Project 6 CS 31

1. A. int main()

{

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

int\* ptr = arr;

\*ptr = 30; // set arr[0] to 30

\*(ptr + 1) = 20; // set arr[1] to 20 *added parenthesis to index rather than add 1*

ptr += 2;

\* ptr = 10; // set arr[2] to 10 *changed to \*ptr because we already indexed to 2*

ptr = arr; // *set ptr position back to arr position so we start our while loop right*

while (ptr <= (arr+2)) //*flipped condition as ptr is less than arr to begin with*

{

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

ptr++; //*since we flipped conditions, increment ptr rather than decrement*

}

}

B. This function won’t work correctly because it is not changing any values outside of itself. The function implementation is correct, as it indexes through the array and finds values that are greater than the difference to pToMax and changes pToMax to the position of the array with the highest value. However, the value of pToMax is changed within the void function, and nothing about the original parameter passed is changed(in this case, ptr). So to fix this, you can change the function parameters from…

void findMax(int arr[], int n, int\* ptoMax)

TO

void findMax(int arr[], int n, int\* &pToMax)

In order to pass by reference thus changing the actual value of ptr and being able to find the address which the max is located.

C. The main function attempts to access the deference of int\* ptr but it is an unknown address. We can fix this by initializing the pointer before dereferencing, as shown below.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int start = 1;

int\* ptr = &start;

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

D. One problem with this function implementation is that you are comparing str1 and str2 the wrong way. You are comparing the memory addresses, which can be assigned to any value that we don’t know. You have to use the dereference operator before the local variable to ensure that we are “pointing” to the actual character array outside the function, and checking each of its elements. To fix this below, I changed all the comparisons to reflect the actual value held where str1 and str2 “point” and not their memory addresses.

bool strequal(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[15] = "Chang";

char b[15] = "Zhang";

if (strequal(a,b))

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

}

E. What the program is doing that is incorrect is returning nochange(anArray) when anArray is inside the scope of the getPtrToArray function and since the integer anArray gets wiped after completion of the function, the pointer that is generated from nochange points to a memory address that now holds an undefined value. Same thing with void f(), the junk values it creates would have a pointer that points to the memory addresses that previously held our desired information, but have been discarded. So this programs dependence on undefined behavior results in its incorrectness.

2.

A. double\* cat;

B. double mouse[5];

C. cat = &mouse[4];

D. \*cat = 25;

E. \*(mouse+3) = 54;

F. cat = cat - 3;

G. cat[1] = 42

H. cat[0] = 17;

I. bool d = (cat == mouse);

J. bool b = (\*cat == \*(cat+1));

3.

a. double mean(const double\* scores, int numScores)

{

replaceInc = 0;

double tot = 0;

while (replaceInc != numScores)

{

tot += \*(replaceInc + scores);

replaceInc++;

}

return tot/numScores;

}

B. // This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

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

{

for (int k = 0; \*(str + k) != 0; k++) // condition is as long as nullbyte isn’t found, keep looping

if (\*str == chr)

return str;

return nullptr;

}

C. // This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

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

{

do {

if (\*str == chr) {

return str;

}

str++;

}

while (\*str);

return nullptr;

}

4. #include <iostream>

using namespace std;

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

{

if (\*a > \*b) //compares actual values of what the parameters point to

return a; //returns memory address of a if the value in a is > value in b

else

return b; //returns memory address of b if the value in b is > value in a

}

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

{

int\* temp = a; //sets integer pointer to point to a’s memory address

a = b; // a’s memory address is = to b’s memory address

b = temp; // b’s memory address is equal to temps memory address

}

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

{

int temp = \*a; //integer value temp takes on value held at memory address a

\*a = \*b; // value at a is = to value at b

\*b = temp; // value at b is equal to temp

} //function swaps values

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 }; //declares integer array of size 6

Compares the actual values of array at index 0 and 2, so 5 and 4, then since 5 > 4, this returns the memory address of a, thus creating a pointer to a.

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

\*ptr = -1; //the value that ptr points (array[0] is assigned -1

ptr += 2; //the pointer is incremented 2 memory addresses, now points to array[2]

ptr[1] = 9; // array[3] becomes 9

\*(array+1) = 79; //array[1] becomes 79 (we haven’t been incrementing array)

//array = {-1, 79, 4, 9, 22, 19}... Indexes 0, 1, and 3 changed.

cout << &array[5] - ptr << endl; //(&array[5] - ptr) accesses the memory addresses of //array[5] and array[2], since array[2] is where ptr currently points. Since an array is contiguous //in memory, we know that regardless of the memory values (&array[5] - ptr) is equal to 3. This //line outputs 3 then newline.

swap1(&array[0], &array[1]); //Here we swap the memory addresses of array[0] and array[1] Does not change numerical values of array

swap2(array, &array[2]); //Here we swap the value held in array[0](pointer to array) with the

//value in array [2] Current Array: {4, 79, -1, 9, 22, 19}

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

cout << array[i] << endl; Outputs the array then a newline character

}

OUTPUT:

3 //(&array[5] - ptr) ptr == &array[2], contiguous in memory so the output is 3(5-2=3)

4 //swapped array[0] with array[2] when swap2(array, &array[2])

79 //\*(array+1) = 79 and the value of 79 was not altered with swaps

-1 //swapped array[0] with array[2] when swap2(array, &array[2])

9 //ptr[1] = 9, since ptr is at array[2], ptr[1] == array[3]

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

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

5.

void removeS(char\* arr){

char\* real = arr; //sets pointer real to where arr is pointing

for (real = arr; \*real != ‘\0’; ++real, ++arr){

//as long as zero byte isn’t reached, iterate both pointers position

while(\*real == ‘S’ || \*real == ‘s’){

//increments real so it “skips” over the ‘S’ or ‘s’ characters in arr

++real;

}

\*arr = \*real; //sets real(arr with all s’s removed) to arr’s value

}

\*arr = ‘\0’; //after all loops we can’t forget the nullbyte character on end of c string!

}