/\*

COMPRO2 MACHINE PROBLEM1

Author: CHUA, KYLE MATTHEW

Date of Submission: November 14, 2014

Note: this is the skeleton code for MP1. Fill in the necessary contents.

Do not forget to rename this file with your own lastname.

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

typedef char Str30[31];

#define LIST\_SIZE 154459

struct listTag{

Str30 element;

struct listTag \*pNext;

};

int getHash(Str30 word){

int i;

int hashAdd =0;

for (i =0; word[i]!='\0'; i++)

hashAdd = (hashAdd \* 10 + word[i]) % LIST\_SIZE;

return hashAdd;

}

void Traverse(struct listTag \*pCurrent){

if(pCurrent != NULL){

while(pCurrent){

printf("%s\n", pCurrent->element);

pCurrent = pCurrent->pNext;

}

}

else

printf("NOMATCH\n");

}

struct listTag \*Create\_Sort\_Node(struct listTag \*pList, Str30 word){

struct listTag \*pTemp;

struct listTag \*pTrail;

struct listTag \*pCurrent;

pTrail = NULL;

pCurrent = pList;

pTemp = malloc(sizeof(struct listTag));

strcpy(pTemp->element, word);

pTemp->pNext = NULL;

if(pList == NULL)

pList = pTemp;

else{

while(pCurrent != NULL){

if(strcmp(pTemp->element, pCurrent->element) == 1){

pTrail = pCurrent;

pCurrent = pCurrent->pNext;

}

else

break;

}

if(pTrail != NULL){

pTrail->pNext = pTemp;

pTemp->pNext = pCurrent;

}

else{

pTemp->pNext = pCurrent;

pList = pTemp;

}

}

return pList;

}

/\*

This function should initialize the contents of your data structure,

i.e., the storage representation of the words in the universe of words.

Specify the data type of the universe parameter.

\*/

void Initialize( struct listTag \*\*universe)

{

int i;

Str30 word;

int hashAdd;

struct listTag \*pTemp;

\*(universe) = malloc(sizeof(struct listTag) \* LIST\_SIZE);

for(i=0; i<LIST\_SIZE; i++){

strcpy(((\*universe)+i)->element, "-1");

((\*universe)+i)->pNext = NULL;

}

while(scanf("%s", word) == 1){

hashAdd = getHash(word);

if(!(strcmp(((\*universe)+hashAdd)->element, "-1"))){

strcpy(((\*universe)+hashAdd)->element, word);

}

else if(((\*universe)+hashAdd)->pNext != NULL){

pTemp = ((\*universe)+hashAdd)->pNext;

while(pTemp->pNext)

pTemp = pTemp->pNext;

pTemp->pNext = malloc(sizeof(struct listTag));

pTemp->pNext->pNext = NULL;

strcpy(pTemp->pNext->element, word);

}

else{

((\*universe)+hashAdd)->pNext = malloc(sizeof(struct listTag));

((\*universe)+hashAdd)->pNext->pNext = NULL;

strcpy(((\*universe)+hashAdd)->pNext->element,word);

}

}

}

/\*

This function will search the word in the universe of words.

Specify the data type of universe parameter.

\*/

void Search( struct listTag \*universe, Str30 word)

{

int i, j, k;

int quesNum=0, astNum=0, astCtr;

struct listTag \*pTemp;

struct listTag \*pList;

Str30 key = "\0";

pList = NULL;

pTemp = NULL;

for(i=0; word[i] != '\0'; i++){

if(word[i]=='?'){

key[i] = word[i];

quesNum++;

}

else if(word[i]=='\*'){

key[i] = word[i];

astNum++;

}

else

key[i] = word[i];

}

strlwr(key);

if(quesNum == 0 && astNum == 0){

if(!(strcmp((universe+getHash(key))->element, key)))

pList = Create\_Sort\_Node(pList, key);

else if((universe+getHash(key))->pNext != NULL){

pTemp = (universe+getHash(key))->pNext;

while(pTemp != NULL){

if(!(strcmp(pTemp->element, key))){

pList = Create\_Sort\_Node(pList, key);

break;

}

else

pTemp = pTemp->pNext;

}

}

}

else if(quesNum > 0){

for(i=0; i<LIST\_SIZE; i++){

pTemp = (universe+i);

if(strcmp(pTemp->element, "-1")){

while(pTemp != NULL){

if(strlen(pTemp->element) == strlen(key)){

for(j=0; key[j] != '\0'; j++){

if(key[j] != '?'){

if(key[j] != pTemp->element[j])

break;

}

}

if(j == strlen(key)){

pList = Create\_Sort\_Node(pList, pTemp->element);

}

}

pTemp = pTemp->pNext;

}

}

}

}

else if(astNum > 0){

for(i=0; i<LIST\_SIZE; i++){

pTemp = (universe+i);

if(strcmp(pTemp->element, "-1")){

while(pTemp != NULL){

k=0;

j=0;

astCtr = astNum;

while(key[j] != '\0'){

if(astCtr){

if(key[j] != '\*'){

if(key[j] != pTemp->element[k])

break;

else{

k++;

j++;

}

}

else{

j++;

astCtr--;

if(key[j] == '\0')

k = strlen(pTemp->element);

else if(key[j+1] != '\0'){

if(astCtr){

while(pTemp->element[k] != key[j] && pTemp->element[k] != '\0')

k++;

if((strlen(pTemp->element)-k < strlen(key)-j-astCtr))

break;

else{

j++;

k++;

}

}

}

}

}

else{

if(key[j] != '\0'){

if(key[j+1] == '\0'){

k = strlen(pTemp->element)-1;

if(key[j] == pTemp->element[k]){

j++;

k++;

}

else

break;

}

else{

k = strlen(pTemp->element) - strlen(key) + j;

if(key[j] == pTemp->element[k]){

j++;

k++;

}

else

break;

}

}

}

}

if(j == strlen(key) && k == strlen(pTemp->element)){

pList = Create\_Sort\_Node(pList, pTemp->element);

}

pTemp = pTemp->pNext;

}

}

}

}

Traverse(pList);

free(pList);

}

int main()

{

time\_t t0, t1; /\* time\_t is defined on <time.h> \*/

clock\_t c0, c1; /\* clock\_t is defined on <time.h> \*/

struct listTag \*universe; // provide the data type of universe

// the universe will contain the words from enable2k.txt

// declare your other own local variables here

/\*

DO NOT remove the next two assignment operations

which measure start time t0, c0

\*/

t0 = time(NULL); // time() measures wallclock time

c0 = clock(); // clock() measures CPU time

/\* initialize the universe of words \*/

Initialize(&universe);

/\* test the Search() function \*/

Search(universe, "computer");

Search(universe, "COmPuTeR"); // should produce same result as above

Search(universe, "a?t");

Search(universe, "a\*t");

Search(universe, "art\*");

Search(universe, "\*art");

Search(universe, "wxyz"); // this will not produce a match

/\*

DO NOT remove the remaining statements below.

Measure end time, and report elapsed time.

Note: the elapsed time will tell you the performance of your

design/implementation. Strive for a computationally

efficient solution.

\*/

// measure end time t1, c1

t1 = time(NULL);

c1 = clock();

printf ("\tbegin (wall): %ld\n", (long) t0);

printf ("\tbegin (CPU): %d\n", (int) c0);

printf ("\tend (wall): %ld\n", (long) t1);

printf ("\tend (CPU); %d\n", (int) c1);

printf ("\telapsed wall clock time: %ld\n", (long) (t1 - t0));

printf ("\telapsed CPU time: %f\n", (float) (c1 - c0)/CLOCKS\_PER\_SEC);

free(universe);

return 0;

}