Austin Smothers

CISP 430

Assignment 10

Hash all the potatoes

I feel like a potato after spending so much time on that perfect hash table

/\*

ImpHash.cpp

Author: Austin Smothers

Original: November 2017

Description:

A program which reads protein sequences from a file,

counts how many times each sequence occurs,

and allows the user to lookup if a protein is in the .txt file.

Hash Function:

H(key) = (first letter + 2\*last letter) % 40

\*/

/\*---------------------------------------------------------------------------------------

Header File Inclusion

---------------------------------------------------------------------------------------\*/

#include <iostream> //included for cin, cout, etc

#include <fstream> //included for file i/o

#include <string.h> //included for strcmp

#include <iomanip> //included for setw

#include <ctype.h> //included for tolower

using namespace std;

/\*---------------------------------------------------------------------------------------

Structure Definitions

---------------------------------------------------------------------------------------\*/

struct arrayelement {

char protein[30]; //None of the proteins are more than 30 chars

int count; //counts how many times a protein occurs

};

arrayelement proteins[40]; //stores the proteins. Less than 20 are unique

/\*---------------------------------------------------------------------------------------

Function Prototypes

---------------------------------------------------------------------------------------\*/

void readfile(void); //reads the proteins.txt file into the protein array

void outTable(void); //outputs the protein structure

void searchTable(void); //gets user input. searches the protein table for it

char inSingle(void); //gets a single character as input and validates it

/\*---------------------------------------------------------------------------------------

Global Data Declarations

---------------------------------------------------------------------------------------\*/

char userIn[30] = { 0 }; //used to store user input to compare to protein data

int i = 0; //loops control variable

char emp[30] = { 0 }; //an empty char array to use for comparisons

/\*---------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------

THE MAIN FUNCTION

-----------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------\*/

void main()

{

readfile(); //read the file into the protein structure

outTable(); //output the proteins and their counts

searchTable(); //allow the user to search for a protein

}

/\*---------------------------------------------------------------------------------------

Function Definitions

---------------------------------------------------------------------------------------\*/

/\*Reads the protein.txt file into the protein array and stores

the number of times that an element occurs within the file\*/

void readfile()

{

//------------------------Local Variables-----------------------------------------

char buffer[30]; //a temporary storage for strings read from the file

int hash = 0; //a variable used to obtain hash keys

int j = 0; //stores location of last element of buffer

int first = 0; //stores the integer value of buffer's first element

int last = 0; //stores the integer value of buffer's last element

//------------------Check for & open the file-------------------------------------

ifstream fin("proteins.txt"); //open the text file to be used

if (!fin) { //if the file cannot be opened, output an error

cerr << "Input file could not be opened.\n";

exit(1); //after outputing an error, quit the program

}

//---Read the file line by line and store data in the protein structure-----------

while (fin >> buffer) { //while there are strings to input

//the below for loop will find the last element of the string being read

for (int i = 0; buffer[i] != NULL; i++)

j = i;

//obtain the next hash key. we use -65 to change A-Z from 65-91 -> 0-25

first = (int)buffer[0] - 65; //convert buffer[0] to an integer

last = (int)buffer[j] - 65; //convert buffer[j] to an integer

hash = ((first + (2 \* last)) % 40);

i = 1; //set i to 1 to loop the next while loop "forever"

while (i) {

//if the key already exists in the hash table, and our input = the stored string

if (strcmp(proteins[hash].protein, buffer) == 0) {

//increment count. We found a duplicate

proteins[hash].count++;

break; //break the while loop

}

/\*if the key does not exist, it must be new. Put it in the table.

We can tell because if they key exists, hash will be 1 or greater

which evaluates to true. Negating this means if only executes if count = 0\*/

if (proteins[hash].count == 0) {

//copy buffer into the hash table, one element at a time

for (i = 0; i < 30; i++) {

proteins[hash].protein[i] = buffer[i];

}

//increment count. This element occurred

proteins[hash].count++;

break; //break out of infinite while loop

}

//if we reach this point, they key exists, but the protein != buffer

if (hash == 40)

hash = 0; // reset hash if it gets to 40

else

hash++; //increment the hash. We found a collision

}

}

fin.close(); //close the proteins file. We don't need it anymore

}

/\*Outputs each element of the protein table, as

well as the number of times each element occured\*/

void outTable()

{

//--Read the table line by line. Output data & number of times it occurs----------

cout << setw(30) << "Proteins:" << " " << "Count:" << endl;

for (int i = 0; i < 40; i++) { //loop through each element of protein structure

/\*if the element contains a protein chain, output it and the count.

we compare using userIn because it's empty at this point\*/

if (!(strcmp(proteins[i].protein, emp) == 0)) {

cout << setw(30) << proteins[i].protein << " "

<< proteins[i].count << endl;

}

}

setw(0); //reset setw()

}

/\*Takes user input and searches the protein table for it.

If found, it will output the element and the number of times it occurs

If not found, it will tell the user that their protein is not in the table\*/

void searchTable()

{

//------------------------Local Variables-----------------------------------------

char failbit; //input regulator

//----------------------Search and Destroy!---------------------------------------

//create a while loop to iterate while the user wants to search for proteins

while (inSingle() == 'y') {

cout << setw(30) << "Please enter a sequence:";

cin.getline(userIn, 30); //get user input

cout << endl;

//iterate through the hash table to see if userIn is there

for (int i = 0; i < 40; i++) {

//if the user has input a null, return not found

if ((strcmp(userIn, emp) == 0)) {

//output not found because we shouldnt be looking for nulls

cout << setw(30) << "FOUND NOTHING. Don't do that again."

<< endl;

i = 40; //set i to 40 so we dont waste time looping

}

//user input is equivalent to the checked element in the table

else if (strcmp(proteins[i].protein, userIn) == 0) {

//output that it was found, and how many of them there are

cout << setw(30) << "FOUND " << proteins[i].count

<< endl;

i = 40; //set i to 40 so we dont keep checking

}

//if i has reached 39+, we have found no matches

else if (i >= 39)

cout << setw(30) << "NOT FOUND" << endl;

}

}

}

/\*Takes a single character (Y/N) as input from the user and validates it.

It is not case sensitive. If the user inputs something else\*/

char inSingle()

{

//------------------------Local Variables-----------------------------------------

char a; //stores user input

char failbit; //input regulator

//--------------------------Gets Input--------------------------------------------

cout << "Would you like to search for a protein sequence? (Y/N): ";

cin >> a;

//--------------------Validation of User Input------------------------------------

while ((a != 'Y' && a != 'y' &&

a != 'N' && a != 'n'))

{ //this loop validates user input

cin.clear(); //clear the input queue

cin.get(failbit); //reset failbit (eat the terminal NULL input)

cout << "\aError: Invalid response!\n";

cout << "Please enter your answer in Y/N "

"format: ";

cin >> a; //ask for new input

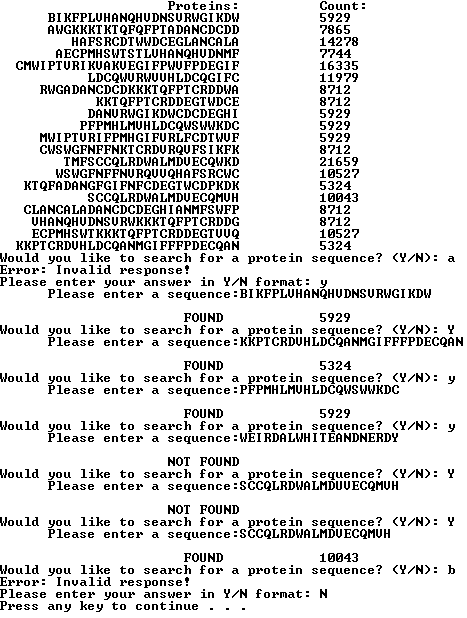
}

cin.clear(); //clear away any excess input

cin.get(failbit); //eat terminal NULL left by cin

return (tolower(a)); //return in lower case to avoid case sensitivity

}



/\*

PerfHash.cpp

Author: Austin Smothers

Original: November 2017

Description:

A program which reads keyword sequences from a file,

counts how many times each keyword occurs,

and allows the user to lookup if a keyword is in the .txt file.

This program also makes use of perfect hashing.

Perfect Hash Function:

H(key) = ((g[first\_letter] + (2 \* g[last\_letter]) + size\_of\_keyword) % 28)

\*/

/\*---------------------------------------------------------------------------------------

Header File Inclusion

---------------------------------------------------------------------------------------\*/

#include <iostream> //included for cin, cout, etc

#include <fstream> //included for file i/o

#include <string.h> //included for strcmp

#include <array> //included for sizeof()

#include <iomanip> //included for setw

#include <ctype.h> //included for tolower

using namespace std;

/\*---------------------------------------------------------------------------------------

Structure Definitions

---------------------------------------------------------------------------------------\*/

struct arrayelement {

char keyword[10]; //None of the keywords are more than 8 chars

int count; //counts how many times a keyword occurs

};

arrayelement keywords[28]; //stores the keywords. There are 28 unique ones

/\*---------------------------------------------------------------------------------------

Hash-Function Lookup Table

---------------------------------------------------------------------------------------\*/

// a, b, c, d, e, f, g, h, i, j, k, l, m,

int g[28] = { 7, 2, 17, 4, 4, 2, 2, 14, 2, 9, 10, 5, 12,

// n, o, p, q, r, s, t, u, v, w, x, y, z

11, 14, 16, 16, 16, 8, 19, 20, 23, 8, 23, 24, 25 };

/\*---------------------------------------------------------------------------------------

Function Prototypes

---------------------------------------------------------------------------------------\*/

void readfile(void); //reads the keywords.txt file into the keywords table

void outTable(void); //outputs the keyword structure

void searchTable(void); //gets user input. searches the keywords table for it

char inSingle(void); //gets a single character as input and validates it

/\*---------------------------------------------------------------------------------------

Global Data Declarations

---------------------------------------------------------------------------------------\*/

char userIn[sizeof(keywords[0].keyword)] = { 0 }; //used to store user input

int i = 0; //loop control variable

char emp[sizeof(keywords[0].keyword)] = { 0 }; //an empty char array for strcmp

/\*---------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------

THE MAIN FUNCTION

-----------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------\*/

void main()

{

readfile(); //read the file into the keywords structure

outTable(); //output the keywords and their counts

searchTable(); //allow the user to search for a keyword

}

/\*---------------------------------------------------------------------------------------

Function Definitions

---------------------------------------------------------------------------------------\*/

/\*Reads the keywords.txt file into the keywords array and stores

the number of times that an element occurs within the file\*/

void readfile()

{

//------------------------Local Variables-----------------------------------------

char buffer[sizeof(keywords[0].keyword)]; //temp storage for strings read from file

int hash = 0; //a variable used to obtain hash keys

int j = 0; //stores location of last element of buffer

int first = 0; //stores the integer value of buffer's first element

int last = 0; //stores the integer value of buffer's last element

//------------------Check for & open the file-------------------------------------

ifstream fin("keywords.txt"); //open the text file to be used

if (!fin) { //if the file cannot be opened, output an error

cerr << "Input file could not be opened.\n";

exit(1); //after outputing an error, quit the program

}

//---Read the file line by line and store data in the keywords structure----------

while (fin >> buffer) { //while there are strings to input

//the below for loop will find the last element of the string being read

for (int i = 0; buffer[i] != NULL; i++)

j = i;

//obtain the next hash key. we use -65 to change A-Z from 65-91 -> 0-25

first = (int)buffer[0] - 65; //convert buffer[0] to an integer

last = (int)buffer[j] - 65; //convert buffer[j] to an integer

hash = ((g[first] + (2 \* g[last]) + j) % 28);

i = 1; //set i to 1 to loop the next while loop "forever"

while (i) {

//if the key already exists in the hash table, and our input = the stored string

if (strcmp(keywords[hash].keyword, buffer) == 0) {

//increment count. We found a duplicate

keywords[hash].count++;

break; //break the while loop

}

/\*if the key does not exist, it must be new. Put it in the table.

We can tell because if they key exists, hash will be 1 or greater

which evaluates to true. Negating this means if only executes if count = 0\*/

if (keywords[hash].count == 0) {

//copy buffer into the hash table, one element at a time

for (i = 0; i < sizeof(keywords[0].keyword); i++) {

keywords[hash].keyword[i] = buffer[i];

}

//increment count. This element occurred

keywords[hash].count++;

break; //break out of infinite while loop

}

//if we reach this point, they key exists, but the keyword != buffer

if (hash == 27)

hash = 0; // reset hash if it gets to 27

else

hash++; //increment the hash. We found a collision

}

}

fin.close(); //close the keywords file. We don't need it anymore

}

/\*Outputs each element of the keyword table, as

well as the number of times each element occured\*/

void outTable()

{

//--Read the table line by line. Output data & number of times it occurs----------

cout << setw(30) << "Keywords:" << " " << "Count:" << endl;

//loop through each element of keywords

for (int i = 0; i < 28; i++) {

/\*if the element contains a keyword chain, output it and the count.

we compare using userIn because it's empty at this point\*/

if (!(strcmp(keywords[i].keyword, emp) == 0)) {

cout << setw(30) << keywords[i].keyword << " "

<< keywords[i].count << endl;

}

}

setw(0); //reset setw()

}

/\*Takes user input and searches the keyword table for it.

If found, it will output the element and the number of times it occurs

If not found, it will tell the user that their keyword is not in the table\*/

void searchTable()

{

//------------------------Local Variables-----------------------------------------

char failbit; //input regulator

//----------------------Search and Destroy!---------------------------------------

//create a while loop to iterate while the user wants to search for keywords

while (inSingle() == 'y') {

cout << setw(30) << "Please enter a keyword:";

cin.getline(userIn, 8); //get user input

cout << endl;

//iterate through the hash table to see if userIn is there

for (int i = 0; i < 28; i++) {

//if the user has input a null, return not found

if ((strcmp(userIn, emp) == 0)) {

//output not found because we shouldnt be looking for nulls

cout << setw(30) << "FOUND NOTHING. Don't do that again."

<< endl;

i = 28; //set i to 28 so we dont waste time looping

}

//user input is equivalent to the checked element in the table

else if (strcmp(keywords[i].keyword, userIn) == 0) {

//output that it was found, and how many of them there are

cout << setw(30) << "FOUND " << keywords[i].count

<< endl;

i = 28; //set i to 28 so we dont keep checking

}

//if i has reached 27+, we have checked the table and not found a match

else if (i >= 27)

cout << setw(30) << "NOT FOUND" << endl;

}

}

}

/\*Takes a single character (Y/N) as input from the user and validates it.

It is not case sensitive. If the user inputs something else\*/

char inSingle()

{

//------------------------Local Variables-----------------------------------------

char a; //stores user input

char failbit; //input regulator

//--------------------------Gets Input--------------------------------------------

cout << "Would you like to search for a keyword? (Y/N): ";

cin >> a;

//--------------------Validation of User Input------------------------------------

while ((a != 'Y' && a != 'y' &&

a != 'N' && a != 'n'))

{ //this loop validates user input

cin.clear(); //clear the input queue

cin.get(failbit); //reset the failbit flag (eat the terminal null input)

cout << "\aError: Invalid response!\n";

cout << "Please enter your answer in Y/N "

"format: ";

cin >> a; //ask for new input

}

cin.clear(); //clear away any excess input

cin.get(failbit); //eat terminal NULL left by cin

return (tolower(a)); //return in lower case to avoid case sensitivity

}

