public:
      Time();      // Constructor prototype
...
```

Constructor Definition

- Note no return type, prefixed with Class::

```
// file time.cpp
#include <sstream>
#include <iomanip>
using namespace std;

#include "time.h"

Time::Time() {   // initializes hour and minute
   hour = 12;
   minute = 0;
}
void Time::setHour(int hr) {
  hour = hr;            // hour is a member var
}
void Time::setMinute(int min) {
  minute = min;         // minute is a member var
}
```


Constructor "call"

- From main:

```
//using Time class (driver.cpp)
#include<iostream>
#include "time.h"
using namespace std;

int main() {
  Time t;          //Constructor called implicitly here

  cout << t.display() <<endl;
  t.addMinute();
  cout << t.display() << endl;
  return 0;
}
```

Output:
12:00
12:01

Default Constructors

- A default constructor is a constructor that takes no arguments (like Time()).
- If you write a class with NO constructors, C++ will write a default constructor for you, one that does nothing.
- A simple instantiation of a class (with no arguments) calls the default constructor:

```
int main() {
  Time t;          //calls DEFAULT constructor
  t.setHour(12);
  t.setMinute(58);
  cout << t.display() <<endl;
```


Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - Indicate parameters in prototype:

```
class Time
{
   public:
      Time(int,int);      // Constructor prototype
...
```

  - Use parameters in the definition:

```
Time::Time(int hr, int min) {
   hour = hr;
   minute = min;
}
```

Passing Arguments to Constructors

- Then pass arguments to the constructor when you create an object:

```
int main() {
  Time t (12, 59);
  cout << t.display() <<endl;
}
```


Classes with no Default Constructor

- When all of a class's constructors require arguments, then the class has NO default constructor.
  - C++ will NOT automatically generate a constructor with no arguments unless your class has NO contructors at all.
- When there are constructors, but no default constructor, you **must** pass the required arguments to the constructor when creating an object.

Destructors

- Member function automatically called when an object is destroyed
- Destructor name is ~classname, e.g., `~Time`
- Has no return type; takes no arguments
- Only one destructor per class, i.e., it cannot be overloaded, cannot take arguments
- If constructor allocates dynamic memory, destructor should release it

Destructors

- Example: class decl

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

  class PasswordManager
  {
    private:
      char *encryptedPassword;

    public:
      PasswordManager(char *encPW);   //constructor
      ~PasswordManager();             //destructor
  };
```

Destructors

- Example: member functiond definitions (class impl)

```
#include "pwman.h"

PasswordManager::PasswordManager(char *encPW){
   encryptedPassword = new char [strlen(encPW)+1];
   strcpy(encryptedPassword,encPW);
}

PasswordManager::~PasswordManager() {
   delete [] encryptedPassword;
}
```


Destructors

- Example: member functiond definitions (class impl)

```
int main() {

  char k[] = "SSS";
  PasswordManager pm(k);     //calls constructor

  //do stuff with pm here

  return 0;
}    //end of prog, pm destroyed here, calls destructor
```

- When is an object destroyed?
  - at the end of its scope
  - when it is deleted (if it's dynamically allocated)

Overloaded Constructors

- Recall: when 2 or more functions have the same name they are *overloaded*.
- A class can have more than one constructor
  - They have the same name, so they are overloaded
- Overloaded functions must have different parameter lists:

```
class Time
{
    private:
      int hour;
      int minute;
    public:
      Time();
      Time(int);
      Time(int,int);
...
```


Overloaded Constructors

- definitions:

```
#include "time.h"

Time::Time() {
   hour = 12;
   minute = 0;
}
Time::Time(int hr) {
   hour = hr;
   minute = 0;
}
Time::Time(int hr, int min) {
   hour = hr;
   minute = min;
}
```

Overloaded Constructor “call”

- From main:

```
int main() {
  Time t1;
  Time t2(2);
  Time t3(4,50);

  cout << t1.display() <<endl;
  cout << t2.display() <<endl;
  cout << t3.display() << endl;
  return 0;
}
```

Output:
12:00
2:00
4:50

Overloaded Member Functions

- Non-constructor member functions can also be overloaded
- Must have unique parameter lists as for constructors

```
class Time
{
    private:
      int hour;
      int minute;
    public:
      Time();
      Time(int);
      Time(int,int);
      void addMinute();          //adds one minute
      void addMinute(int);       //adds minutes from arg
...
```

Arrays of Objects

- Objects can be the elements of an array:

```
int main() {
  Time  missedCalls[10];  //times of last 10 missed calls
}
```

- Default constructor for object is used when array is defined

Arrays of Objects

- Must use initializer list to invoke a constructor that takes arguments:

```
int main() {
  Time  missedCalls[10] = {1,2,3,4,5,6,7,8,9,10};
}
```

- The constructor taking one argument is used to initialize each of the 10 Time objects here

Arrays of Objects

- If the constructor requires more than one argument, the initializer must take the form of a function call:

```
int main() {

  Time  missedCalls[5] = {Time(1,5),
                          Time(2,13),
                          Time(3,24),
                          Time(3,55),
                          Time(4,50)};

}
```


Arrays of Objects

- It isn't necessary to call the same constructor for each object in an array:

```
int main() {

  Time  missedCalls[7] = {1,
                          Time(2,13),
                          Time(3,24),
                          4,
                          Time(4,50)};
}
```

- If there are fewer initializers in the list than elements in the array, the default constructor will be called for all the remaining elements.

Accessing Objects in an Array

- Objects in an array are referenced using subscripts
- Member functions are referenced using dot notation:

```
missedCalls[2].setMinute(30);

cout << missedCalls[4].display() << endl;
```