CS31 Project6

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--;

}

cout << endl;

}

b. Since the function findDisorder only creates a copy of pointer which points to the first element of nums and after its execution, the local copy p is deleted so the address of the pointer ptr hasn’t been change which is still &nums[0]. So, the program won’t get executed as expected. One way to fix it is to change int\* p to int\*& p, which pass the pointer ptr by reference. The fixed function is shown below:

**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**;

}

c. Since in the main function, the pointer to double p hasn’t been declared when passing it to the function hypotenuse, so the program will encounter a problem since any access to uninitialized pointers will be considered bad access by the compiler. One way to fix the main is declare pointer p before calling it:

**int** main()

{

**double** a = 0;

**double**\* p = &a ;

hypotenuse(1.5, 2.0, p);

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

}

d. The first problem of this function match is in the while loop, we should change string into pointer when comparing with the nullptr, so we should add \* before str1. Next, if we want to compare corresponding characters, we should also add \* before str1 and str2. Exiting the while loop, since array comparison always evaluate to false, we should change array comparison into pointer comparison by adding \* to both str1 and str2. Here’s the modified function:

**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++;

// cout << \*str1 << " " << \*str2 << " " ;

}

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

}

e. Since the array defined in the function is a local variable, and as the function call gets over, the memory of that variable does not exist anymore.

2. **int** main () {

string\* fp;

string fish[5];

fp = fish + 4;

\*fp = "yellowtail";

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

fp -=3;

fp[1] = "carp";

fp[0] = "smelt";

**bool** d;

**if** (fp == &fish[0])

d = **true**;

**else**

d = **false**;

**bool** b;

**if** (\*fp == \*(fp +1))

b = **true**;

**else**

b = **false**;

}

3.a. **double** computeAverage(**const** **double**\* scores, **int** nScores){

**const** **double**\* ptr = scores;

**double** tot = 0;

**for** (**int** i = 0; i < nScores; i++)

{

tot += \*(ptr + 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)

{

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

str ++;

};

**return** **nullptr**;

}

4.Program’s output:

diff=1

4

79

5

9

-1

19

Program ended with exit code: 0

// Let’s analyze the main function. The first line **int** array[6] = { 5, 3, 4, 17, 22, 19 }; generate an int array consisting of six integers.

Then pointer to int ptr get assigned with value &array[2] since the function minimart return a pointer which value it’s pointing to is smaller (4<5).

ptr[1] = 9 assign the value next to ptr as 9. So array[6] now is {5,3,4,9,22,19}/

ptr += 2; push the pointer ptr pointing to integers two positions after the original position (now points to 4th).

\*ptr = -1 assigns the 4th position with value -1. So array[6] now is {5,3,4,9,-1,19}

\*(array +1) = 79 assigns the first position with value of 79 . So array[6] now is {5,79,4,9,-1,19}

Since ptr points to the 4th postion, the difference between pointer &array[5] and ptr is 1.

Swap1 function only swaps pointers, so the value will remain the same.

Swap2 function swaps the first and third values so the first value will become 4 and the third become 5. So the array becomes {4,79,5,9,-1,19}

Finally the for loop prints the value of the array.

5. **void** deleteG(**char**\* msg){

**char** \*newMessage;

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

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

newMessage = msg;

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

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

newMessage++;

}

}

**else**{

msg++;

}

}

}