

Classes and Objects

Week 5

Gaddis: 13.2-13.12
14.3-14.4

CS 5301
Spring 2018

Jill Seaman

1

The Class

- A class in C++ is similar to a structure.
- A class contains members:
 - variables AND
 - functions (often called methods) (these manipulate the member variables).
- Members can be:
 - private: inaccessible outside the class (this is the default)
 - public: accessible outside the class.

2

Example class: Time

class declaration with functions defined inline

```
class Time {           //new data type
private:
    int hour;
    int minute;
public:
    void setHour(int hr)    { hour = hr; }
    void setMinute(int min) { minute = min; }
    int getHour() const    { return hour; }
    int getMinute() const  { return minute; }
    void display() const   { cout << hour << ":" << minute; }
};
int main()
{
    Time t1, t2;

    t1.setHour(6);
    t1.setMinute(30);
    cout << t1.getHour() << endl;

    t2.setHour(9);
    t2.setMinute(20);
    t2.display();
    cout << endl;
}
```

Output:

```
6
9:20
```

3

Using const with member functions

- `const` appearing after the parentheses in a member function declaration specifies that the function will **not** change any data inside the object.

```
int getHour() const    { return hour; }
int getMinute() const { return minute; }
void display() const   { cout << hour << ":" << minute; }
```

- These member functions don't change hour or minute.

4

Accessors and mutators

- Accessor functions
 - return a value from the object (without changing it)
 - a “getter” returns the value of a member variable

```
int getHour() const { return hour; }  
int getMinute() const { return minute; }
```

- Mutator functions
 - Change the value(s) of member variable(s).
 - a “setter” changes (sets) the value of a member variable.

```
void setHour(int hr) { hour = hr; }  
void setMinute(int min) { minute = min; }
```

5

Access rules

- Used to control access to members of the class
- public: can be accessed by functions inside AND outside of the class
- private: can be called by or accessed by only functions that are members of the class (inside)

```
int main()  
{  
    Time t1;  
  
    t1.setHour(6);  
    t1.setMinute(30);  
    cout << t1.hour << endl; //Error, hour is private  
};
```

6

Separation of Interface from Implementation

- Class declarations are usually stored in their own header files (Time.h)
 - called the specification file
 - filename is usually same as class name.
- Member function definitions are stored in a separate file (Time.cpp)
 - called the class implementation file
 - it must #include the header file,
- Any program/file using the class must include the class's header file (#include “Time.h”) 7

Time class, separate files

Time.h

```
// models a 12 hour clock  
class Time {  
  
private:  
    int hour;  
    int minute;  
  
public:  
    void setHour(int);  
    void setMinute(int);  
    int getHour() const;  
    int getMinute() const;  
  
    void display() const;  
};
```

Time.cpp

```
#include <iostream>  
#include "Time.h"  
using namespace std;  
  
void Time::setHour(int hr) {  
    hour = hr;  
}  
  
void Time::setMinute(int min) {  
    minute = min;  
}  
  
int Time::getHour() const {  
    return hour;  
}  
  
int Time::getMinute() const {  
    return minute;  
}  
  
void Time::display() const {  
    cout << hour << ":" << minute;  
}
```

8

Time class, separate files

Driver.cpp

```
//Example using Time class
#include<iostream>
#include "Time.h"
using namespace std;

int main() {
    Time t;
    t.setHour(12);
    t.setMinute(58);
    t.display();
    cout <<endl;
    t.setMinute(59);
    t.display();
    cout << endl;
}
```

9

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
- It can be overloaded: more than one constructor function, each with different parameter lists.
- A constructor with no parameters is the **default** constructor.
- If your class defines **no** constructors, C++ will provide a default constructor automatically.

Constructor Declaration+Definition

- Note no return type, same name as class:

<pre>// models a 12 hour clock class Time { private: int hour; int minute; public: Time(); Time(int,int); void setHour(int); void setMinute(int); int getHour() const; int getMinute() const; void display() const; };</pre>	<pre>#include <iostream> using namespace std; #include "Time.h" Time::Time() { hour = 12; minute = 0; } Time::Time(int hr, int min) { hour = hr; minute = min; } ...</pre>
--	--

11

Constructor Use

- Called from the object declaration

```
//Example using Time class
#include<iostream>
#include "Time.h"
using namespace std;

int main() {
    Time t;
    t.display();
    cout <<endl;

    Time t1(10,30);
    t1.display();
    cout << endl;
}
```

Output:

```
12:0
10:30
```

12

Destructors

- Member function that is 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 the class allocates dynamic memory, the destructor should release (delete) it.

```
class Time
{
public:
    Time();      // Constructor prototype
    ~Time();    // Destructor prototype ...
```

13

Arrays of Objects with initializer lists

- Each initializer takes the form of a function call:

```
int main() {
    Time recentCalls[7] = {Time(1),
                          Time(2,13),
                          Time(3,24),
                          Time(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.
- This array is initialized to 7 Time objects, set to 1:00, 2:13, 3:24, 4:00, 4:50, 12:00 and 12:00.

14

Composition

- When one class contains another as a member:

```
#include "Time.h"
class Calls
{
private:
    Time calls[10]; // times of 10 phone calls
    // this array is initialized using default constructor
public:
    void set(int,Time);
    void displayAll();
}
Calls.h
```

```
#include "Calls.h"
#include <iostream>
using namespace std;

void Calls::set(int i, Time t) {
    calls[i] = t;
}
void Calls::displayAll () {
    for (int i=0; i<10; i++) {
        calls[i].display(); //calls member function
        cout << " ";
    }
}
Calls.cpp
```

15

Composition

- Driver for Calls

```
//Example using Calls and Time classes
#include<iostream>
#include "Calls.h" //this includes "Time.h"
using namespace std;

int main() {
    Calls callTimes;
    Time t1(4,30);
    callTimes.set(0,t1);
    Time t2(11,42);
    callTimes.set(1,t2);

    callTimes.displayAll();
    cout << endl;
}
```

Output:

```
4:30 11:42 12:0 12:0 12:0 12:0 12:0 12:0 12:0 12:0
```

16

Copy Constructors

- Special constructor used when a newly created object is initialized using another object of the **same class**.

```
Time t1;  
Time t2 = t1;  
Time t3 (t1);
```

Both of the last two
use the copy constructor

- The **default copy** constructor, provided by the compiler, copies member-to-member.
- Default copy constructor works fine in most cases
- You can re-define it for your class as needed.

17

IntCell declaration

- Problem with the default copy constructor: what if the class contains a pointer member?

```
class IntCell  
{  
public:  
    IntCell (int);  
    int read () const;  
    void write (int );  
  
private:  
    int *storedValue;  
};  
  
IntCell::IntCell (int initialValue)  
{ storedValue = new int;  
  *storedValue = initialValue; }  
  
int IntCell::read () const  
{ return *storedValue; }  
  
void IntCell::write (int x)  
{ *storedValue = x; }
```

Note: dynamic
memory allocation

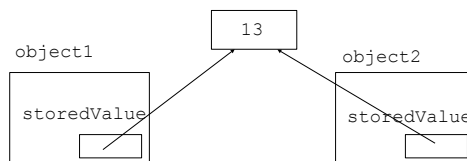
18

Problem with member-wise assignment

- What we get from member-wise assignment in objects containing dynamic memory (ptrs):

```
IntCell object1(5);  
IntCell object2 = object1; // calls copy constructor  
  
//object2.storedValue=object1.storedValue  
  
object2.write(13);  
cout << object1.read() << endl;  
cout << object2.read() << endl;
```

Output: 13
13



19

Programmer-Defined Copy Constructor

- Prototype and definition of copy constructor:

```
IntCell(const IntCell &obj); ← Add to class declaration
```

```
IntCell::IntCell(const IntCell &obj) {  
    storedValue = new int;  
    *storedValue = *(obj.storedValue);  
}
```

- Copy constructor takes a **reference** parameter to an object of the class
- This is required.

20

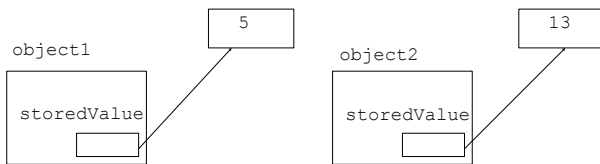
Programmer-Defined Copy Constructor

Each object now points to separate dynamic memory:

```
IntCell object1(5);
IntCell object2 = object1; //now calls MY copy constr

object2.write(13);
cout << object1.read() << endl;
cout << object2.read() << endl;
```

Output: 5
13



21

Pointers to Objects

- We can define pointers to objects, just like pointers to structures

```
Time t1(12,20);
Time *timePtr;
timePtr = &t1;
```

- We can access public members of the object using the structure pointer operator (->)

```
timePtr->setMinute(21);
cout << timePtr->display() << endl;
```

Output:
12:21

22

Dynamically Allocating Objects

- Objects can be dynamically allocated with new:

```
Time *tPtr;
tPtr = new Time(12,20);
...
delete tPtr;
```

You can pass arguments to a constructor using this syntax.

- Arrays of objects can also be dynamically allocated:

```
Time *tPtr;
tPtr = new Time[100];
tPtr[0].setMinute(11);
...
delete [] tPtr;
```

It can use only the default constructor to initialize the elements in the new array.

23