CSCI 123 Introduction to Programming Concepts in C++

Brad Rippe

Pointers, Dynamic Memory
Overview

9.1 Pointers

9.2 Dynamic Arrays
9.1

Pointers
Pointers

• A pointer is the memory address of a variable
• Memory addresses can be used as names for variables
  – If a variable is stored in three memory locations, the address of the first can be used as a name for the variable.
  – When a variable is used as a call-by-reference argument, its address is passed
Pointers Tell
Where To Find A Variable

• An address used to tell where a variable is stored in memory is a pointer

  – Pointers "point" to a variable by telling where the variable is located
Declaring Pointers

• Pointer variables must be declared to have a pointer type
  – Example: To declare a pointer variable p that can "point" to a variable of type double:

    ```
    double *p;
    ```
  – The asterisk identifies p as a pointer variable
Multiple Pointer Declarations

• To declare multiple pointers in statements, use the asterisk before each pointer variable
  – Example:
    ```
    int *p1;
    int *p2;
    int v1;
    int v2;
    ```
    p1 and p2 *point* to variables of type int
    v1 and v2 *are* variables of type int
The address of Operator

• The & operator can be used to determine the address of a variable which can be assigned to a pointer variable
  – Example: p1 = &v1;

p1 is now a pointer to v1
v1 can be called v1 or "the variable pointed to by p1"
The Dereferencing Operator

• C++ uses the * operator in yet another way with pointers
  – The phrase "The variable pointed to by p" is translated into C++ as *p
  – Here the * is the dereferencing operator
    • p is said to be dereferenced
A Pointer Example

- \( v1 = 0; \)
- \( p1 = &v1; \)
- \( *p1 = 42; \)
- \( \text{cout} \ll v1 \ll \text{endl}; \)
- \( \text{cout} \ll *p1 \ll \text{endl}; \)

output:

42
42

\( v1 \) and \(*p1\) now refer to the same variable.
Pointer Assignment

• The assignment operator = is used to assign the value of one pointer to another
  – Example: If p1 still points to v1 (previous slide) then
    
    p2 = p1;
    
    causes *p2, *p1, and v1 all to name the same variable
Caution! Pointer Assignments

- Some care is required making assignments to pointer variables
  
  \( p1 = p3; \) // changes the location that \( p1 \) "points" to

  \( *p1 = *p3; \) // changes the value at the location that \( p1 \) "points" to
The new Operator

• Using pointers, variables can be manipulated even if there is no identifier for them
  – To create a pointer to a new "nameless" variable of type int:
    
    ```
    p1 = new int;
    ```
  – The new variable is referred to as *p1
  – *p1 can be used any place an integer variable can
    ```
    cin >> *p1;
    *p1 = *p1 + 7;
    ```
Assignment Operator

Uses of the Assignment Operator

```
p1 = p2;
```

Before:
- `p1` points to 84
- `p2` points to 99

After:
- `p1` points to 84
- `p2` points to 99

```
*p1 = *p2;
```

Before:
- `p1` points to 84
- `p2` points to 99

After:
- `p1` points to 99
- `p2` points to 99
Dynamic Variables

• Variables created using the new operator are called dynamic variables
  – Dynamic variables are created and destroyed while the program is running
// Program to demonstrate pointers and dynamic variables.
#include <iostream>
using namespace std;

int main()
{
    int *p1, *p2;

    p1 = new int;
    *p1 = 42;
    p2 = p1;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    *p2 = 53;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    p1 = new int;
    *p1 = 88;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;
    cout << "Hope you got the point of this example!\n";
    return 0;
}

Sample Dialogue
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
(a)
```
int *p1, *p2;
```

(b)
```
p1 = new int;
```

(c)
```
*p1 = 42;
```

(d)
```
p2 = p1;
```

(e)
```
*p2 = 53;
```

(f)
```
p1 = new int;
```

(g)
```
*p1 = 88;
```

---

**DISPLAY 9.3  Explanation of Display 9.2**
Passing Arguments by Reference with Pointers

There are three ways to pass arguments to a function in C++: *pass by value, pass by reference with reference arguments*, and *pass by reference with pointers*. 
void passByRefWithPtrArgs(int *aP1, int *aP2); 
int main() {
    int p1 = 44;
    int p2 = 22;
    // &p1 returns the memory address of p1
    passByRefWithPtrArgs(&p1, &p2);
    return 0;
}

void passByRefWithPtrArgs(int *aP1, int *aP2) {
    cout << "aP1 == " << *aP1 << endl;
    cout << "aP2 == " << *aP2 << endl;
}
new and Class Types

• Using operator new with class types calls a constructor as well as allocating memory
  – If Bike is a class type, then

    Bike *myBike;  // creates a pointer to a
                   // variable of type Bike
    myBike = new Bike;
               // calls the default constructor

    myBike = new Bike ("My Bike", 17, 26.0);
               // calls Bike(string, int, double);
Basic Memory Management

• An area of memory called the freestore is reserved for dynamic variables
  – New dynamic variables use memory in the freestore
  – If all of the freestore is used, calls to new will fail

• Unneeded memory can be recycled
  – When variables are no longer needed, they can be deleted and the memory they used is returned to the freestore
The delete Operator

• When dynamic variables are no longer needed, delete them to return memory to the freestore
  – Example:
    
    ```cpp
    delete p;
    ```

    The value of `p` is now undefined and the memory used by the variable that `p` pointed to is back in the freestore
Dangling Pointers

• Using delete on a pointer variable destroys the dynamic variable pointed to
• If another pointer variable was pointing to the dynamic variable, that variable is also undefined
• Undefined pointer variables are called dangling pointers
  – Dereferencing a dangling pointer (*p) is usually disastrous
Automatic Variables

• Variables declared in a function are created by C++ and destroyed when the function ends (popped off the function stack)
  – These are called automatic variables because their creation and destruction is controlled automatically

• The programmer manually controls creation and destruction of pointer variables with operators new and delete
Global Variables

- Variables declared outside any function definition are global variables
  - Global variables are available to all parts of a program
  - Global variables are not generally used
Type Definitions

• A name can be assigned to a type definition, then used to declare variables

• The keyword **typedef** is used to define new type names
  – Syntax:
    ```
    typedef KnownTypeDefinition  NewTypeName;
    ```

• KnownTypeDefinition can be any type
Defining Pointer Types

• To avoid mistakes using pointers, define a pointer type name
  – Example:  
    ```
    typedef int* IntPtr;
    ```

    Defines a new type, IntPtr, for pointer variables containing pointers to int variables

  – IntPtr p;
    is equivalent to
    int *p;
Multiple Declarations Again

• Using our new pointer type defined as
  
  ```c
  typedef int* IntPtr;
  ```

  – Prevent this error in pointer declaration:
    ```c
    int *P1, P2;  // Only P1 is a pointer variable
    ```

  • with
    ```c
    IntPtr P1, P2;  // P1 and P2 are pointer variables
    ```

• I don’t recommend declaring variables on one line so this problem only exists if you do
Pointer Reference Parameters

• A second advantage in using typedef to define a pointer type is seen in parameter lists
  – Example:
    ```
    void sampleFunction(IntPtr& pointerVar);
    ```
    is less confusing than
    ```
    void sampleFunction( int*& pointerVar);
    ```
Section 9.1 Conclusion

• Can you
  – Declare a pointer variable?
  – Assign a value to a pointer variable?
  – Use the new operator to create a new variable in the freestore?
  – Write a definition for a type called NumberPtr to be a type for pointers to dynamic variables of type int?
  – Use the NumberPtr type to declare a pointer variable called myPoint?
9.2

Dynamic Arrays
Dynamic Arrays

• A dynamic array is an array whose size is determined when the program is running, not when you write the program
Pointer Variables and Array Variables

• Array variables are actually pointer variables that point to the first indexed variable
  – Example:

        int array[10];
        typedef int* IntPtr;
        IntPtr arrayPtr;

• Variables `array` and `arrayPointer` are the same kind of variable

• Since `array` is a pointer variable that points to `array[0]`
  (*address of the first element*)

        arrayPtr = array;

causes `arrayPtr` to point to the same location as `array`
Pointer Variables
As Array Variables

- Continuing the previous example:
  Pointer variable `arrayPtr` can be used as if it were an array variable
- Example:

  ```
  arrayPtr[0], arrayPtr[1], ...arrayPtr[9]
  ```

  are all legal ways to use `arrayPtr`
- Variable `array` can be used as a pointer variable except the pointer value in `array` cannot be changed
  - This is not legal:

  ```
  IntPtr p2;
  ...
  // p2 is assigned a value
  array = p2
  // attempt to change array
  // error on the assignment
  ```
Creating Dynamic Arrays

• Normal arrays require that the programmer determine the size of the array when the program is written
  – What if the programmer estimates too large?
    • Memory is wasted
  – What if the programmer estimates too small?
    • The program may not work in some situations

• Dynamic arrays can be created with just the right size while the program is running
Creating Dynamic Arrays

• Dynamic arrays are created using the new operator
  – Example: To create an array of 10 elements of type double:

    ```
    typedef double* DoublePtr;
    DoublePtr d;
    d = new double[10];
    ```

• d can now be used as if it were an ordinary array!
Dynamic Arrays (cont.)

• Pointer variable $d$ is a pointer to $d[0]$

• When finished with the array, it should be deleted to return memory to the freestore

  – Example:

    ```
    delete [ ] d;
    ```

• The brackets tell C++ a dynamic array is being deleted so it must check the size to know how many indexed variables to remove

• Forgetting the brackets, is legal, but would tell the computer to remove only one variable
Pointer Arithmetic

• Arithmetic can be performed on the addresses contained in pointers
  – Using the dynamic array of doubles, d, declared previously, recall that d points to d[0]
  – The expression d+1 evaluates to the address of d[1] and d+2 evaluates to the address of d[2]
    • Notice that adding one adds enough bytes for one variable of the type stored in the array
Pointer Arithmetic Operations

• You can add and subtract with pointers
  – The ++ and -- operators can be used
  – Two pointers of the same type can be subtracted to obtain the number of indexed variables between
    • The pointers should be in the same array!

This code shows one way to use pointer arithmetic:

```c
for (int i = 0; city[i] != '\0'; i++) {
    cout << "city[" << i << "] " << *(city+i) << endl;
}
```
Multidimensional Dynamic Arrays

• To create a 3x4 multidimensional dynamic array
  – View multidimensional arrays as arrays of arrays
  – First create a one-dimensional dynamic array
    • Start with a new definition:
      \[ \text{typedef int* IntArrayPtr;} \]
    • Now create a dynamic array of pointers named m:
      \[ \text{IntArrayPtr *m = new IntArrayPtr[3];} \]
    – For each pointer in m, create a dynamic array of int's
      • \[ \text{for (int i = 0; i<3; i++)} \]
        \[ \text{m[i] = new int[4];} \]
A Multidimensional Dynamic Array

- The dynamic array created on the previous slide could be visualized like this:
Deleting
Multidimensional Arrays

• To delete a multidimensional dynamic array
  – Each call to new that created an array must have a corresponding call to delete[ ]
  – Example: To delete the dynamic array created on a previous slide:

    ```cpp
    for ( i = 0; i < 3; i++)
        delete [ ] m[i]; //delete the arrays of 4 int's
    delete [ ] m; // delete the array of IntArrayPtr's
    ```
Section 9.2 Conclusion

• Can you
  – Write a definition for pointer variables that will be used to point to dynamic arrays? The array elements are of type char. Call the type CharArray.

  – Write code to fill array "entry" with 10 numbers typed at the keyboard?

    ```
    int * entry;
    entry = new int[10];
    ```