Sadhana Vadrevu

CS31

Professor Smallberg

Project 6

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

ptr += 2;

ptr[0] = 10; // set arr[2] to 10

while (**ptr >= arr**)

{

**ptr--;**

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

}

}

**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;

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

ptr = arr;

while (ptr < arr + 3)

{

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

ptr++;

}

}

b. The problem with the function findMax is the pointer as the argument in the function. The pointer was being passed by value, so there was no actual change happening to the pointer because a copy of the pointer was created. To fix this, we need to pass the pointer by reference so that the function is able to actually change the pointer in the main function.

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

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \*pToMax)

pToMax = arr + i;

}

}

c. This function won’t work correctly because ptr is not initialized. To fix this problem, we must create a variable that holds a number and then initialize the pointer to point to the address of that variable.

int main()

{

int a = 0;

int\* ptr = &a;

computeCube(5, ptr);

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

}

d. This program fails because of the way the C-strings are compared. This function takes in arrays as parameters but then compares only the base addresses of the arrays, which will never be equal to each other as the pointers are pointing to different things. This problem can be solved by comparing the values the pointers point to by using the \* operator. The first problem occurs in the while conditional statement. The condition needs to check whether the values str1 and str2 equal 0, indicating the end of a C-string, not whether the addresses equal 0. The next error is in the if statement condition. The values that pointers str1 and str2 point to need to be compared rather than the actual addresses, so the pointers need to be dereferenced in the comparison. The final error is in the return statement. The comparison of the values of the pointers should be returned, not the comparison of the addresses.

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?

}

e. The main issue with this program is the scope of the local variable anArray within the getPtrToArray function. anArray is a local variable to that function, so the pointer anArray is initially pointing to the first element of that array, but once we exit the function, all the elements within anArray are destroyed and are basically set to undefined values. This program fails is because it relies on undefined behavior and thus prints out some random values because of the scope of anArray.

1. a. double\* cat;

b. double mouse[5];

c. cat = &mouse[4];

d. \*cat = 25

e. \*(mouse + 3) = 42;

f. cat -= 3;

g. cat[1] = 27;

h. cat[0] = 54;

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

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

1. 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) **//returns the pointer position of the greater of the two variables**

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b) **//swaps the addresses of two values**

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) **//swaps the values that a and b are pointing to**

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); **//compares the values 5 (at position 0) and 4(at position 2) and sets ptr to the address of array (&array[0])**

\*ptr = -1; **//sets the value ptr is pointing to to -1 (array[0] = -1)**

ptr += 2; **//ptr now points to address of array[2] (&array[2])**

ptr[1] = 9; **//makes value of array[3] equal to 9**

\*(array+1) = 79; **//sets array[1] to 79**

**//array = {-1, 79, 4, 9, 22, 19}**

cout << &array[5] - ptr << endl; **//prints out the difference between the addresses of array[5] and ptr (&array[2]). &array[5] - &array[2] is the same as (array + 5) – (array + 2), which equals 3. The output is 3 and a new line**

swap1(&array[0], &array[1]); **//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 (array[0] = 4, array[2] = -1)**

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

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

}

**OUTPUT:**

**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;

}