Bryan Wong

805111517

Professor Stahl

CS31

March 7, 2019

Project 6 Report

­­

1. **a.** In the original code, the pointer is used to change the last value of the array to 1000, but the pointer is not incremented to match the last value of the array. When the values at the pointer are printed at the end, it does not print the last value because it is pointing to the second-to-last value. We correct this by incrementing the pointer after it modifies the last value in the array. Also, when printing out the elements of the array, the pointer is decremented before printing. This also causes the last value to not get printed, and a garbage value to be printed at the end. We fix this by moving the decrement statement to the bottom.

**int** main()

{

**int** arr[4] = { 0, 1, 2, 3 };

**int**\* ptr = arr;

\*ptr = arr[ 1 ]; *// set arr[0] to 1*

\*(ptr + 1) = arr[ 0 ] \* 10; *// set arr[1] to 10*

ptr += 2;

\*ptr = arr[ 1 ] \* 10; *// set arr[2] to 100*

ptr++; *// ptr was not incremented to*

*// properly point to the last value it*

*// modified. In the original code,*

*// only the first 3 values are printed*

\*ptr = 1000; *// set arr[3] to 1000*

**while** (ptr >= arr)

{

cout << " " << \*ptr; *// print values*

ptr--; *// in the original code, ptr was decremented first.*

*// This caused the last value in the array to never be*

*// printed*

}

cout << endl;

**return**( 0 );

}

**b.** In the original code, the parameter int\* p is a pass-by-value rather than a pass-by-reference. When the function ends, the inputted ptr is not pointing to the same place that p was. To fix this, we add an ampersand to the input parameter p.

**void** findLastZero(**int** arr[], **int** n, **int**\* &p)

**c.** In the original code, the pointer p is not linked to any address. Therefore, when we try to feed the function a null pointer, it creates an error. We correct this by first linking the pointer p to an int variable.

**int** main()

{

**int** i;

**int**\* p = &i;

biggest(15, 20, p);

cout << "The biggest value is " << \*p << endl;

**return**( 0 );

}

**d.** In the original code, many of the comparison statements between str1 and str2 are missing asterisks, which means the addresses are being compared rather than the values that are the addresses. To correct this, we add asterisks where appropriate.

**bool** match(**const** **char** str1[], **const** **char** str2[])

{

**bool** result = **true**;

**while** (\*str1 != 0 && \*str2 != 0) *// zero bytes at ends*

{

**if** (\*str1 != \*str2) *// compare corresponding characters*

{

result = **false**;

**break**;

}

str1++; *// advance to the next character*

str2++;

}

**if** (result)

{

result = (\*str1 == \*str2); *// both ended at same time?*

}

**return**( result );

}

**int** main()

{

**char** a[10] = "pointy";

**char** b[10] = "pointless";

**if** (match(a,b))

{

cout << "They're the same!" << endl;

} **else** cout << "They don't match" << endl;

}

**e.** The array arr only exists within the scope of the computeFibonacciSequence() function. When it ends, all the values inside the array are no longer at the same addresses. When the pointer locates the values at those addresses, garbage values show up instead.

1. 1. F

2. g

3. a **pointer set to point at fifth element of array**

4. b

5. d

6. b. **this points to the fifth element**

7. b. **this points to the second element**

8. e.

9. h

**I think there is an error in this question. The fourth element (index 3) is never pointed to. If the question really meant the index 4, then #4 and #6 satisfies d.**

1. This is our output:

diff=1

4

79

5

9

-1

19

Explanation: diff=1 is printed because ptr is pointing to the array at an index that is 1 from the last value.

The rest of the values are the members of the array after performing several functions. This is explained in-depth in the following annotated code:

**int**\* minimart(**int**\* a, **int**\* b) *// this function returns the address of the*

*// pointer that points to the smaller value*

{

**if** (\*a < \*b)

**return** a;

**else**

**return** b;

}

**void** swap1(**int**\* a, **int** \*b) *// this functions swaps the addresses that the two*

*// pointers are pointing to*

{

**int**\* temp = a;

a = b;

b = temp;

}

**void** swap2(**int**\* a, **int** \*b) *// this function swaps the values that the*

*// pointers are pointing to*

{

**int** temp = \*a;

\*a = \*b;

\*b = temp;

}

**int** main()

{

**int** array[6] = { 5, 3, 4, 17, 22, 19 };

**int**\* ptr = minimart(array, & array[2]); *// returns the address that*

*// contains the smaller value out*

*// of array[0] and array[2].*

*// Since 4 < 5, the address of*

*// array[2] is returned*

ptr[1] = 9; *// sets array[3] to 9*

ptr += 2; *// increments pointer to point to array[4]*

\*ptr = -1; *// Sets the value at array[4] to -1.*

\*(array+1) = 79; *// Sets the value at array[1] to 79.*

*// This is the array now: { 5, 79, 4, 9, -1, 19 }*

cout << "diff=" << &array[5] - ptr << endl; *// this is number of indexes*

*// from the end that ptr is*

*// since it is pointing to*

*// array[4], it is 1 away*

*// from the end*

swap1(&array[0], &array[1]); *// this function swaps the addresses the two*

*// pointers are pointing to, but array*

*// pointers are constant. Also, the function*

*// uses pass-by-values.*

*// Therefore, nothing happens.*

swap2(array, &array[2]); *// this function swaps the values that are*

*// located at the two pointers.*

*// array[0] now contains 4. array[2] now*

*// contains 5.*

**for** (**int** i = 0; i < 6; i++) *// prints out the values in our array!*

cout << array[i] << endl;

**return**( 0 );

}

1. I wrote a function that uses a pointer to locate digits. If located, every value in the array will be shifted to the left.

**void** deleteDigits(**char** arr[]) {

**char** \* ptr = arr;

**while**(\*ptr != '\0') { *// will search values of the array until it encounters the null*

*// character, signaling the end of the array*

**if**(\*ptr >= '0' && \*ptr <= '9') { *// checks that the value at the pointer is a digit*

*// if so, we will need to delete the values by shifting them over.*

**while**(\*ptr != '\0'){ *// this section of the code moves all remaining values "left"*

\*ptr = \*(ptr+1);

ptr++;

}

ptr = arr - 1; *// We reset ptr to the first index of arr, and repeat the process.*

*// We subtract by 1 to offset the incrementing later*

}

ptr++; *// increment to search all values of the array*

}

}

**int** main()

{

**char** msg[100] = "Happy 2019!";

deleteDigits(msg);

cout << msg << endl; *// prints: Happy !*

}