CS 31 – Project 6 – Jahan Kuruvilla Cherian 104436427

1.a) int main()

{

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

int\* ptr = arr;

\*ptr = 30;

**\*ptr + 1 = 20 This is a bug**

**ptr += 2; This is a bug**

ptr[0] = 10;

while (**ptr >= arr**) This is a bug in that it prints the wrong way around

{

**ptr--;**

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

}

}

MY FIXED SOLUTION:

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

ptr[2] = 10; // set arr[2] to 10

ptr = arr;

while (ptr <= arr + 2)

{

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

ptr++;

}

}

1.b) The problem with the function findMax is the pointer as the argument in the function. The function provides a call by value to the pointer and thus when its passed into the function in the main routine, there is no actual change in the pointer because a copy of the pointer is created. To fix this we need to pass the pointer as a call by reference to the pointer so that any change within the actual function itself to the pointer is also done to the actual pointer. The fix is:

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

1.c) The reason it doesn’t work is because the pointer variable is declared but never initialized and thus doesn’t particularly point to anything in the memory. What we must therefore do is create a variable that holds the number we wish to cube and then generate a pointer pointing to that address of the variable within the main routine. A fix would be:

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int k =5;

int\* ptr = &k;

computeCube(k, ptr);

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

}

1.d) The reason this program fails is because of the way the C strings are compared. What this program does is take in the parameters as arrays and then during comparison, think they’re comparing each element of the array, but in fact are just comparing the base addresses of the arrays, which in turn will always evaluate to false, because the addresses of two different arrays will always be different. There are two ways of fixing this program. The first continuing with the array style, but looping through the arrays separately and using square brackets to compare each character. The second more relevant fix would be using pointers, as follows:

The first mistake was passing in the functions as arrays, we fix this by passing in pointers to the arrays. The second mistake was comparing the entire array to check for the zero byte, we must instead dereference the pointers and check if their values point to the zero byte. The third mistake was comparing the base addresses of the two C strings, when instead we need to dereference and compare the values. The final mistake was to return the comparison of the addresses of the two C strings, when we must instead return the value they point to.

bool strequal(const char\* str1, const char\* str2)

{

while (\*str1 != 0 && \*str2 != 0)

{

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

return false;

str1++; // advance to the next character

str2++;

}

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

}

1.e) The main issue with this program is essentially the scope of the local variable anArray within the getPtrToArray function. anArray is a local variable to that function, and as such the pointer is initially pointing to the first element of that array, but when we exit the function, all the elements within anArray are destroyed and basically set to undefined values. The f() function is a junk function wherein some values from here could be set in the ‘destroyed’ anArray (because the values that now lie within anArray are random undefined values), and as such the reason this program fails is because it relies on undefined behavior and thus prints out some random values because of the scope of anArray.

2.a) double\* cat;

b.) double mouse[5];

c.) cat = mouse + 4;

d.) \*cat = 42;

e.) \*(mouse + 3) = 17;

f.) cat -= 3;

g.) cat[1] = 25;

h.) cat[0] = 54;

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

j.) bool d = cat == mouse;

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

{

const double\* ptr = scores;

double tot = 0;

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

tot +=ptr[i];

return tot/numScores;

}

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

{

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

if (\*str== chr)

return str;

return nullptr;

}

4.) #include <iostream>

using namespace std;

int\* maxwell(int\* a, int\* b) **//basically returns the pointer position based on which one is greater.**

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b) **//swaps the addresses of the values but has no effect on overall output.**

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) **//actually swaps the values.**

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 }; **//sets an array of size 6**

int\* ptr = maxwell(array, &array[2]); **//compares the value 5 and 4 and sets pointer position to the the base address of 5 (i.e &array[0])**

\*ptr = -1; **//sets the value of &array[0] to -1 instead of 5.**

ptr += 2; **//moves the pointer up to position &array[2].**

ptr[1] = 9; **//makes the value of &array[3] to 9 instead of 17**

\*(array+1) = 79; **//changes the value 3 to 79**

cout << &array[5] - ptr << endl; **//prints out the difference of 5 and 2, because ptr is pointing at &array[2]. This is similar to pointer arithmetic wherein we have &array[5] - &array[2] which is the same as (array + 5) – (array + 2) which is 5- 2= 3. Thus the output is 3, and a new line.**

swap1(&array[0], &array[1]); **//doesn’t have an overall effect on the output, but swaps the base addresses of &array[0] and &array[1]**

swap2(array, &array[2]); **//swaps the values at position 0 and 2 of the array, that is swapping the value-1 and the value 4.**

for (int i = 0; i < 6; i++) **//goes through the modified array**

cout << array[i] << endl; **//prints out each value of each element in the array one line at a time.**

}

The OUTPUT is as follows:

**3**

**4**

**79**

**-1**

**9**

**22**

**19**

5.) void removeS(char\* str)

{

char\* destination = str;

while(\*str) {

if(\*str == 's' || \*str == 'S') {

str++;

} else {

\*destination = \*str;

destination++;

str++;

}

}

\*destination = 0;

}