Shaan Mathur

Stahl

CS31

March 2, 2016

Project 6 Homework

//Problem 1: This problem involves errors related to the use of pointers.

//a. This program is supposed to write 1000 100 10 1 , but it doesn't. Find all of the bugs and show a fixed version of the 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 **/\*It didn’t use parentheses**

**Before. I added them.\*/**

ptr += 2;

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

ptr[1] = 1000; // set arr[3] to 1000

ptr += 1;

while (ptr >= arr)

{

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

ptr--; **//It had decremented first before. I swapped the**

**//order so we wouldn’t accidentally try to //access unallocated memory**

}

cout << endl;

return(0);

}

//b. The findLastZero function is supposed to find the last element in an array whose value is zero, and sets the parameter p to point to that element, so the caller can know the location of that element holding the value zero. Explain why this function won't do that, and show how to fix it. Your fix must be to the function only; you must not change the main routine below in any way. As a result of your fixing the function, the main routine below needs to work correctly.

**/\*The issue is that the pointer int\* p was not passed by reference. In the function locally, it existed fine and well and found the last zero. But as soon as the function ended, the pointer fell out of scope and was garbage collected by C++. The main function wouldn’t have ptr changed, unless you make the findLastZero signature include int\*& p.\*/**

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

{

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. The biggest function is correct, but the main function has a problem. Explain why it may not work, and show a way to fix it. Your fix must be to the main function only; you must not change the biggest function in any way.

**/\*In the main function, p is uninitialized with no real value. With normal variables, it would at least have some garbage value in the stack. In the biggest function, the resultPtr is dereferenced. Since p doesn’t even have a real memory address, C++ would be unable to access any real memory location. Thus, there is a compile error because p isn’t initialized. A simple fix is to give p a valid address to an arbitrary int.\*/**

#include <iostream>

using namespace std;

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

{

if (value1 > value2)

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

int x = 0;

int\* p = &x;

biggest(15, 20, p);

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

return(0);

}

//d. The match function is supposed to return true if and only if its two C-string arguments have exactly same text. Explain what the problems with the implementation of the function are, and show a way to fix them.

**/\*The programmer mistakenly used the pointers themselves for evaluating boolean expressions. This results in the comparison of memory locations, not the values pointed to by these pointers. Instead, they must dereference the pointers and specify a specific index to examine, as implemented below.\*/**

// return true if two C strings are equal

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

{

bool result = true;

int i = 0;

while (str1[i] != ‘\0’ && str2[i] != ‘\0’) // zero bytes at ends

{

if (str1[i] != str2[i]) // compare corresponding characters

{

result = false;

break;

}

i++; // advance to the next character

}

if (result)

{

result = (str1[i] == str2[i]); // 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. This program is supposed to write 1 1 2 3 5 8 13 21 but it probably does not. What is the program doing that is incorrect? (We're not asking you explain why the incorrect action leads to the particular outcome it does, and we're not asking you to propose a fix to the problem.)

**/\*You cannot return an array created locally in a function via a pointer, as they try to do at the end of computeFibonacciSequence(int& n). The pointer returned will point to a location in memory that has been erased in the stack once the function itself ends. If the array had been made in the heap using the keyword new, however, the main function would be able to view it.\*/**

#include <iostream>

using namespace std;

int fibonacci(int n)

{

int tmp;

int a = 1;

int b = 1;

for (int i = 0; i < n - 2; i++)

{

tmp = a + b;

a = b;

b = tmp;

}

return b;

}

int\* computeFibonacciSequence(int& n)

{

int arr[8];

n = 8;

for (int k = 0; k < n; k++)

{

arr[k] = fibonacci(k);

}

return arr;

}

int main()

{

int m;

int\* ptr = computeFibonacciSequence(m);

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

{

cout << ptr[i] << ' ';

}

return(0);

}

//Problem 2: Match each of the following statements with the explanation of what the statement does. (NOTE: One of the descriptions listed below actually matches two statements)

1. string\* fp;

f. declares a pointer variable to point to a variable of type string

2. string fish[ 5 ];

g. declares a five element array of string

3. fp = &fish[ 4 ];

a. sets a pointer variable to the last element of an array of five strings

4. \*fp = "salmon";

b. sets the string pointed to by a pointer variable to the value "salmon"

5. fp -= 3;

d. moves the pointer fp back three strings in the array it points to

6. \*(fp + 3) = "salmon";

c. sets the fourth element of an array pointed to by the variable fp to the value "salmon"

7. fp[ 0 ] = "salmon"

b. sets the string pointed to by a pointer variable to the value "salmon"

8. bool b = (fp == fish);

e. initializes a boolean to true if the pointer variable fp points to the string at the start of

the fish array, false otherwise

9. bool b = (\*fp == \*(fp + 1));

h. initializes a boolean to true if fp points to a string whose value matches the string

immediately following the string pointed to by fp,    
false otherwise

Problem 3: Following the supplied instructions, rewrite each function provided to you below so that it returns the same result but performs the operation in a slightly different way.

a. Rewrite the following function so that it does not increment the variable ptr. Your new program must not use any square brackets, but must use an int variable to visit each double in the array. You may eliminate any unneeded variable.

double computeAverage(const double\* scores, int nScores)

{

int i = 0;

double total = 0;

while (i < nScores)

{

total += \*(ptr + i);

i++;

}

return total / nScores;

}

//b. Rewrite the following function so that it does not use any square brackets (not even in the parameter declarations) but does use the integer variable k. Do not use any of the C-string functions such as strlen, strcpy, etc.

// This function searches through str for the last occurrence of the character chr.

// If the chr is found at all, it returns a pointer into str where the character was last found.

// Otherwise, it returns the value nullptr to designate that the character chr not found.

const char\* findTheLastChar(const char\* str, char chr)

{

const char \* result = nullptr;

for (int k = 0; \*(str + k) != ‘\0’; k++)

{

if (\*(str + k) == chr)

{

result = &\*(str + k);

}

}

return result;

}

//c. Now rewrite the function shown in part b so that it uses neither square brackets nor any integer variables. Your new function must not use any local variables other than the parameters.

const char\* findTheLastChar(const char\* str, char chr){

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

if (\*str == chr) {

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

str++;

}

while (\*str != chr) {

str--;

}

return str;

}

str++;

}

return nullptr;

}

//Problem 4: What does the following program print and why? Be sure to explain why each line of output prints the way it does to get full credit.

Look at comments for answer to problem.

#include <iostream>

using namespace std;

int\* minimart(int\* a, int\* b)

{

if (\*a < \*b) **//Returns pointer which points to a lower value**

return a;

else

return b;

}

void swap1(int\* a, int \*b) **//Swaps pointers local to function. Nothing is affected**

{ **//in main function because this was pass by value**

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int \*b) **//Swaps values that pointers a and b are pointing to**

{ **//by DEREFERENCING the values**

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = minimart(array, &array[2]); **//Pointing to 4**

ptr[1] = 9; **//Makes array[3] = 9**

ptr += 2; **//Moves pointer to 22**

\*ptr = -1; **// makes array[4] = -1**

\*(array + 1) = 79; **//makes array[1] = 79**

**//Current state: array = {5, 79, 4, 9, -1, 19}**

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

**// are 1 index away from each other**

swap1(&array[0], &array[1]); **//doesn’t do anything because pass by value**

swap2(array, &array[2]); **//array = {4, 79, 5, 9, -1, 19}. Switched //array[0] and array[2] properly because //pointers are dereferenced before swapping in //implementation**

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

cout << array[i] << endl; **//Prints all members of {4, 79, 5, 9, -1, 19}**

**//on new lines**

return(0);

}

**Prints:**

**diff=1**

**4**

**79**

**5**

**9**

**-1**

**19**

**Justification and Tracing are in comments of the code.**

//Problem 5: Write a function named deleteCapitals that accepts one character pointer as a parameter and returns no value. The parameter must be a C-string. This function must remove all of the upper case letters from the string. The resulting string must be a valid C-string.

//Your function must declare no more than one local variable in addition to the parameter; that additional variable must be of a pointer type. Your function must not use any square brackets and must not use the strlen or strcpy library functions.

int main()

{

char msg[100] = "Happy Days Are Here Again!";

deleteCapitals(msg);

cout << msg << endl; // prints: appy ays re ere gain!

}

void deleteCapitals(char\* p) {

char\* p2 = p;

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

while (\*p2 >= 'A' && \*p2 <= 'Z') {

p2++;

}

\*p = \*p2;

p++;

if(\*p2 != '\0')

p2++;

}

}