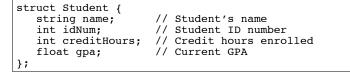
Pointers to Structs and Objects, and the "this" pointer Sections: 11.9, 13.3, & 14.5

CS 2308 Spring 2013

Jill Seaman

11.9: Pointers to Structures

• Given the following Structure:



• 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;</pre>

// ERROR

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

studentPtr is not a structure!

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

• So this will work:

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

structure pointer operator: ->

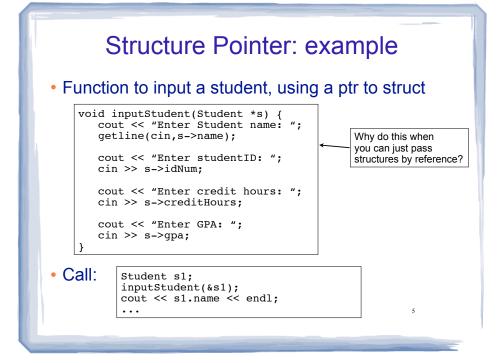
 Due to the "awkwardness" of the notation, C has provided an operator for dereferencing structure pointers:

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 a structure pointed to by sptr



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;

Structures and Pointers: syntax

• Expressions:

| s->m | s is a structure pointer, m is a member |
|---------|--|
| *a.p | a is a structure, p (a pointer) is a member. This expr is the value pointed to by p: *(a.p) |
| (*s).m | s is a structure pointer, m is a member. Equivalent to s->m |
| *s->p | s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to $*(s-p)$. |
| *(*s).p | s is a structure pointer, and p (a pointer) is in the structure pointed to by s. Equiv to $*(s-p)$. |

in 13.3: Pointers to Objects

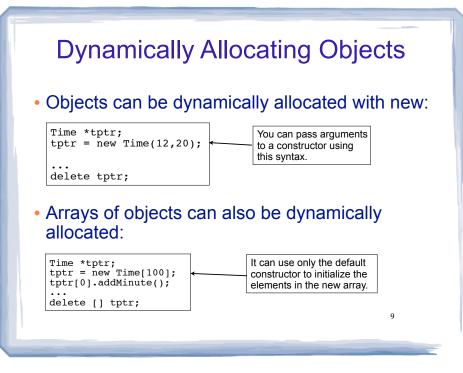
 We can define pointers to objects, just like pointers to structures

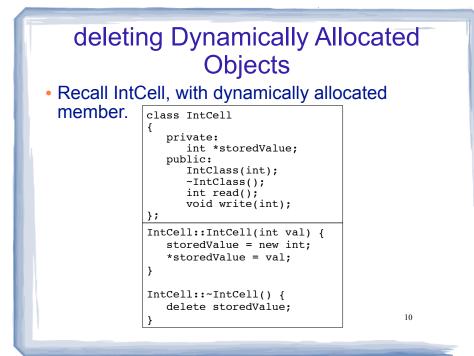
Time t1(12,20); Time *timePtr; timePtr = &t1;

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

timePtr->addMinute(); cout << timePtr->display() << endl;</pre>

Output: 12:21





deleting Dynamically Allocated Objects

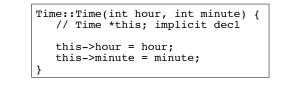
When is the storedValue deallocated?

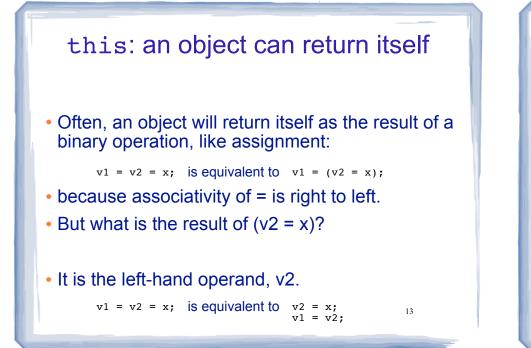
| <pre>#include "IntCell.h"</pre> | #include "IntCell.h" |
|--|---|
| <pre>int main() {</pre> | int main() { |
| <pre>IntCell *icptr; icptr = new IntCell(5);</pre> | <pre>IntCell ic(5);</pre> |
| <pre>cout << icptr->read()</pre> | <pre>cout << ic.read()</pre> |
| delete icptr; | |
| // | // |
| return 0; | return 0; |
| } | } |
| This calls icptr->~IntCell() first, which deletes (deallocates) icptr->storedValue. Then it deallocates icptr. | ic.~IntCell() is called here, which deletes (deallocates) ic.storedValue. Then ic is destroyed. |

in 14.5 The this pointer

- <u>this</u>: a predefined pointer available to a class's member functions
- this always points to the instance (object) of the class whose function is being executed.
- Use this to access member vars that may be hidden by parameters with the same name:

12





Returning this

