

CISC181 Introduction to Computer Science

Dr. McCoy

Lecture 23
November 19, 2009

1

Another Look at Classes - const

- We have declared constants before – they are objects of a type that cannot be modified.
- Recall – the compiler complains if you try to modify an object that is declared to be a constant.
- With classes, must assure the compiler that a constant is not being modified – so need to declare some member functions to take const objects

2

7.2 const (Constant) Objects and const Member Functions

- Principle of least privilege
 - Only allow modification of necessary objects
- Keyword `const`
 - Specify object not modifiable
 - Compiler error if attempt to modify `const` object
 - Example
 - ```
const Time noon(12, 0, 0);
```
    - Declares `const` object `noon` of class `Time`
    - Initializes to 12

© 2001 Pearson Education, Inc. All rights reserved. 24 18

### 7.2 const (Constant) Objects and const Member Functions

- `const` member functions
  - Member functions for `const` objects must also be `const`
    - Cannot modify object
  - Specify `const` in both prototype and definition
    - Prototype
      - After parameter list
    - Definition
      - Before beginning left brace

© 2001 Pearson Education, Inc. All rights reserved. 24 18

### 7.2 const (Constant) Objects and const Member Functions

- Constructors and destructors
  - Cannot be `const`
  - Must be able to modify objects
    - Constructor
      - Initializes objects
    - Destructor
      - Performs termination housekeeping

© 2001 Pearson Education, Inc. All rights reserved. 24 18

## Example D&D Exercise 7.7

Create a date class with the following capabilities:

1. Output the date in multiple formats such as: `ddd yyyy`; `mm/dd/yy`; `June 14, 1992`
  2. Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (1).
- `~/Class/cisc181/examples`
  - Driver: `22-date.cc`; Header: `date.h`; implementation: `date.cc`
  - This particular solutions does not use `const` objects – but you can see that the print functions allow the date variable to be `const` (see next example in slides)

6

## 7.2 const (Constant) Objects and const Member Functions

### • Member initializer syntax

- Initializing with member initializer syntax
  - Can be used for
    - All data members
  - Must be used for
    - `const` data members
    - Data members that are references

© 2011 Pearson Education, Inc. All rights reserved.

```

1 // fig. 7.2: fig07_04.cpp
2 // using a member initializer to initialize a
3 // member of a built-in data type.
4 #include <iostream>
5
6 using namespace std;
7 using namespace std;
8
9 class members {
10
11 public:
12 members() { int i = 0; int j = 0; } // default constructor
13
14 void addmembers();
15
16 //
17 int i;
18 int j; // and function addmembers
19
20 void print() const; // prints i and j
21
22

```

Outline

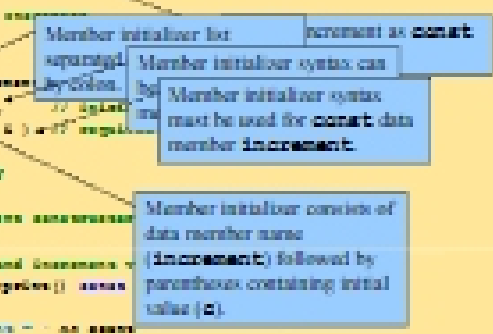
fig07\_04.cpp  
(1 of 3)

© 2011 Pearson Education, Inc. All rights reserved.

```

22 print();
23
24 int i;
25 int j; // const data member
26
27 }; // end class members
28
29 // constructor
30 members() {
31 i = 0;
32 j = 0; // const
33 }
34 // empty body
35
36 }; // end members constructor
37
38 // prints i and j
39 void members::print() const
40 {
41 cout << "i = " << i << ", j = " << j << endl;
42 }
43 // end function print
44

```



© 2011 Pearson Education, Inc. All rights reserved.

```

44 int main()
45 {
46 members m; // i = 0, j = 0
47
48 cout << "before incrementing: i = " << m.i << ", j = " << m.j << endl;
49 m.addmembers();
50 cout << "after increment: i = " << m.i << ", j = " << m.j << endl;
51 }
52
53 before incrementing: i = 0, j = 0
54 after increment: i = 10, j = 1
55 after increment: i = 20, j = 2
56 after increment: i = 30, j = 3
57
58 // end main
59

```

Outline

fig07\_04.cpp  
(2 of 3)

fig07\_04.cpp  
output(2 of 3)

© 2011 Pearson Education, Inc. All rights reserved.

```

1 // fig. 7.3: fig07_05.cpp
2 // attempting to initialize a member of
3 // a built-in data type with an assignment.
4 #include <iostream>
5
6 using namespace std;
7 using namespace std;
8
9 class members {
10
11 public:
12 members() { int i = 0; int j = 0; } // default constructor
13
14 void addmembers();
15
16 //
17 int i;
18 int j; // and function addmembers
19
20 void print() const; // prints i and j
21
22

```

Outline

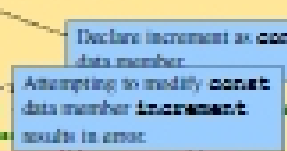
fig07\_05.cpp  
(1 of 3)

© 2011 Pearson Education, Inc. All rights reserved.

```

22 print();
23
24 int i;
25 int j; // const data member
26
27 }; // end class members
28
29 // constructor
30 members() { int i; int j; }
31 {
32 // members public
33 i = 0;
34 j = 0; // const: cannot modify a const object
35 }
36 // empty body
37
38 }; // end members constructor
39
40 // prints i and j
41 void members::print() const
42 {
43 cout << "i = " << i << ", j = " << j << endl;
44 }
45 // end function print
46

```



Outline

fig07\_05.cpp  
(2 of 3)

© 2011 Pearson Education, Inc. All rights reserved.

```

16 int main()
17 {
18 int month = 10, day = 10;
19
20 Date date("October 10, 2010");
21 date.getAddress();
22 Date date2("October 10, 2010");
23 date2.getAddress();
24 }
25
26 return 0;
27 } // end main

```

Outline  
 0207\_05.cpp  
 (1 of 3)  
 0207\_05.cpp  
 output(1 of 3)

Not using member initializer syntax to initialize **const** data member **increment** results in error.

Attempting to modify **const** data member **increment** results in error.

```

0207_05.cpp(20): error C4555:
 increment : must be initialized in non-static base
 class initializer list
0207_05.cpp(22):
 see declaration of 'increment'
0207_05.cpp(23): error C4555:
 increment : must be initialized in non-static base
 class initializer list

```

© 2011 Pearson Education, Inc. All rights reserved.

### 7.3 Composition: Objects as Members of Classes

- **Composition**
  - Class has objects of other classes as members
- **Construction of objects**
  - Member objects constructed in order declared
    - Not in order of constructor's member initializer list
    - Constructed before enclosing class objects (host objects)

© 2011 Pearson Education, Inc. All rights reserved.

```

1 // fig. 7.4: date.h
2 // more class definitions.
3 // member functions defined in date.cpp
4 #ifndef DATE_H
5 #define DATE_H
6
7 class Date {
8
9 public:
10 Date(int m = 1, int d = 1, int y = 2010);
11 void print() const; // prints
12 ~Date(); // provided to enforce destruction order
13
14 private:
15 int month; // 1-12 (mandatory member)
16 int day; // 1-31 based on month
17 int year; // any year
18
19 // utility function to see proper day for month and year
20 int checkDay(int d) const;
21 }; // end class Date
22
23 #endif

```

Outline  
 date.h (1 of 3)

Note no constructor with parameter of type **Date**. Recall compiler provides default copy constructor.

© 2011 Pearson Education, Inc. All rights reserved.

```

1 // fig. 7.5: date.cpp
2 // member function definitions for class Date.
3 #include "date.h"
4
5 using namespace std;
6
7 // include more class definitions from date.h
8 #include "date.h"
9
10 // constructor verifies proper values for month, valid
11 // utility function checkDay to verify proper values for day
12 Date::Date(int m, int d, int y) {
13 if (m < 1 || m > 12) // validate the month
14 month = 1;
15
16 if (d < 1 || d > 31 || !checkDay(m, d)) // validate the day
17 day = 1;
18
19 year = y; // should validate yr
20 day = checkDay(m, d); // validate the day
21 }

```

Outline  
 date.cpp (1 of 3)

© 2011 Pearson Education, Inc. All rights reserved.

```

26 // output Date objects to show when the constructor is called
27 Date date("October 10, 2010");
28 Date date2("October 10, 2010");
29
30 } // end Date constructor
31
32 // prints Date objects to show when
33 void main::print() const {
34 Date date("October 10, 2010");
35 Date date2("October 10, 2010");
36 }
37
38 // prints Date objects to show when
39 void main::~print() const {
40 Date date("October 10, 2010");
41 Date date2("October 10, 2010");
42 }
43
44 // prints Date objects to show when
45 void main::print() const {
46 Date date("October 10, 2010");
47 Date date2("October 10, 2010");
48 }
49
50 } // end destructor main

```

Outline  
 date.cpp (2 of 3)

No arguments; each member function contains implicit handle to object on which it operates.

Output to show timing of destructors.

© 2011 Pearson Education, Inc. All rights reserved.

```

55 // utility function to verify proper day values based on
56 // month and year; handles leap years, too
57 int Date::checkDay(int month, int day) const {
58 if (month < 1 || month > 12) return 0;
59 return month;
60
61 // February 28 check for leap year
62 if (month == 2 && year % 4 == 0 &&
63 (year % 100 != 0 ||
64 (year % 400 == 0 && year % 100 != 0))) {
65 return month;
66 }
67
68 return month < 8 ? month : month < 31 ? month : 31;
69 }
70
71 // Date objects in main::main were left bad values

```

Outline  
 date.cpp (3 of 3)

© 2011 Pearson Education, Inc. All rights reserved.