Week 4

Pointers & Dynamic Memory Allocation

Gaddis: Chapters 9, 11

CS 5301 Spring 2014

Jill Seaman

Pointers and Addresses

• The <u>address operator</u> (&) returns the address of a variable.

int x; cout << &x << endl; // 0xbffffb0c</pre>

- <u>Pointer</u>: a variable that stores the address of another variable, providing indirect access to it.
- An asterisk is used to define a pointer variable

int *ptr;

"ptr is a pointer to an int". It can contain addresses of int variables.
 ptr = &x;

Dereferencing and initializing

- The unary operator * is the dereferencing operator.
- *ptr is an alias for the variable that ptr points to.

int x = 10; int *ptr; //declaration, NOT dereferencing ptr = &x; //ptr gets the address of x *ptr = 7; //the thing ptr pts to gets 7

• Initialization:

```
int x = 10;
int *ptr = &x; //declaration, NOT dereferencing
```

 ptr is a pointer to an int, and it is initialized to the address of x.

Pointers as Function Parameters

• Use pointers to implement pass by reference.

```
//prototype: void changeVal(int *);
void changeVal (int *val) {
    *val = *val * 11;
}
int main() {
    int x;
    cout << "Enter an int " << endl;
    cin >> x;
    changeVal(&x);
    cout << x << endl;
}
```

 How is it different from using reference parameters?

Pointers and Arrays

• You can treat an array variable as if it were a pointer to its first element.

int numbers[] = {10, 20, 30, 40, 50};	Output:
<pre>cout << "first: " << numbers[0] << endl; cout << "first: " << *numbers << endl;</pre>	first: 10 first: 10 0xbffffb00
<pre>cout << &(numbers[0]) << endl; cout << numbers << endl;</pre>	0xbffffb00

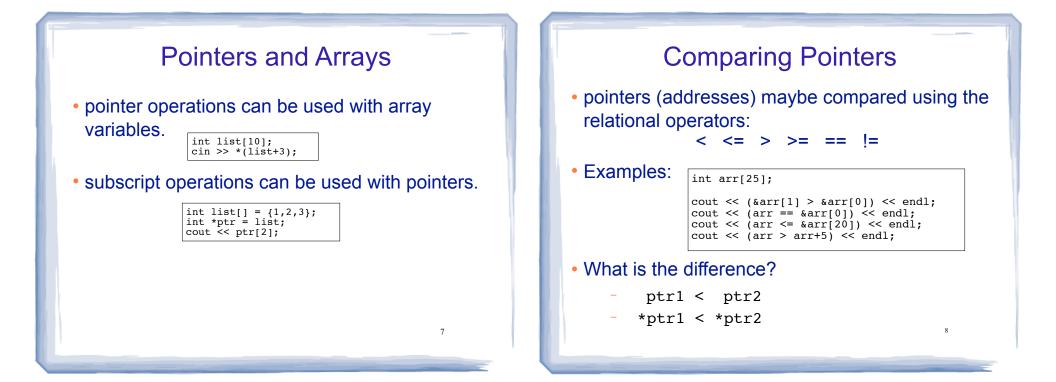
Pointer Arithmetic

 When you add a value n to a pointer, you are actually adding n times the size of the data type being referenced by the pointer.

int numbers[] = {10, 20, 30, 40, 50};

// sizeof(int) is 4. // Let us assume numbers is stored at 0xbfffb00 // Then numbers+1 is really 0xbfffb00 + 1*4, or 0xbfffb04 // And numbers+2 is really 0xbffffb00 + 2*4, or 0xbffffb08 // And numbers+3 is really 0xbffffb00 + 3*4, or 0xbffffb0c cout << "second: " << numbers[1] << endl;</pre> Output: cout << "second: " << *(numbers+1) << endl;</pre> second: 20 cout << "size: " << sizeof(int) << endl;</pre> second: 20 cout << numbers << endl;</pre> size: 4 0xbffffb00 cout << numbers+1 << endl;</pre> 0xbffffb04

Note: array[index] is equivalent to *(array + index)



5

Dynamic Memory Allocation

- When a function is called, memory for local variables is automatically allocated.
- When a function exits, memory for local variables automatically disappears.
- Must know ahead of time the maximum number of variables you may need.
- Dynamic Memory allocation allows your program to create variables on demand, during run-time.

The new operator

• "new" operator requests dynamically allocated memory for a certain data type:

```
int *iptr;
iptr = new int;
```

- new operator returns <u>address</u> of newly created <u>anonymous</u> variable.
- use dereferencing operator to access it:

```
*iptr = 11;
cin >> *iptr;
int value = *iptr / 3;
```

Dynamically allocated arrays

dynamically allocate arrays with new:

```
int *iptr; //for dynamically allocated array
int size;
cout << "Enter number of ints: ";
cin >> size;
iptr = new int[size];
for (int i=1; i<size; i++) {
    iptr[i] = i;
```

 Program will throw an exception and terminate if not enough memory available to allocate

delete!

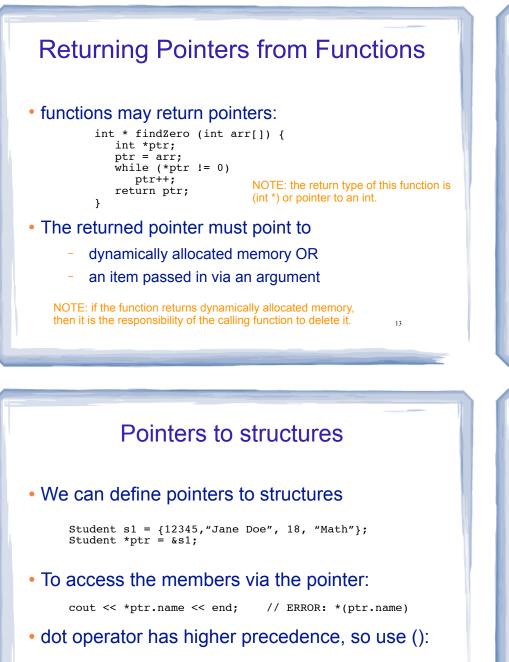
• When you are finished using a variable created with new, use the delete operator to destroy it:

```
int *ptr;
double *array;
ptr = new int;
array = new double[25];
...
delete ptr;
delete [] array; // note [] required for dynamic arrays!
```

 Do not "delete" pointers whose values were NOT dynamically allocated using new!

12

 Do not forget to delete dynamically allocated variables (Memory Leaks!!).



15

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

or equivalently, use ->:

cout << ptr->name << end;

Returning Pointers from Functions: duplicateArray

int *duplicateArray (int *arr, int size) { int *newArray; if (size ≤ 0) //size must be positive //NULL is 0, an invalid address return NULL; newArray = new int [size]; //allocate new array for (int index = 0; index < size; index++)</pre> newArray[index] = arr[index]; //copy to new array return newArray; Output int a $[5] = \{11, 22, 33, 44, 55\};$ int *b = duplicateArray(a, 5); 0 ok for (int i=0; i<5; i++) 1 ok if (a[i] == b[i])2 ok cout << i << " ok" << endl;</pre> 3 ok delete [] b; //caller deletes mem

14

4 ok

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;

No arrows (->) necessary. It's just an array of Student

16