Chris Dang Lab 4

1. Add function stub with empty main. Check when done\_\_x\_\_

2. Now add two typedef statements: typedef array-element-type type-name[array-size];

ahead of main() to define the two data types:

IntegerArray for arrays with 16 integer elements

CharArray for arrays with 10 character elements

Check here when done\_\_x\_\_

3. Inside the main() function, declare and initialize an IntegerArray variable prime to be an array containg the 16 integers 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, using a declaration of the form:

type-name array-name = {list-of-values} ;

Check here when done\_\_x\_\_

4. Check that the array have been initialized properly by writing a for loop to display the elements of prime. Then compile, link, and execute the program to check if your statements are correct.

Check here when done\_\_x\_\_

5. Test to see what happens when you have too many items in an initializer list. Describe what happens:

**Visual Studio has an error saying that there are too many initializers**

6. Having an initializer list with too few values is ok. What happens when you have fewer than 16 elements? Describe what happens: **Visual studio reports no errors and the array is displayed normally with the 15 elements I used.**

7. Now you will repeat the experiment with CharArray instead of IntegerArray. Use char animal[10] = {'r', 'h', 'i', 'n', 'o', 'c', 'e', 'r', 'o', 's'} and display using a for loop. Check here when done\_\_x\_\_

8. Now check if adding one or more characters causing too many initializers error.Describe what happens: **Visual Studio has an error saying that there are too many initializers**

9. Now check what happens when there are fewer values in the initializer list. Remove all but the first 5 characters in the initializer list and compile, link and execute. Describe what happens: **Visual studio reports no errors and the array is displayed normally with the 5 elements I used with whitespace where the other characters were**

10. What did the output operator do when the for-loop sent it to the character array elements that had not been initialized? To see this, modify the output statement in the for-loop to display the ascii codes generated for each character array elements. (Hint: Use type cast) Tell below what is used to initialize the uninitialized array elements. **The uninitialized elements were not displayed as anything. In place, zeroes were added as the contents of the uninitialized array.**

11. Initialize the char array with a string literal “elephant” and then display the array directly with << as in   
 cout << animal << “\*\*\*\*\*\n” ; What happened? **The array animal was outputted directly followed by the stars without needing a for loop.**

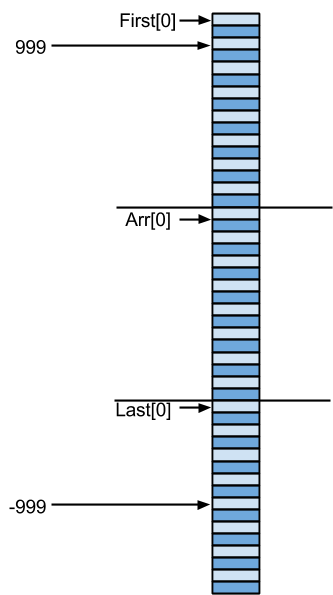
12 Repeat with “rhinoceros” instead and describe what happens. “**rhinoceros” was said to have 11 characters in it’s string, making it too long of an initializer string**

(ii) Write C++ statements below that could be used to determine length of the string stored in animal, that is the number of non-null characters.

**CharArray[] = "zebra" ;**

**for(int i = 0; i < 10; i++)**

**cout << animal[i] << i+1 << " ";**

13. Initialize int arrays with *first* to all 0’s, *arr* to all 1’s, and *last* to all 2’s

Display and write down their memory locations and include a

memory

map to the side.

first : **0042FDE8**

arr : **0042FDA0**

last : **0042FD58**

How many bytes are allocated to each array? **64 bytes**

Check using sizeof(type). Check here when finished\_\_x\_\_

What are addresses of:

arr[2]? **0042FDA8**

arr[3]? **0042FDAC**

Add code to show addresses of arr[2] and arr[3]. Results are:

arr[2] is at **0042FDA8**

arr[3] is at **0042FDAC**

Are they the same guesses?

**Yes they are the same as the earlier guess.**

14. In the output statement of the step 13, remove the & of &first,& arr, and &last. What can you conclude about the value of an array variable?

**Memory addresses are stored in the array variables.**

15. DIsplay the addresses of the indexes of arr[]. What can be concluded? **The addres of each location in the int array is 4 bytes away from the preceding and following index.**

16. Display the values of arr using subset[] notation and pointer notation. What can you conclude?  
**The output appears the same; it displays the contents of the array, which is all 1’s**

What happens if you remove the parantheses from \*(arr+i)?

**It dereferences the array index contents, and then adds the value of i.**

17**.** Display elements of first, arr, and last, in that order. Check when finished\_\_x\_\_

18. Add arr[-10] = -999; and arr[20] = 999; before the output statements of question 17. What elements change?

**first[2] was changed to 999 and last[8] was changed to -999**

Add marks to memory map from step 13 where the variables were changed. **Check when done\_\_x\_\_**

Add cout << “\narr[-10]...arr[20]:\n”;

for (int i = -10; i <=20; i++)

cout << arr[i] << “ “ ;

Why were the elements of first and last changed? **Because the assignments access the locations back and forth of arr because arrays are all stored in line.**

19. add:

**int mat[3][4] = { {11, 22, 33, 44},**

**{55, 66, 77, 88},**

**{-1, -2, -3, -4} } ;**

**for (int i = 0; i < 3; i++) {**

**for (int j = 0; j < 4; j++)**

**cout << setw(5) << mat[i][j] ;**

**cout << endl ;**

**}// end for** check when done\_\_x\_\_

20. Write a line of code you would use to find the base address of mat.

**cout << mat << endl ;**

What is the base address of mat? **0032F8E8**

Find addresses of the elements of mat, then draw a memory map.

Was the allocation done rowwise or columnwise?

**Done vertically in columnwise**

21. Add statements to code to allow you to determine the values of the expressions.

\*(mat + 0)  **0037FD20** \*(mat + 1) **0037FD30** \*(mat + 2) **0037FD40**

\*\*(mat + 0) **11**  \*\*(mat + 1) **55** \*\*(mat + 2)  **-1**

\*(\*mat + 0) **11**  \*(\*mat + 1) **22** \*(\*mat + 2)  **33**

\*(\*(mat + 1) + 0) **55** \*(\*(mat + 1) + 1) **66** \*(\*(mat + 1) + 2)  **77**

22. Compare answers with memory map and guess what the following expressions give.

\*(mat + i) **This will give a memory address, because this is pointer notation for an index**

**\*\***(mat + i) **This will give the contents of \*(mat +i) because it dereferences the address**

\*(\*mat + i) **This will give the contents of \*mat (initial location, index 0), then add i to get to mat+i and then dereference to get the value**

**\*(\*(mat + i) + j) This will give a value that first differenced the (mat + i) to give a starting location (index i) then jumps to mat+j followed with the dereference to give a value.**