UCLA ID: - 705299746

Name: - Avnish Sengupta

***Problem 1: -***

1. There was a bug in the while loop where it was looping beyond the array as it would increment one even when it was at the last index of the element. There was also a logic error where the pointer variable was incremented by 2 but there was nothing resetting it back to its normal form before the while loop.

#include <iostream>

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

ptr += 2;

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

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

ptr -= 3;

while (ptr < &arr[3]) // loop over the whole array

{

ptr++;

cout << " " << \*ptr; // print a single value

}

cout << endl;

return(0);

}

1. There was a bug in the program with regards to the pass-by-parameter scheme of the void function. The last parameter should be passed-by-reference and not value for the actual ptr variable to change and point at the 0 element of the array. The second bug is in the for loop where the “greater than and equal to” instead of the “greater than” operator is used, causing the loop to loop beyond the array and display undefined behavior.

#include <iostream>

#include <string>

using namespace std;

void findFirstZero(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

// since we want the first zero value, as we keeping looping, we'll get to the 0 that is behind us in the array...

{

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

}

}

}

int main()

{

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

int\* ptr = nullptr;

findFirstZero(nums, 6, ptr);

if (ptr == nullptr)

{

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

}

else

{

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

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

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

}

return(0);

}

1. There is only one bug in this program where since the pointer variable p is not initialized it might point to illegal parts of the computer’s memory. Since the program only changes the value of the referent and not the memory it’s pointing at, we need to initialize it to ensure it doesn’t point to illegal parts of the memory.

#include <iostream>

#include <string>

using namespace std;

#include <iostream>

using namespace std;

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

{

if (value1 < value2)

{

\*resultPtr = value1;

}

else

{

\*resultPtr = value2;

}

}

int main()

{

int a = 0;

int\* p= &a;

smallest(15, 20, p);

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

return(0);

}

1. The functions deals with c-strings that are essentially pointer variables pointing to the first element of the array. One of the major bugs in the program was using str1 or str2 to refer to the referent of the pointer variable instead of \*str1 or \*str2. The last bug is where the program refers to the null character as 0 instead of ‘\0’.

#include <iostream>

#include <string>

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;

}

}

1. Since the array is declared inside computeFibonacciSequence(int& n) and returns only the pointer variable pointing to the first element of the array, the rest of the array after the first element has local scope and dies off at the end of the program. So, when the program tries to access the referents in the contiguous memory locations after the first element, it returns random values.

***Problem 2: -***

#include <iostream> //normal pre-processor directives.  
 using namespace std;   
  
    int main( )  
    {  
        int x[ 5 ] = { 1, 2, 3, 4, 5 };  
        int \* p = x; //pointer variable now points in the same location as the array.  
        int i;   
  
        for (i = 0; i < 2; i++)  
        {  
             int temp = \*(p + i); //the value of the element in the ith index is stored in temp  
             \*(p + i) = \*(p + 4 - i); //the ith- last element is stored in the ith index.  
             \*(p +4 - i) = temp; //the ith index value is stored in ith-last element.  
        }   
 //the statements in the for-loop reverse the elements of the array.

        for (i = 0; i < 5; i++)  
        {  
             cout << x[i] << " "; // for loop prints out the elements of the array on the console screen.  
        }  
       cout << endl;  
       return( 0 ); //since main is a function of int return type, it needs to return a number.  
   }

**The above code prints out the following output: - 5 4 3 2 1**

**Reason: - Since the pointer variable \*p is pointing at the same location as the array x and since the elements of an array are contiguously placed after the 0th element, the array and the pointer variable have the same referents and are aliases of one another. So, the for loop ends up exchanging the positions of the 1st and 5th element, 2nd and 4th element which ends up reversing the order of the elements of the original array**

***Problem 3: -***

 #include <iostream>  
    using namespace std;

    int main()  
    {  
        double x = 5.999; //initializes value of variable x.  
        double \*y, \*z;  // Line A - why is the second \* required? Because y and z are both pointer variables.  
  
        y = &x; //y is now pointing to the location of x in memory.  
        z = y; //z is now pointing to the same location as y.  
  
        cout << x << " " << \*(&x) << " " << \*y << \*z << endl;

        return( 0 ); //since main is a function of int return type, it needs to return a number.  
    }

**The above code prints out the following output: - 5.999 5.999 5.9995.999**

**Reason: - y and z are pointer variables pointing to the location of x in memory. Therefore, \*(&x) (the referent at the address of x) = \*y (the referent of y) = \*z (the referent of z) = x = 5.999.**

**The second \* is required in line A as y and z are both pointer variables pointing at double values and when multiple pointer variables of the same type are being initialized, C++ syntax requires both of them to have the \* sign on them.**

***Problem 4: -***

#include <iostream>  
  using namespace std; 

    int main()  
    {  
        int track[ ] = { 10, 20, 30, 40 };  
        int \* ptr;  
  
        ptr = track; //ptr points at the same location as track[0] does.  
        track[1] += 30; //changes the value of the second element of the array to 50.  
        cout << \* ptr << " "; //prints out first element of the array.  
        \*ptr -= 10; //changes the first element of the array to 0.  
        ptr++; //ptr points at first element of the array.  
        cout << \* ptr << " "; //prints out the first element of the array.  
        ptr += 2; //ptr points at the last element of the array.  
        cout << \* ptr << " "; //prints out the last element of the array.  
        cout << track[ 0 ] << endl; //prints out the first element of the array.

        return( 0 ); //since main is a function of int return type, it needs to return a number.  
    }

**The above code prints the following output: - 10 50 40 0**

**Reason: - Since \*ptr and track have the same referents and are aliases of each other, any changes to either affects both of them. When the 1st index or second element of track is incremented by 30, the value increases from 20 to 50. Since the ptr variable is pointing to the first element of the array, it prints out 10 to the console screen at the first cout statement. The ptr variable is incremented by 1 and points at the second element of the array and therefore prints out 50 to the console screen at the second cout statement. The ptr variable is then incremented by 2 and points at the last element of the array, and therefore prints out 40 at the third cout statement. When the referent of ptr is decremented by 10, the first element of the array which is the referent of the ptr variable is reset from 10 to 0. So, when track[0] is printed in the final cout statement, it prints the first element of the array which is 0.**

***Problem 5: -***

#include <iostream>

#include <string>

using namespace std;

int main()

{

int num[5];

num[0] = 100;

num[1] = 90;

num[2] = 80;

num[3] = 70;

num[4] = 60;

for (int i = 0; i < 5; i++)

cout << num[i] << " ";

cout << endl;

return(0);

}

***Problem 6: -***

void deleteDigits(char a[]) {

char\* b= &a[0];

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

if (\*a == '0' || \*a == '1' || \*a == '2' || \*a == '3' || \*a == '4' || \*a == '5' || \*a == '6' || \*a == '7' || \*a == '8' || \*a == '9') {

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

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

a++;

}

a = b;

}

else {

a++;

}

}

}

int main()

{

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

deleteDigits(msg);

cout << msg << endl;

return 0;

}