**Project 6 HW**

1a)

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--;

}

cout << endl;

}

1b) In the original function int\* p is passed by value. Therefore, although its value has been correctly determined at the end of the function, it is only stored in a local copy which cannot be accessed outside the function. Therefore, in the main routine, ptr’s value does not actually change. The correct way to implement the function would be passing int\* p by reference.

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 = &nums[0];

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;

}

}

1c) The function assumes that when resultPtr is passed to the function, it has been initialized to point to some double variable. However, the main routine does not do this. This can be fixed by declaring a double “result” and then initializing the pointer “p” that is being passed to the function by making it point to result.

#include <iostream>

#include <cmath>

using namespace std;

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

{

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

}

int main()

{

double result;

double\* p = &result;

hypotenuse(1.5, 2.0, p);

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

}

1d) One way to fix the function has been pasted below

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

}

We need to compare the *characters* so we write if (\*str1 != \*str2) not if (str1 != str2). And, at the end, we want to check if they ended at the same time so we need to check whether they both point to the zero byte.

1e) The computeSquares function returns a pointer to the element of a locally defined array. Since the array is locally defined, all its elements exist locally and not outside the function. So, after the function is called, the pointer that is returned exhibits undefined behavior since the variable it points to doesn’t exist outside the function.

Q2)

1. string\* fp;
2. string fish[5];
3. fp = &fish[4];
4. \*fp = "yellowtail";
5. \*(fish + 3) = "salmon";
6. fp -= 3;
7. fp[1] = "carp";
8. fp[0] = "smelt";
9. bool d = (fp == fish);
10. bool b = (\*fp == \*(fp + 1));

Q3a)

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

{

const double\* ptr = scores;

double tot = 0;

int i = 0;

while (i != nScores)

{

tot += \*(ptr+i);

i++;

}

return tot/nScores;

}

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){

while (\*str != '\0'){ // till the end of the c string

if (\*str == chr) return str;

str ++;

}

return nullptr;

}

Q4)

**Output:**

diff=1

4

79

5

9

-1

19

**Explanation:** I have copy pasted the code in the main routine and explained what each line of code does there, which in turn explains how each of the elements in the array are being altered and therefore explains the output. The comments (my explanation) have been emboldened.

int main()

{

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

int\* ptr = minimart(array, &array[2]); **// ptr points to 4 (3rd element of array) as minimart returns a pointer to the element that is smaller (or the second one if equal)**

ptr[1] = 9; **// the 4th element is now 9 (not 17)**

ptr += 2; **// pointer now points to the 5th element**

\*ptr = -1; **// change the 5th element to -1**

\*(array+1) = 79; **// the 2nd element is now 79**

cout << "diff=" << &array[5] - ptr << endl; **// &array[5] points to 6th element, ptr points to 5th element, so the difference is 1**

swap1(&array[0], &array[1]); **// doesn't do anything as the function only modifies local copies of pointers**

swap2(array, &array[2]); **// switches the value of the 1st and 3rd element. so first element is now 4, third element is now 5**

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

cout << array[i] << endl;

}

/\*

**In summary,**

**Diff = 1**

**1st elmt: 4**

**2nd elmt: 79**

**3rd elmt: 5**

**4th elmt: 9**

**5th elmt: -1**

**6th elmt: 19**

**Which is the output we see**

**\*/**

Q5)

void deleteG(char\* msg){

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

if(\*msg == 'g' || \*msg == 'G'){

char\* g\_pointer = msg;

while (\*(g\_pointer)!= '\0'){

\*g\_pointer = \*(g\_pointer+1);

if (\*g\_pointer == '\0') break;

// at the final iteration, even the zero byte is moved one position to the left so we want to check if g\_pointer is now the zero\_byte, and then break

g\_pointer++;

} msg--; // so that after the shift, the msg pointer still points correctly

}

msg++;

}

}