CS 31 Project 6

Note that corrections are highlighted in the code.

1. a)

int main()

{

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

int\* ptr = arr;

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

\*(ptr + 1) = 20; // set arr[1] to 20

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

}

Explanation:

\*ptr + 1 = 20 should be \*(ptr + 1) = 20

ptr--; can be placed after cout in the loop

cout printed out the values vertically, so we can change it so cout << “ “ << \*ptr; so it prints out in one line

1. b)

void findDisorder(int arr[], int n, int\* &p)

{

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

{

if (arr[k] < arr[k - 1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

int main()

{

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

int\* ptr;

findDisorder(nums, 6, ptr);

if (ptr == nullptr)

cout << "The array is ordered" << endl;

else

{

cout << "The disorder is at address " << ptr << endl;

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

cout << "The item's value is " << \*ptr << endl;

}

}

Explanation:

The pointer needed to be passed by reference in order for the main function to point to the location in findDisorder. Changing p to &p allowed the program to thus run as intended.

1. c)

#include <iostream>

#include <cmath>

using namespace std;

void hypotenuse(double leg1, double leg2, double\* resultPtr)

{

\*resultPtr = sqrt(leg1 \* leg1 + leg2 \* leg2);

}

int main()

{

double n;

double\* p = &n;

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

Explanation:

In the main function, pointer p did not point to a location in memory because it was not set to point to a double. Thus, we needed a double in the main that p could point to.

1. d)

// return true if two C strings are equal

bool match(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 \*p1 == \*p2; // both ended at same time?

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a, b))

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

}

Explanation:

We needed each array assigned to a pointer since we are not able to directly compare arrays. To ensure that the values were being compared rather than the memory addresses of those values, we can also dereference our pointers.

1. e)

The main function is trying to assign int\* ptr to the output of the function computeSquares. The array that the function computeSquares returns exists only in that function, so calling it in the main function will not work because the array does not exist in the main and thus is undefined behavior.

2.

a) string\* fp;

b) string fish[5];

c) fp = &fish[4];

d) \*fp = "yellowtail";

e) \*(fish + 3) = "salmon";

f) fp -= 3;

g) fp[1] = "basa";

h) fp[0] = "sole";

i) bool d = (\*fp == fish);

j) bool b = (\*fp == \*(fp + 1));

3. a)

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

{

double tot = 0;

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

{

tot += \*(scores + k);

}

return tot / nScores;

}

3. 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++)

if (\*(str + k) == chr)

return (str + k);

return nullptr;

}

3. c)

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

{

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

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

4.

This program first declares an array of six integers {5, 3, 4, 17, 22, 19}, and then uses a pointer which it passes to the to position 0 (the first element of the array) and position 2 (the third element of the array) of that array to the minimart function. The minimart function returns the smaller value which for this array is 4. The main function will then assign an integer pointer to 4. Ptr[1] for this array refers to the fourth element of the array since we started at the third element from the minimart function. The main function sets the fourth element to 9. Ptr += 2; moves the pointer up two places to the fifth element, and \*ptr = -1 sets the pointer to point to the fifth element and set it to -1. \*(array + 1) refers to the second element of the array and sets it to 79. The main function will then print out the difference between the address, not the value, of the sixth element of the array (which is 5) and the address of the element the pointer points to which is the fifth element (which is 4). The output is 5-4 or 1. The main function will then pass the memory addresses of the first and second elements (so, positions 0 and 1) to the swap1 function. The swap1 function swaps the memory addresses of the first and second elements, but only affects the local variables a and b in the function. So, the array itself is not affected. The main function will then pass the memory addresses of the first and third elements (so positions 0 and 2) to the swap2 function. The swap 2 function will swap the values of these elements, setting 4 to position 0 and 5 to position 2. The main function will finally print out all the elements of the array vertically after having printed out the difference earlier:

diff=1

4

79

5

9

-1

19

5.

void deleteG(char s[])

{

char\* ptr = s;

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

{

if (\*ptr == 'g' || \*ptr == 'G')

{

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

{

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

ptr++;

}

ptr = s;

}

ptr++;

}

}