/\* Assignment: Array Wordout with Pointers

Author: Ryan Wood

Created On: February 8, 2018

Purpose: demonstrate profficiency in using pointers with arrays

\*/

#include <iostream>

#include <fstream>

using namespace std;

ofstream outFile("arry.out");

/\* Function: \*table\_contain(int\*, int, int)

Purpose: determines whether the value given as

the third argument is located ibn the

given array, which must be of the

given size, returning the location

at which the value is found or NULL

Parameters: the array, the size of the array,

the value to search for

Return: a pointer to the location in the

array where the value was found or NULL

\*/

int \*table\_contain(int \*arry, int size, int val)

{

int \*iter;

int count = 0;

bool found = false;

iter = arry;

while(count <= size && !found)

{

if(\*iter == val)

{

found = true;

}

else

{

iter++;

count++;

}

}

if(!found)

iter = NULL;

return iter;

}

/\* Function: table\_sum(int\*, int)

Purpose: determines the sum of all elements in

the given array of the given size

Parameters: the array, the size of the array

Return: the sum of all elements

\*/

int table\_sum(int \*arry, int size)

{

int \*iter;

int sum = 0;

int count = 0;

iter = arry;

while(count < size)

{

sum += \*iter;

iter++;

count++;

}

return sum;

}

/\* Function: table\_avg(int\*, int)

Purpose: determines the average of all the

values in the given array of the given size

Parameters: the array, the size of the array

Return: the average of the values

\*/

double table\_avg(int \*arry, int size)

{

int sum = 0;

double average = 0.0;

sum = table\_sum(arry, size);

average = (double) sum/size;

return average;

}

/\* Function: table\_max(int\*, int)

Purpose: determiens what value in the given array of

the given size is the largest value

Parameters: the array, the size of the array

Return: the largest element

\*/

int table\_max(int \*arry, int size)

{

int \*iter;

int largest = 0;

int count = 0;

largest = \*arry;

iter = arry;

while(count < size)

{

if(largest < \*iter)

{

largest = \*iter;

}

iter++;

count++;

}

return largest;

}

/\* Function: table\_min(int\*, int)

Purpose: determines the smalles value in the given

array of the given size

Parameters: the array, the size of the array

Return: the smalles value in the array

\*/

int table\_min(int \*arry, int size)

{

int \*iter;

int smallest = 0;

int count = 0;

smallest = \*arry;

iter = arry;

while(count < size)

{

if(\*iter < smallest)

smallest = \*iter;

iter++;

count++;

}

return smallest;

}

/\* Function: table\_copy(int\*, int\*, int)

Purpose: copies all values of the array given as the first

argument into the array given as the second argument,

both of which must be of the given size

Parameters: the copy from array, the copy to array, the size of the arrays

Precondition: the arrays must have had memory allocated. This function

will not create either array

\*/

void table\_copy(int \*arryA, int \*arryB, int size)

{

int count = 0;

int \*iter;

iter = arryA;

while(count < size)

{

\*arryB = \*arryA;

iter++;

count++;

}

}

/\* Function: table\_print\_rev(int\*, int)

Purpose: prints the given array from the last

element to the first element as a comma

delineated list of values.

Prints to the console and the output file

Parameters: the array, the size of the array

\*/

void table\_print\_rev(int \*arry, int size)

{

int \*iter;

iter = &arry[size-1];

while(iter > arry)

{

cout << \*iter << ", ";

outFile << \*iter << ", ";

if(iter != arry)

iter--;

}

cout << \*iter;

outFile << \*iter;//write last val with no comma

}

/\* Function: table\_add1(int\*, int)

Purpose: increments each value in the given array

of the given size by one

Parameters: the array, the size of the array

\*/

void table\_add1(int \*arry, int size)

{

int \*iter;

int count = 0;

iter = arry;

while(count < size)

{

\*iter+=1;

iter++;

count++;

}

}

/\* Function: table\_fill(int\*, int, int)

Purpose: fills the given array of the given size with the

given value. All values from the given array start pointer

to the value at start+size are given the value

Parameters: the array (which may be a location pointer later in the array),

the size (which does not have to be the total size, may be smaller,

but must be less than the number of elements from start to start+size),

the value to fill the array elements with

Precondition: the array must have had memory allocated already. This function will

not create the array. Passing a NULL pointer as the first argument

will cause the program to crash

\*/

void table\_fill(int \*arry, int size, int num)

{

int \*iter;

int count = 0;

iter = arry;

while(count < size)

{

\*iter = num;

iter++;

count++;

}

}

/\* Function: table\_print(int\*, int)

Purpose: prints the given array of the given size

to both the console and the output file

as a comma delineated list of values

Parameters: the array, the size of the array

\*/

void table\_print(int \*arry, int size)

{

int \*iter;

int count = 0;

if(NULL == arry)

{

cout << "null array" << endl;

outFile << "null array" << endl;

return;

}

iter = arry;

while(count < size-1)

{

cout << \*iter << ", ";

outFile << \*iter << ", ";

iter++;

count++;

}

cout << \*iter;//write the last value with no comma

outFile << \*iter;//write the last value with no comma

}

int main()

{

int \*studentGrades = NULL;

int \*p = NULL;

int \*s = NULL;

int \*tp = NULL;//temp pointer

int \*loc = NULL;//pointer for holding location of found element

int Max = 0;//num values in array

int max = 0;//max value in array

int min = 0;//minimum value in array

int count = 0;//for looping

int vals = 0;//num vals less than average

int numGrades = 0;

double avg = 0.0;//the average of the values

Max = 20;

p = new int[Max];

s = new int[Max];

table\_fill(p, Max, 10 );

cout << endl << "Fill array with 10 " << endl;

outFile << endl << "Fill array with 10 " << endl;

table\_print(p, Max);

cout << endl << endl;

outFile << endl << endl;

cout << endl << "Add 1 to first 10 elements " << endl;

outFile << endl << "Add 1 to first 10 elements " << endl;

table\_add1(p, 10);

table\_print(p, Max);

cout << endl << endl;

outFile << endl << endl;

tp = s;// save pointer

cout << "tp = s" << endl;

outFile << "tp = s" << endl;

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

{

cout << i << " ";

outFile << i << " ";

table\_fill(s, 1, i);

s++;

}

cout << endl << "Fill array with 1 to 20 " << endl;

outFile << endl << "Fill array with 1 to 20 " << endl;

s = tp; // restore pointer

table\_print(s, Max);

cout << endl << endl;

outFile << endl << endl;

cout << endl << "Print reverse order " << endl;

outFile << endl << "Print reverse order " << endl;

table\_print\_rev(s, Max);

cout << endl << endl;

outFile << endl << endl;

table\_fill(p, Max, 0);

cout << endl << "Zero out array " << endl;

outFile << endl << "Zero out array " << endl;

table\_print(p, Max);

cout << endl << endl;

outFile << endl << endl;

s = p; // What's happening here?

for (int i=Max; i>=0; i--)

{

table\_add1( s, i );

}

cout << endl << "Fill array with ???? " << endl;

outFile << endl << "Fill array with ???? " << endl;

table\_print( p, Max );

cout << endl << endl;

outFile << endl << endl;

cout << endl << " Do we have an 8 in the array?" << endl;

outFile << endl << " Do we have an 8 in the array?" << endl;

if(table\_contain( p, Max, 8))

{

cout << " There is an 8 in the array " << endl;

outFile << " There is an 8 in the array " << endl;

}

else

{

cout << " There is not an 8 in the array " << endl;

outFile << " There is not an 8 in the array " << endl;

}

cout << endl << "Print reverse order " << endl;

outFile << endl << "Print reverse order " << endl;

table\_print\_rev( p, Max );

cout << endl << endl;

outFile << endl << endl;

for(int i = 0; i < Max / 2 + 1; i++)

{

s = p+10+i;

table\_fill(s, 1, i \* 2);

s = p+10-i;

table\_fill(s, 1, i \* 3);

}

cout << endl << "Fill array with <-> " << endl;

outFile << endl << "Fill array with <-> " << endl;

table\_print(p, Max);

cout << endl << endl;

outFile << endl << endl;

cout << endl << " Do we have a 20 in the array?" << endl;

outFile << endl << " Do we have a 20 in the array?" << endl;

if(table\_contain(p, Max, 20))

{

cout << " There is a 20 in the array " << endl;

outFile << " There is a 20 in the array " << endl;

}

else

{

cout << " There is not a 20 in the array " << endl;

outFile << " There is not a 20 in the array " << endl;

}

max = table\_min(p, 15);

min = table\_max(p+10, 9);

avg = table\_avg(p+5, 12);

cout << endl << "Min for previous table ";

cout << endl << "Max for previous table ";

cout << endl << "Average for previous table " << avg << endl;

outFile << endl << "Min for previous table ";

outFile << endl << "Max for previous table ";

outFile << endl << "Average for previous table " << avg << endl;

/\*cannot seem to get the following to execute without a segment fault error\*/

table\_fill(p, Max, 5);

table\_fill(s, Max, 10);

for (int i = 0; i < Max/2; i++)

{

cout << i;

table\_fill(table\_contain(p+i\*2, 1, 5), 1, i\*i);

}

cout << " \t\t P array with sum changes: ever other one \n";

outFile << " \t\t P array with sum changes: ever other one \n";

table\_print(p, Max);

cout << endl;

outFile << endl;

cout << "\n\n\n";

outFile << "\n\n\n";

//................

//Add your code here

//................

//get the average value

avg = table\_avg(p, Max);

cout << avg << endl;

outFile << avg << endl;

/\*loop backward from avg, one integer at a time to the

lowest value in the array, checking to see

whether the value is in the list.

If the value is in the list, increment a counter.

When loop is done, count is the number of values below avg.\*/

//assumes only positive values are in the array

for(count = avg; count >= min; count--)

{

if(table\_contain(p, Max, avg))

vals++;

}

cout << "Number of values in array less than average = " << count << endl;

outFile << "Number of values in array less than average = " << count << endl;

numGrades = 15;

studentGrades = new int[numGrades];

loc = studentGrades;//we'll use loc as an iterator

table\_fill(loc++, 1, 79);

table\_fill(loc++, 1, 88);

table\_fill(loc++, 1, 85);

table\_fill(loc++, 1, 94);

//first test grade, counts twice

table\_fill(loc++, 1, 87);

table\_fill(loc++, 1, 87);

table\_fill(loc++, 1, 99);

table\_fill(loc++, 1, 85);

//second test grade, counts twice

table\_fill(loc++, 1, 88);

table\_fill(loc++, 1, 88);

table\_fill(loc++, 1, 85);

table\_fill(loc++, 1, 87);

//final, counts three times

table\_fill(loc++, 1, 77);

table\_fill(loc++, 1, 77);

table\_fill(loc, 1, 77);

table\_print(studentGrades, numGrades);

cout << endl;

outFile << endl;

avg = table\_avg(studentGrades, numGrades);

cout << "Student Average is " << avg << endl;

outFile << "Student Average is " << avg << endl;

outFile.close();

return 0;

}