Ashley Chang

504449890

Project 6: Get the Point?

1. The subparts to this problem involve errors in the use of pointers.

1. This program is supposed to write **30 20 10**, but it doesn't. Find all of the bugs and show a fixed version of the program:

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

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

\*(ptr + 1) = 20; // set arr[1] to 20

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

int i = 0;

while (i < 3)

{

cout << \*ptr << ' '; //ptr points to array[2]

ptr--;

i++;

}

cout << endl;

}

b. The findDisorder function is supposed to find the first item in an array that is less than the element preceding it, and set the p parameter to point to that item, so the caller can know the location of that item. 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 the main routine below in any way, yet as a result of your fixing the function, the main routine below must work correctly.

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

{

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

int main()

{

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

int\* ptr;

findDisorder(nums, 6, ptr);

if (ptr == nullptr)

cout << "The array is ordered" << endl;

else

{

cout << "The disorder is at address " << ptr << endl;

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

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

}

}

The original function will not perform properly because the parameter p is passed by value into the function. Thus, after findDisorder is called in the main function, the variable k disappears and ptr is left unchanged/uninitialized. To fix this, I added an ampersand before the parameter p so that ptr can be passed by reference and have a value outside of the function. With this change, the function returns a value (arr+k) to store in the memory at ptr’s address, and the ptr in the main function is given a value to work with.

c. The hypotenuse 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 hypotenuse function in any way.

#include <iostream>

#include <cmath>

using namespace std;

void hypotenuse(double leg1, double leg2, double\* resultPtr)

{

\*resultPtr = sqrt(leg1\*leg1 + leg2\*leg2);

}

int main()

{

double x;

double\* p = &x ; //Need to initialize the pointer to an address

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

The main function does not work because the pointer p needs to be initialized to hold the address of some variable. Otherwise, when the hypotenuse function is run, resultPtr tries to access something undefined; it does not change any value outside the function and there is no effect. To fix it, I created a new double x for the pointer p to point to, so when the hypotenuse function is run, it can store the result in x.

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.

// return true if two C strings are equal

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

{

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

{

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

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a,b))

cout << "They're the same!\n";

}

The errors in the match function result primarily from a lack of dereferencing “\*” operators.

The conditions of the original while loop checked to see if the str1 and str2 pointers themselves had a value of 0, which is incorrect because the function is meant to look for zero bytes inside the C strings rather than the addresses held by the pointers. This could be fixed by adding a “\*” before the pointers, which would dereference the pointers and allow access to the values inside the C strings, and by changing the 0 to ‘\0’ to indicate C string zero bytes rather than null pointers.

Inside the while loop, the original if statement has the same dereferencing problem in that without the “\*” before str1 and str2, it compares the values in the pointers (addresses) rather than the values in the C string that the pointers are pointing to, so it would always return false.

The return line at the end of the function again does not dereference properly with “\*” and so checks if the address values held by the pointers are equal rather than if the values that the pointers point to are equal.

e. This program is supposed to write 1 4 9 16 25 36 49 64 81 100 , 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.)

#include <iostream>

using namespace std;

int\* computeSquares(int& n)

{

int arr[10];

n = 10;

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

arr[k] = (k+1) \* (k+1);

return arr;

}

void f()

{

int junk[100];

for (int k = 0; k < 100; k++)

junk[k] = 123400000 + k;

}

int main()

{

int m;

int\* ptr = computeSquares(m);

f();

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

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

}

The computeSquares function creates an array that is declared only inside the scope of the function and thus returns a pointer to an array that does not exist outside the function. When the function is called in the main function, the pointer points to an invalid location in memory. Similarly, the void f() function creates a local array that does not exist outside of the function and does not have any effect in the main function.

2. For each of the following parts, write a single C++ statement that performs the indicated task. For each part, assume that all previous statements have been executed (e.g., when doing part e, assume the statements you wrote for parts a through d have been executed).

1. Declare a pointer variable named fp that can point to a variable of type string.

string\* fp;

1. Declare fish to be a 5-element array of strings.

string fish[5];

1. Make the fp variable point to the last element of fish.

fp = fish+4;

1. Make the string pointed to by fp equal to "yellowtail", using the \* operator.

\*fp = “yellowtail”;

1. Without using the fp pointer, and without using square brackets, set the fourth element (i.e., the one at index 3) of the fish array to have the value "salmon".

\*(fish+3) = “salmon”;

1. Move the fp pointer back by three strings.

fp = fp - 3;

1. Using square brackets, but without using the name fish, set the third element (i.e., the one at index 2) of the fish array to have the value "cod".

fp[1] = “cod”;

1. Without using the \* operator, but using square brackets, set the string pointed to by fp to have the value "eel".

fp[0] = “eel”;

1. Using the == operator in the initialization expression, declare a bool variable named d and initialize it with an expression that evaluates to true if fp points to the string at the start of the fish array, and to false otherwise.

bool d = (fp == &fish[0]);

1. Using the \* operator in the initialization expression, but no square brackets, declare a bool variable named b and initialize it to true if the string pointed to by fp is equal to the string immediately following the string pointed to by fp, and false otherwise.

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

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

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

{

const double\* ptr = scores;

double tot = 0;

int i = 0;

while (i < nScores)

{

//use integer to visit each double in array

tot += \*(ptr+i); //adds value in pointer to total

i++;

}

return tot/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 <cstring> functions such as strlen, strcpy, etc.

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

{

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

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

return (str+k);

return nullptr;

}

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

{

while (\*str != 0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

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.

#include <iostream>

using namespace std;

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

{

if (\*a < \*b) //compares values that a & b point to

return a;

else

return b;

} //returns the lesser value

void swap1(int\* a, int \*b)

{

int\* temp = a;

a = b;

b = temp;

} //swaps address numbers...but doesn’t do anything outside of function!

void swap2(int\* a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

} //swaps values in the pointers

int main()

{

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

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

//function passes 5 and 4

//creates a pointer that stores the address of 4

ptr[1] = 9; //changes the value of 17 to 9

ptr += 2; //pointer moves to index 4 (5th element)

\*ptr = -1; //changes value of index 4 (5th element) to -1 (22🡪-1)

\*(array+1) = 79; //changes value of index 1 (2nd element) to 79 (3🡪79)

//{5, 79, 4, 9, -1, 19}

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

//address of index 5 - address of index 4 (where pointer is located)

swap1(&array[0], &array[1]); //NO EFFECT

swap2(array, &array[2]); //swaps 5 and 4

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

cout << array[i] << endl;

}

The first line of output prints 1 because the difference between the addresses of array[5] and ptr, which is located at index 4, is 1.

The remaining lines of output print:

4

79

5

9

-1

19

* The 4 is printed because the swap2 function swaps the values inside the 0th and 2nd indexes, which were 5 and 4 respectively.
* The 79 is printed because \*(array+1)=79 sets the value of the 1st index to 79.
* The 5 is printed because of the swap2 function, so 5 occupies the 2nd index where 4 previously was.
* The 9 is printed because of the ptr[1]=9 line. After calling the minimart function, ptr was set to hold the address of (the original placement of 4), which was at the 2nd index. ptr[1] increments the pointer to the next element (3rd index), and sets it to 9.
* The -1 is printed because after ptr +=2, ptr points to the 4th index. Then, \*ptr = -1 sets the value of the 4th index to -1.
* The 19 is printed because it is at the 5th index, and the for loop at the end of the main function prints out all the values in the array up from the 0th to the 5th index. 19 is unchanged from the original array.

5. Write a function named deleteG that accepts one character pointer as a parameter and returns no value. The parameter is a C string. This function must remove all of the upper and lower case 'g' 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.

void deleteG (char \* cString)

{

char \* charPtr = cString;

for ( ; \*cString != '\0'; cString++)

{

if (\*cString != 'g' && \*cString != 'G') //If non-G character, give same value

{

\*charPtr = \*cString;

charPtr++;

}

//If G character, then skip over without incrementing charPtr

}

\*charPtr = '\0'; //Invalidate extraneous characters

}

int main()

{

char msg[100] = "I recall the glass gate next to Gus in Lagos, near the gold bridge.";

deleteG(msg);

cout << msg; // prints I recall the lass ate next to us in Laos, near the old bride.

}