Pointers to Structs and Objects

Sections: 11.9 & 13.3

CS 2308
Fall 2016

Jill Seaman

11.9: Pointers to Structures

- Given the following Structure:

```
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

```
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?

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

cout << *studentPtr.name << end;          // ERROR
```

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

studentPtr is not a structure!

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

- You must dereference the pointer first:

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


structure pointer operator: ->

- Due to the awkwardness of the pointer notation, C provides an operator for dereferencing structure pointers:

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

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

```
s1.name      // a member of structure s1

sptr->name   // a member of the structure sptr points to
```

Structure Pointer: example

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

```
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;
}
```

Or you could use a reference parameter. I'm using a pointer to give an example of using the -> syntax.

- Call:

```
Student s1;
inputStudent(&s1);
cout << s1.name << endl;
...
```


Dynamically Allocating Structures

- Structures can be dynamically allocated with new:

```
Student *sptr;
sptr = new Student;

sptr->name = "Jane Doe";
sptr->idNum = 12345;
...
delete sptr;
```

- Arrays of structures can also be dynamically allocated:

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

If a pointer points to an array, you can use square brackets with it, as if it were an array. Do not use -> here.

in 13.3: 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->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;
```

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].addMinute();
...
delete [] tptr;
```

It uses the default constructor to initialize the elements in the new array.
Initializer lists are not allowed.