Pointers to Structs and Objects, and the “this” pointer
Sections: 11.9, 13.3, & 14.5

CS 2308
Spring 2014
Jill Seaman

11.9: Pointers to Structures

• Given the following Structure:

```c
struct Student {
    string name;      // Student’s name
    int idNum;        // Student ID number
    int creditHours;  // Credit hours enrolled
    float gpa;        // Current GPA
};
```

• We can define a pointer to a structure

```c
Student s1 = {“Jane Doe”, 12345, 15, 3.3};
Student *studentPtr;
studentPtr = &s1;
```

• Now studentPtr points to the s1 structure.

Pointers to Structures

• How to access a member through the pointer?

```c
Student s1 = {“Jane Doe”, 12345, 15, 3.3};
Student *studentPtr;  // studentPtr is not a structure!
studentPtr = &s1;
cout << *studentPtr.name << end;  // ERROR
```

• dot operator has higher precedence than the
dereferencing operator, so:

```c
*studentPtr.name is equivalent to *(studentPtr.name)
```

• So this will work:

```c
cout << (*studentPtr).name << end; // WORKS
```

structure pointer operator: ->

• Due to the “awkwardness” of the notation, C has provided an operator for dereferencing structure pointers:

```c
studentPtr->name is equivalent to (*studentPtr).name
```

• The structure pointer operator is the hyphen (-) followed by the greater than (>), like an arrow.

• In summary:

```c
s1.name  // a member of structure s1
sptr->name // a member of a structure pointed to by sptr
```
Structure Pointer: example

- Function to input a student, using a ptr to struct

```cpp
void inputStudent(Student *s) {
    cout << "Enter Student name: ";
    getline(cin, s->name);
    cout << "Enter studentID: ";
    cin >> s->idNum;
    cout << "Enter credit hours: ";
    cin >> s->creditHours;
    cout << "Enter GPA: ";
    cin >> s->gpa;
}
```

- Call:

```cpp
Student s1;
inputStudent(&s1);
```

Dynamically Allocating Structures

- Structures can be dynamically allocated with new:

```cpp
Student *s.Ptr;
sptr = new Student;
sptr->name = "Jane Doe";
sptr->idNum = 12345;
...
delete sptr;
```

- Arrays of structures can also be dynamically allocated:

```cpp
Student *sptr;
sptr = new Student[100];
sptr[0].name = "John Deer";
...
delete [] sptr;
```

Structures and Pointers: syntax

- Expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-&gt;m</td>
<td>s is a structure pointer, m is a member</td>
</tr>
<tr>
<td>*a.p</td>
<td>a is a structure, p (a pointer) is a member. This expr is the value pointed to by p: *(a.p)</td>
</tr>
<tr>
<td>(*s).m</td>
<td>s is a structure pointer (a pointer to a structure), m is a member. Equivalent to s-&gt;m</td>
</tr>
<tr>
<td>*s-&gt;p</td>
<td>s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to *(s-&gt;p).</td>
</tr>
<tr>
<td>*(s)-&gt;p</td>
<td>s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to *(s-&gt;p).</td>
</tr>
</tbody>
</table>

in 13.3: Pointers to Objects

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

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

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

```cpp
timePtr->addMinute();
cout << timePtr->display() << endl;
```

Output:

```
12:21
```
Dynamically Allocating Objects

- Objects can be dynamically allocated with new:

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

- Arrays of objects can also be dynamically allocated:

```
Time *tptr;
tptr = new Time[100];
tptr[0].addMinute();
...
delete [] tptr;
```

Arrays of objects can be dynamically allocated:

```
Time *tptr;
```
**this: an object can return itself**

- Often, an object will return itself as the result of a binary operation, like assignment:
  
  ```
  v1 = v2 = x;  is equivalent to  v1 = (v2 = x);
  ```

- because associativity of = is right to left.

- But what is the result of (v2 = x)?

- It is the left-hand operand, v2.
  
  ```
  v1 = v2 = x;  is equivalent to  v2 = x;
  ```

**Returning *this**

```cpp
class Time {
  private:
    int hour, minute;
  public:
    Time operator= (Time);
};

Time Time::operator= (Time right) {
  hour = right.hour;
  minute = right.minute;
  return *this;
}
```

```cpp
Time time1, time2, time3(2,25);
time1 = time2 = time3;
```

```
cout << time1.display() << " 
<< time2.display() << " 
<< time3.display() << endl;
```

**Output:**

```
2:25 2:25 2:25
```

Note that this is a pointer, so it must be dereferenced to get the Time object.