**MSCF Programming Prep**

**Homework 2**

***Due At 11:59pm Monday, Aug. 13***

**1. Arrays and loops. (50 points)**

The arrays we have looked at so far are called *local automatic* (or *local temporary*) arrays. Such an array is automatically (and temporarily) allocated in a region of memory called the *stack* when the function containing the array is called. The array is automatically *deallocated* from the stack when the function containing the array returns.

More about initializing arrays:

1. An *uninitialized* local automatic array contains *undefined* element values (perhaps whatever junk happens to be on the stack at the location where the array is allocated).

**int a[5]; // 20 bytes worth of junk**

1. You can specify the dimension of (number of elements in) the array, and then initialize the array using a list of values in curly braces. The initial values can involve function calls and arithmetic, for example.

**int a[5]{ 1, 2, 3, atoi("132"), 4 + true };**

1. If you specify the dimension of the array and give *too many* initial values, this is an ***error***—the code won’t compile.

**int a[5]{ 1, 2, 3, 4, 5, 6 }; // error!**

1. If you specify the dimension of the array and give *fewer* than that number of initial values (but at least one), the unspecified elements will be *initialized with 0* (0 for **int**, 0.0 for **double**, …).

**int a[5]{ 1, 2 }; // same as {1, 2, 0, 0, 0};**

So if you need to initialize a huge array to all 0s, you can simply do:

**int a[50000]{}; // same as { 0, …, 0 };**

1. You can *leave out* the dimension of the array, and the compiler will size the array according to the initializer list.

**int a[]{ 1, 2, 3 }; // a is an array-of-3 int**

1. As we have seen, you can use an equal sign to specify initialization (this is the old C-style notation, and still works fine):

**int a[] = { 1, 2, 3 }; // a is an array-of-3 int**

Create a project called **hw2.1** containing a source code file named **hw2.1.cpp**. In your **hw2.1.cpp** file, either type or copy-and-paste this code:

**// File: hw2.1.cpp**

**// Author(s):**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int a[5];**

**int b[5]{ 1, 2, 3, 4, 5 };**

**int c[5]{ 6, 7 }**

**int d[5]{ 8, 9, 10, 11, 12, 13 };**

**int e[5]{};**

**int f[]{ 14, 15, 16, 17, 18, 19, 20, 21 };**

**cout << "In hw2.1:\n";**

**}**

1. Compile and execute this program. If you need to *add* or *remove* any code to get the program to compile successfully, please make the necessary changes, with a comment in each place describing what you changed.
2. Below the line **cout << "In hw2.1:\n";** add a **for** loop that displays each of the element values of the array **a** *on a single line* of output, with blank spaces separating the values. Add a comment in your source code showing the output. Does the output make sense?
3. As we have seen, you can declare and initialize a loop counter variable directly in the initialization part of the **for** loop. The *scope* of such a variable (that is, the part of the code in which the variable is visible for use) is restricted to the **for** loop itself. For example:

**for (int i = 0; i < 5; ++i) { // i is created here...**

**cout << "a[" << i << "]: " << a[i] << '\n';**

**} // ...and destroyed here**

As you would expect, you can use **(***n***)** or **{***n***}** rather than **=** *n* in initialization:

**for (int i(0); …**

and

**for (int i{0}; …**

also work for initializing the loop counter.

Write a **for** loop similar to the one you wrote in part (b) to display the elements of array **b** on a single line of output, but this time make sure to declare and initialize a loop counter variable in the initialization part of the **for** loop. Add a comment in your source code showing the output. Does the output make sense?

1. Add a **for** loop that displays array **c**’s elements, using the same loop counter variable name you used in part (c) above, to demonstrate that this reuse of the variable name *does not* *clash* with the previous use of this name in part (c). Add a comment in your code showing the output. Does the output make sense?
2. Declare *and initialize* an array-of-4-**double** named **g**, then write a **for** loop to display **g**’s element values on a single line.
3. Recall that you can use **sizeof** to get the number of bytes occupied by an object in memory. Display the sizes in bytes of the arrays **b**, **f**, and **g**. Add a comment in your code explaining these sizes.
4. Display the sizes in bytes of the array *elements* **b[0]**, **f[0]**, and **g[0]**. Add a comment in your code explaining these sizes.
5. How can you compute, *within the program*, the number of elements of an initialized array like **f** where the compiler counts the number of initial values to determine the array’s dimension? Display the number of bytes in **f**, the number of bytes in *one element* of **f**, and the number of elements in **f**.
6. Copy or type these two array declarations/initializations into your program:

**double h[] = { 2.2, 4.4, 1.1, 5.5,**

**3.3, 6.6, 8.8, 0.0 };**

**char ca[] = { 65, 'p', 112, 108, 'e' };**

For each array, display the size of the array, the size of the initial element of the array, the number of elements in the array, and the values of the elements in the array.

1. **Pointers and Arrays. (50 points)**
2. Create a project called **hw2.2** containing a source code file named **hw2.2.cpp**. In your **hw2.2.cpp** file, either type or copy-and-paste this code:

**// File: hw2.2.cpp – pointers and arrays**

**// Author(s):**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int i = 3;**

**int j(-7);**

**int k{21};**

**cout << "i: " << i << "\n";**

**cout << "j: " << j << "\n";**

**cout << "k: " << k << "\n";**

**}**

Compile and run this program.

1. Add three additional statements to display the *addresses* of **i**, **j**, and **k**. For example:

**cout << "address of i: " << &i << "\n";**

Are **i**, **j**, and **k** in *ascending* memory locations, or *descending*? Add a comment

to your code saying what you observe. (You *do not* need to explain *why* the

addresses are ascending or descending: this is up to the compiler to decide.)

1. Add declarations of three pointer variables—**pi**, **pj**, and **pk**—*initialized* to contain the addresses of **i**, **j**, and **k**, respectively. Then, add three statements to display the value of each pointer, and the “contents-at” the location where each pointer points. For example, one of these lines of output could look like:

**value of pi: 0x2201410, contents at pi: 3**

Add comments to your code describing the meanings of the values you are seeing.

1. Add this declaration and initialization of an array-of-5 **double** named **a**:

**double a[]{ 0.9, 1.8, 2.7, 3.6, 4.5 };**

Write a **for** loop that displays the values of the elements in **a**.

1. Add a declaration of a pointer-to-**double** variable named **pa2**, initialized with the address of the sub-2 (the third) element of **a**. Then add a statement to display the value of **a[2]** and the “contents at” the address **pa2**. Are these values the same, or not? Comment.
2. Add these statements to your program; compile and run; add comments to your code, explaining the output.

**cout << "\*pa2 - 1: " << \*pa2 - 1 << "\n";**

**cout << "\*(pa2 - 1): " << \*(pa2 - 1) << "\n";**

**cout << "\*pa2 + 1: " << \*pa2 + 1 << "\n";**

**cout << "\*(pa2 + 1): " << \*(pa2 + 1) << "\n";**

1. You can “do algebra” on offset-and-indirection notation vs. array subscript notation. These relationships are *always always always* true, if *P* is any pointer to data (other than **void**) and *N* is any integer:

*P***[***N***]** *<==>* **\*(***P* **+** *N***)**

**&***P***[***N***]** *<==>* **&\*(***P* **+** *N***)**

*<==>**P* **+** *N*

In fact, addition *commutes* as in arithmetic, so:

*P***[***N***]** *<==>* **\*(***P* **+** *N***)**

*<==>* **\*(***N* **+** *P***)**

*<==> N***[***P***]**

Add a line to your code to see whether your compiler lets you display **1[pa2]**. Most compilers will, although I’m unclear whether this is “standard.”

1. Write a **for** loop, using an integer index variable and array subscript notation, to display all the element values in **a** by using **pa2** as the “name of the array”. Notice that your subscripts will range from -2 to 2, *not* from 0 to 4.
2. Write another **for** loop, using an integer “offset” variable and the “contents at” (or *indirection* on) pointer notation to display the values in **a**, using **a** as a pointer. To illustrate, this statement uses *both* array subscript notation *and* “offset/contents at” notation to display the value 4.5 twice from the end of array **a**:

**cout << "a[4]: " << a[4]**

**<< "\*(a + 4): " << \*(a + 4) << "\n";**

***When you are finished, create a .zip archive containing your group’s hw2.1.cpp file and hw2.2.cpp file. Remember to include your name and your partner(s) name(s) as Authors. Upload the .zip archive to Canvas.***