Lecture 9_2 – Classes

CST238 – Intro to Data Structures
YoungJoon Byun
ITCD

Lecture Objectives

- After completion of this lecture, you will be able to

`Time` Class Example (reminder)

```
1.  class Time
2.  {
3.   public:
4.      Time();
5.      Time(unsigned initHours, unsigned initMinutes, char initAMPM);

6.      unsigned getHours() const;
7.      unsigned getMinutes() const;
8.      unsigned getAMPM() const;
9.      unsigned getMilTime() const;

10.     bool setHours(unsigned newHours);
11.     bool setMinutes(unsigned newMinutes);

12.     void display(ostream & out) const;
13.     void set(unsigned hours, unsigned minutes, char am_pm);

14. private:
15.     unsigned myHours, myMinutes;
16.     char myAMorPM;
17.     unsigned myMilTime;
18. };
```

Copy Operations – Initialization and Assignment

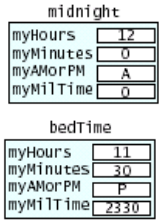
- Two **default copy operations** are provided for all classes.
  - Copy during initialization
  - Copy during assignment
- The operations provide *member-by-member* copy.

4

Copy Operations – Example (1 of 2)

- Suppose that you defined two **Time** objects.

```
Time midnight;
Time bedTime(11,30,`P`);
```

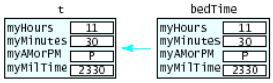


5

Copy Operations – Example (2 of 2)

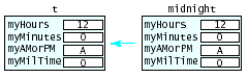
- Copy during initialization

```
Time t = bedTime; // or Time t(bedTime);
```



- Copy during assignment

```
t = midnight;
```



6

Overloading Operators (1 of 2)

- A symbol can be used more than one way similarly as an overloading function.
- Example
  - "**+**" operator is not only for integers, but also for doubles. And also for strings, etc.
  - Programmer can redefine the meaning of these symbols (e.g, +, -, *, etc.) for using in our own classes such as Time.

7

Overloading Operators (2 of 2)

- Two approaches
  - As a function member of a class.
  - As an ordinary function that is not a member function of a class.
- Note
  - Some operators like `::`, `.`, .* can't be overloaded.

8

Syntax of Overloading Operator (Function Member Approach)

- Suppose an operator you want to overload is Δ.
  - Then, we can define the operator as

```
ReturnType operatorΔ(type2 y)
{
  // Function body is here.
}
```

- After the definition, a Δ b is equivalent to

```
a.operatorΔ(b)
```

9

Syntax of Overloading Operator (Ordinary (nonmember) Function Approach)

- Suppose an operator you want to overload is Δ.
  - Then, we can define the operator as

```
ReturnType operatorΔ(type1 x, type2 y)
{
  // Function body is here.
}
```

- After the definition, a Δ b is equivalent to

```
operatorΔ(a, b)
```

10

Example (1 of 2)

- Overload "**>**" (greater than) operator for **Time** objects.
  - e.g., 5:30PM is greater than 1:45PM

```
1.// definition of the overloading operator
2.bool operator> (const Time &t1,
3.                const Time &t2)
4.{
5.  return t1.getMilTime() > t2.getMilTime();
6.}
```

11

Example (2 of 2)

```
1. // Usage of the overloading operator
2. Time mealTime(3, 30, 'P');
3. Time labTime(2, 30, 'P');

4. if (labTime > mealTime)
5. {
6.   cout << "Let's eat before the lab\n";
7. }
8. else
9. {
10.  cout << "Let's eat after the lab\n";
11.}
```

12

Exercise: "==" Operator for Time

```
// Usage of the overloading operator
Time mealTime(2, 30, 'P');
Time labTime(2, 30, 'P');
if (labTime == mealTime)
  cout << "Bad schedule!\n";
```

13

Overloading << (Output) Operator

- What is the operator "**<<**"?
  cout << "Average = " << average << endl;
- So far, we have displayed the content of a Time object as below:
  cout << "Meal time: ";
  mealTime.display(cout);
  cout << endl;
- But what do you think about this?
  cout << "Meal time: " **<< mealTime <<** endl;

14

<< Operator for Time Class (1 of 2)

15

<< Operator for Time Class (2 of 2)

- Usage of the operator

  Time labTime (12, 0, 'P');

  cout << labTime;

16

Overloading >> (Input) Operator

- What is the operator "**>>**"?

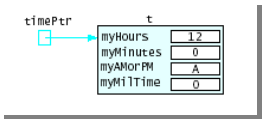
  cin >> average;
- Similarly as "<<" operator, you can define ">>" operator.

  cin >> **mealTime**;
- For detailed implementation, see our text book (page 172).

17

Pointers to Class Objects

```
Time * timePtr = &t;  // static
Time * timePtr = new Time;
// new - Time constructor is called
```



- Access with

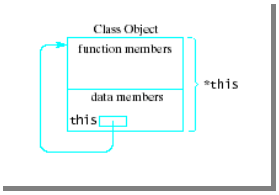
  `timePtr->getMilTime()` or

  `(*timePtr).getMilTime()`

18

The `this` Pointer

- Every class has a keyword, **`this`**
  - A pointer whose value is the address of the object itself.
  - Value of **`*this`** would be the object itself



19

Example of `this` Pointer (1 of 2)

```
1.  class Time {
2.  public:
3.      void Set(unsigned hours, unsigned minutes,
4.                  char ampm);
5.  private:
6.      unsigned hours, minutes;
7.      char ampm;
8.      unsigned myMilTime;
9.  };
```

20

Example of `this` Pointer (2 of 2)

```
1.  void Time::Set(unsigned hours, unsigned minutes, char
    ampm) {
2.      if (hours >= 1 && hours <= 12 && minutes >= 0 &&
3.          minutes <= 59 && (ampm = = 'A' || ampm == 'P')) {
4.              this->hours = hours;
5.              this-> minutes = minutes;
6.              this-> ampm = ampm;
7.              myMilTime = ToMilitary(hours, minutes, am_pm);
8.      } else
9.              cerr << "*** Can't set these values ***\n";
10. }
```

// In the definition of this method we use **this->hours**
// for the data member to distinguish the parameter
// with the same name in the methods.

21

Redundant Declarations

- Suppose a program needs **Time** class, so it has the following statement:
  #include "Time.h"
- The program also needs another class, for example, **Book** class, so it also has
  #include “Book.h"
- But the **Book** class internally uses the class **Time**, so the **Book** class has :
  #include “Time.h"
- What happens?

22

Redundant Declarations - Solution

- Use conditional compilation
  - Put the following compiler directives in all header (e.g., Time.h) files:
  - Example: Time.h

```
#ifndef TIME
#define TIME
…
#endif //at the end of this file
```

23

Summary

- Other class operations (chap 4.5)
  - Overloading operators
  - Copy operations
  - Input and output operations
- Next Lecture
  - Standard C++ Input/Output and String Classes (chap 5)

24

References

- Larry Nyhoff, *ADTs, Data Structures, and Problem Solving with C++*, 2nd Edition, Prentice-Hall, 2005
- Walter Savitch, *Problem Solving with C++*, 6th Edition, Addison-Wesley, 2006
- Dr. Meng Su's Lecture Notes
  http://cs.bd.psu.edu/~mus11/122Fa06/cse122Fa06.htm

25