Artiom Arutiunov

ID: 504597668

3/2/16

Project 6

Problem 1

a.

#include <iostream>

#include <string>

using namespace std;

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 (bug:\*ptr + 1)

ptr += 2;

ptr[0] = arr[1] \* 10;

ptr[1] = 1000;

ptr = arr + 4; // shits pointer to the end of the array

while (ptr > arr) // got rid of equal sign to make sure loop works

{

ptr--;

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

}

cout << endl;

return( 0 );

}

b.

The issue with this function is that in the original version, the pointer int \* p was passed by value instead of reference, meaning that as soon as the program stopped running the new memory address of the pointer was deleted. All that had to be changed was adding an ampersand to make sure that the pointer’s new memory address was retained. Otherwise, the undefined pointer ptr that’s declared in the main will return an undefined value.

#include <iostream>

#include <string>

using namespace std;

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 function may not work because passing a normal pointer declared in the main through the function will lead to undefined behavior. This is because the pointer is not defined in the main. It will give a runtime error.

#include <iostream>

using namespace std;

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

{

if( value1 > value2 )

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

int p;

biggest(15, 20, &p);

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

return( 0 );

}

d.

There are several problems with this code. The first is that you want to compare the content of the pointers as opposed to their memory addresses in the first “while” condition. This also applies to the following “if” statement. Lastly, this principle applies to the last result assignment, when you want to ensure that the strings both ended at the same time. As the program was written in the spec, the function did not behave as expected because it was comparing memory addresses as opposed to the corresponding variables of the pointers.

#include <iostream>

using namespace std;

// 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. The problem with this program is that within the computeFibonacciSequence function, the pointer that is returned points to a set of values that or may not exist. Since the contents of the array that the function replaces are only local to computerFibonacciSequence, one potential way to fix this problem is to declare the array within the main function and change computerFibonacciSequence function to void.

Problem 2

1. F
2. G
3. A
4. B
5. D
6. C
7. B
8. E
9. H

Problem 3

a.

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

{

double total = 0;

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

{

total += \*scores;

scores++;

}

return total/nScores;

}

b.

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.

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

{

while (\*str != 0 && \*str != chr)

{

str++;

}

if (\*str == 0)

return nullptr;

while (\*str != 0)

{

str++;

}

while (\*str != chr)

{

str--;

}

return str;

}

int main()

{

// a couple of test cases…

char a[10];

char b[10];

strcpy(a, "abcda");

strcpy(b, "abcdz");

char \*ptr = &a[4];

char \*ptr2 = &b[4];

assert (findTheLastChar(a, 'a') == ptr);

assert (findTheLastChar(a, 'z') == nullptr);

assert (findTheLastChar(b, 'z') == ptr2);

}

Problem 4

The program prints out the following:

**diff=1**

**4**

**79**

**5**

**9**

**-1**

**19**

The first line that is printed out is “diff=1”, which comes from the main function line **cout << "diff=" << &array[5] - ptr << endl;**

This comes after the function minimart is called, which returns the memory address of array[2] (3rd element) and assigns it to pointer ptr. Then, the following element of where ptr is pointing (4th element of array) is changed by ptr[1] to 9, and then the pointer’s memory address is shifted by 2 so that it now points to the 5th element within array. The third element is changed to -1, and the second element of the array is changed to 79.

At this point, array contains the following contents:

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

The actual cout line mentioned above takes the difference of the memory address of the 6th element of the array minus the memory address of ptr, which is currently at the 5th element of the array. The difference between &array[5] and &array[4] yields 1.

Next, the following two functions are called:

**swap1(&array[0], &array[1]);**

**swap2(array, &array[2]);**

In the swap1 function, the memory address of array’s first element is passed and so is its second element. The address of array[0] is assigned to &array[1] and the address of array[1] is assigned to &array[0]. Since neither of these parameters are pass by reference, nothing is changed in the array.

In the swap2 function, the memory address of the array’s first element is passed and so is the array’s 3rd element. The contents of &array[2] are assigned to array[0], and the contents of array[0] are assigned to array[2]. At this point, the contents of array are the following:

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

Finally, the usual “for” loop outputs the contents of the array one by one, creating the following list:

**4**

**79**

**5**

**9**

**-1**

**19**

Problem 5

#include <iostream>

#include <cstring>

#include <cassert>

using namespace std;

void deleteCapitals(char \* ptr)

{

const char \* ptrbottom = ptr;

while (\*ptr != '\0')

{

if (isupper(\*ptrbottom))

{

ptrbottom++;

}

else

{

\*ptr = \*ptrbottom;

ptr++;

ptrbottom++;

}

}

}

int main()

{

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

deleteCapitals(msg);

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

}