Project 6 Homework Problems

Ben He

UID: 804962948

**1.**

**a)** Before going through the while loop, you must first set ptr to the last element of the array otherwise it will only print from the middle of arr which is the last location that ptr was set to. Additionally, within the while loop, you should print out the value of \*ptr before incrementing it otherwise you are missing the first element because you incremented the ptr before even displaying its dereferenced value. The corrected program:

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

ptr[1] = 1000;

ptr++; //set pointer to end of array

while (ptr >= arr)

{

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

ptr--; //increment ptr AFTER displaying the location’s

//value

}

cout << endl;

return(0);

}

**b)** This function doesn’t work because it only interacts with a copy of the ptr since the pointer is being passed by value. Instead, the ptr parameter should be passed by reference so that the ptr is updated in the main function too.

void findLastZero(int arr[], int n, int\* &p) //pointer parameter must be

//passed by reference instead of

//by value

{

p = nullptr; // default value if there isn't a 0 in the array at all

for (int k = n - 1; k >= 0; k--)

{

if (arr[k] == 0) // found an element whose value is 0

{

p = arr + k; // change the value of p

break; // stop looping and return

}

}

}

int main()

{

int nums[6] = { 10, 20, 0, 40, 30, 50 };

int\* ptr;

findLastZero(nums, 6, ptr);

if (ptr == nullptr)

{

cout << "The array doesn't have any zeros inside it." << endl;

}

else

{

cout << "The last zero is at address " << ptr << endl;

cout << "It's at index " << ptr - nums << endl;

cout << "The item's value is " << \*ptr << " which is zero!" << endl;

}

return(0);

}

**c)** This function might not work because the pointer created in the main was not assigned to any particular address, so it automatically has some garbage-y address stored, and that address might be holding something important. The biggest function changes the value of whatever is at the address of p, so changing the value of whatever is at the garbage address can potentially cause a lot of issues. Instead, the p pointer can be initialized to the address of some placeholder variable. My solution:

#include <cassert>

using namespace std;

void biggest(int value1, int value2, int \* resultPtr)

{

if (value1 > value2)

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

int placeHolder = 0; //placeholder variable created so that

//biggest() doesn’t change something it

//isn’t supposed to

int\* p = &placeHolder;

biggest(15, 20, p);

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

return(0);

}

**d)** In the match function, it must compare each element of the char arrays str1 and str2 to see if they are the same, which means you need to dereference them, otherwise comparing them is meaningless because the while loop will never end since the addresses will never equal zero, and in the if statement, the addresses will never be the same so result will also be set to false, and in the end when the function checks if the cstrings ended at the same time, it will always be false if you just compare the pointers because the addresses of the two character arrays in main are always going to be different. Fixed:

// return true if two C strings are equal

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;

}

}

**e)**

There are two things that this function isn’t doing correctly. The first is that the array created in the computeFibonacciSequence function is local, so as soon as the function ends, the array will be deleted from the stack and the main function won’t be able to access the values created in the array. Additionally, even if the array was accessible (like if it was declared globally or something), the calculation of the sequence is incorrect due to the condition of i<n-2 in the for loop of the fibonacci function, so the arr[8] array would hold 1 1 1 2 3 5 8 13 instead of 1 1 2 3 5 8 13 21.

**2.** Assuming that the statements are independent from each other and aren’t happening in order.

1. F

2. G

3. A

4. B

5. D

6. C

7. B

8. E

9. H

**3.**

The program prints:

diff=1

4

79

5

9

-1

19

Before explaining why, it will be useful to explain what each function does. minimart compares the values that the two int pointers a and b point to and the pointer of the smaller value is returned. The swap1 function does absolutely nothing for the caller because the pointer parameters are passed by value rather than passed by reference, so after the swap1 function completes, nothing in the main function is changed. Finally, the swap2 function changes the values of the variables that are at the addresses of the pointers a and b passed to the function.

Now, looking at the main function, first an int array called array with values { 5, 3, 4, 17, 22, 19 }. Then, the minimart function is called to compare the first and third elements of array, which is assigned to an int pointer variable called ptr. Since the value of the third element is smaller than the value of the first, ptr is set to array[2]. Next, the second value of the ptr array (ptr[1]) is set to 9, which means array is now changed to { 5, 3, 4, 9, 22, 19 }. Next, ptr is shifted two addresses down so that it now points to the address of the 5th element of array, which is array[4]. ptr is dereferenced and assigned to -1, so array is now changed to { 5, 3, 4, 9, -1, 19 }. Next, the value of the array element located at (array+1), which is also array[1], is changed to 79, so array is now changed to {5, 79, 4, 9, -1, 19 }. At this point, ptr is at the address of array[4], so when &array[5] subtracts ptr, it should just be 1. That’s why the first output line is “diff=1”. Next, swap1 is called, but as mentioned above it doesn’t actually change anything. Then swap2 is called with array and &array[2] as parameters, which switches the values of the first and third elements of array, so that array is now changed to { 4, 79, 5, 9, -1, 19 }. Finally, each element of array is printed out on separate lines in order from left to right, which is why the output looks the way it does.

If this explanation was unclear, I pasted the code with comments in bold that explains each step below:

int\* minimart(int\* a, int\* b) **//returns pointer that points to smaller**

{ **//value**

if (\*a < \*b)

return a;

else

return b;

}

void swap1(int\* a, int \*b) **//switch pointer addresses--does nothing**

{ **//because pointers are passed by value not**

**//by reference**

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int \*b) **//switch int values of pointer addresses**

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

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

ptr[1] = 9; **//array changed to { 5, 3, 4, 9, 22, 19 }**

ptr += 2; **//ptr = array[4]**

\*ptr = -1; **//array changed to { 5, 3, 4, 9, -1, 19 }**

\*(array + 1) = 79; **//array changed to { 5, 79, 4, 9, -1, 19 }**

cout << "diff=" << &array[5] - ptr << endl; **//prints out 1**

swap1(&array[0], &array[1]); **//array = { 5, 79, 4, 9, -1, 19 }**

swap2(array, &array[2]); **//array = { 4, 79, 5, 9, -1, 19 }**

for (int i = 0; i < 6; i++)

cout << array[i] << endl;

return(0);

}

4.

void deleteCapitals(char \* p) {

char \* i = p;

while (\*p != '\0') {

if (isupper(\*p)) {

i = p;

while (\*i != '\0') {

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

i++;

}

} else {

p++; //only move forward if there are no more

//capital letters

}

}

}