#include <iostream>

#include <fstream>

using namespace std;

#define MAXLINE 100

int getLenChars(char str[])

{ // returns number of characters in a string

int i = 0;

int len = 0;

while (str[i] != '\0')

{

len++;

i++;

}

return len;

}

void toUpper(char input[])

{

int length = getLenChars(input);

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

{

if ((input[i] >= 'a') && (input[i] <= 'z'))

{// if the character is lowercase

input[i] -= 32; // maker uppercase

}

}

}

bool isLetter(char c)

{

return((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'));

}

void countChars(char inputLine[], int len, int counts[], int &totalLetters)

{

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

{

if (isLetter(inputLine[i]) )

{

totalLetters++;

counts[int(inputLine[i]) - 65]++;

}

}

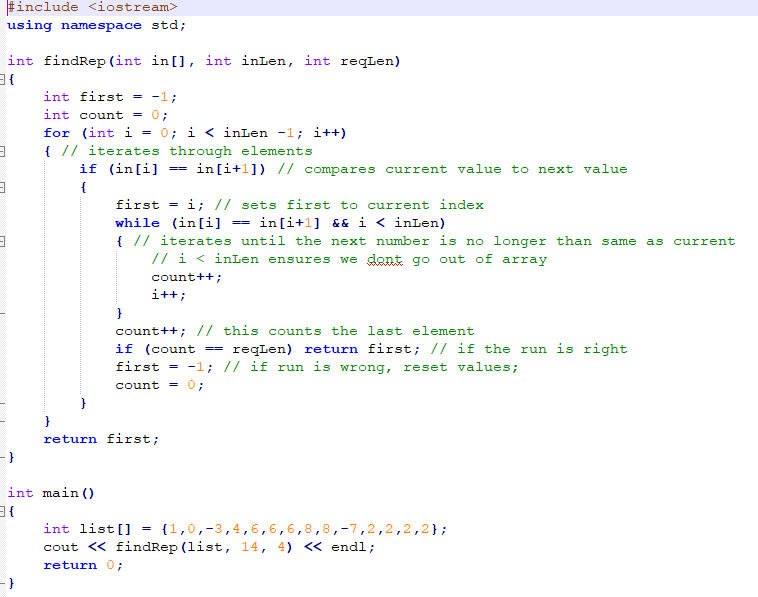
}

void printCounts(int counts[])

{

for (int i = 0; i < 26; i++)

{

 cout << char(i+65) << " : " << counts[i] << endl;

}

}

void findFrequencies(int counts[], int totalLetters, float frequencies[])

{

for (int i = 0; i < 26; i++)

{

if (counts[i] > 0)

{

frequencies[i] = (0.0+counts[i])/(0.0+totalLetters);

}

}

}

void printFrequencies(float frequencies[])

{

for (int i = 0; i < 26; i++)

{

cout << char(i+65) << " : " << frequencies[i] << endl;

}

}

void printAllData(int counts[],float frequencies[],int totalLetters)

{

cout << "Total letter count: " << totalLetters << endl;

cout << endl;

for (int i = 0; i < 26; i++)

{

cout << char(i+65) << ": count: " << counts[i] << " frequencies: " << frequencies[i] << endl;

}

}

int main()

{

ifstream file;

file.open("input.txt", ios\_base::in);

int counts[26] = {0}; // counts[0] is count of A; [1] for Z

int totalLetters = 0;

float frequencies[26] = {0.0};

char inputLine[MAXLINE];

while (file.getline(inputLine, MAXLINE))

{

toUpper(inputLine);

int len = getLenChars(inputLine);

countChars(inputLine, len, counts, totalLetters);

}

findFrequencies(counts, totalLetters, frequencies);

printAllData(counts, frequencies, totalLetters);

file.close();

return 0;

}

#include <iostream>

cout << "--------------------------" << endl;

cout << " Ordering System " << endl;

cout << "--------------------------" << endl;

cout << "Options: " << endl;

cout << "\t'N' - Create new order" << endl;

cout << "\t'S' - Show order queue" << endl;

cout << "\t'F' - Complete first order" << endl;

cout << "\t'P' - Show phone orders" << endl;

cout << "\t'D' - Show delivery queue" << endl;

cout << "\t'U' - Complete first delivery" << endl;

cout << "\t'C' - Clear order queue" << endl;

cout << "\t'Q' - Quit" << endl;

bool charInArray(char c, const char s[])

{ // assumes string is null terminated

int i = 0;

while (s[i] != '\0')

{

if (c == s[i++]) return true;

}

return false;

}

char prompt(const char msg[], const char validChoices[])

{

char input;

if (validChoices[0] == '\0') return '\0';

cout << msg << " ";

while (true)

{

cout << "[";

int i = 0;

while (validChoices[i] != '\0')

{

cout << upcase(validChoices[i]);

if (validChoices[i+1] != '\0') cout << "/";

i++;

}

cout << "] ";

cin >> input;

if (charInArray(upcase(input), validChoices)) return upcase(input);

}

}

#include <iostream>

#include <fstream>

using namespace std;

char endPunctuation[] = ".?!";

bool validChar(char c)

{

if ( (c >= 'A') && (c <= 'Z')) return true;

if ( (c >= 'a') && (c <= 'z')) return true;

if (c == '&' || c == '-' || c == '\'') return true;

return false;

}

void revSingleWord(char s[], int f, int l)

{

for (int i = 0; i < (l-f+1)/2; i++)

{

char hold = s[f+i];

s[f+i] = s[l-i];

s[l-i] = hold;

}

}

void revSingleWordRecursive(char s[], int f, int l)

{

if (f >= l) return;

char hold = s[f];

s[f] = s[l];

s[l] = hold;

revSingleWordRecursive(s, f+1, l-1);

}

void revWords(char s[])

{

int i = 0;

while (s[i] != '\0')

{

int j = i;

while (!validChar(s[j]) && s[j] != '\0')

{

j++;

}

// j is now the beginning of a word

i = j;

while (validChar(s[j]))

{

j++;

}

revSingleWord(s, i, j-1);

i = j;

}

}

int main()

{

char s[] = "this is a string of char's and now it should! be reversed";

revWords(s);

cout << s << endl;

return 0;

}

using namespace std;

#define MAXSUIT 9

struct card\_t{

char suit[MAXSUIT];

int value; // 1 is ace, 11 jack, 12 queen, 13 king

};

bool isFace(card\_t card)

{

return (card.value > 10 || card.value ==1);

}

void sort(int list[], int n)

{

int tmp;

for (int i = 0; i < n-1; i++)

**Prompting**

{

if (list[i] > list[i+1])

{

tmp = list[i];

list[i+1] = list[i];

list[i] = tmp;

}

}

}

bool isSequence(card\_t cards[], int n)

{

int values[n];

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

{

values[i] = cards[i].value;

}

sort(values, n);

for (int i = 0; i < n-1; i++)

{

if (values[i] == 1) // if king

{

if ( values[i+1] != 13 ) return false;

}

else if (values[i+1] != values[i]+1) return false;

}

return true;

}

int getLenChars(char str[])

{ // returns number of characters in a string

int i = 0;

int len = 0;

while (str[i] != '\0')

{

len++;

i++;

}

return len;

}

bool sameString(char first[], char second[])

{ // return true iff two strings are identical, character for character

int firstLen = getLenChars(first);

int secondLen = getLenChars(second);

When it comes to string comparisons,

a < z.

A < a.

if (firstLen != secondLen) return false;

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

{

if (first[i] != second[i]) return false;

#include <iostream>

using namespace std;

void mystery(int l[], int n)

{

int tmp = l[0];

for (int i = 0; i < n -1; i++)

{

l[i] = l[i+1];

}

l[n-1] = tmp;

}

int main()

{

int l[] = {100, 200, 300, 400, 500};

int n = 5;

mystery(l, n);

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

{

cout << l[i] << endl;

}

return 0;

}

}

return true;

}

bool sameSuit(card\_t cards[], int n)

{

char testSuit[MAXSUIT];

int i = 0;

int lenTestSuit = getLenChars(cards[0].suit);

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

{

testSuit[i] = cards[0].suit[i];

}

testSuit[lenTestSuit] = '\0';

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

{

if (!sameString(cards[i].suit, testSuit))

return false;

}

return true;

}

bool isFlush(card\_t cards[], int n)

{

return (sameSuit(cards, n) && !isSequence(cards, n));

}

int main()

{

card\_t cards[3] = { {"Club", 1}, {"Club", 13}, {"Heart", 5}};

cout << isFlush(cards, 3) << endl;

return 0;

}

**STACKS**

#include "stack.h" // We implement this functionality

#include <string> /\* Using the C++ standard string type \*/

#include <assert.h>

using namespace std;

struct stack\_item\_t {

string data;

stack\_item\_t \*next;

};

struct stack\_t {

stack\_item\_t \*head;

};

stack\_t \*create\_stack()

{

stack\_t \*stk = new stack\_t; // Make a new data structure to store the stack

assert(stk != NULL); // If this failed, we're in trouble

stk->head = NULL;

return stk;

}

void destroy\_stack(stack\_t \*stk)

{

assert(stk != NULL);

stack\_item\_t \*p = NULL;

while (!is\_empty\_stack(stk)) // Remove and free the items

pop\_off\_stack(stk);

delete stk; // Delete the stack itself

}

void push\_on\_stack(stack\_t \*stk, string s)

{

assert(stk != NULL);

stack\_item\_t \*p = new stack\_item\_t; // Create a stack item

assert(p != NULL); // If this failed, we're in trouble

p->data = s; // String will copy all the necessary bits

p->next = stk->head;

stk->head = p;

}

bool is\_empty\_stack(stack\_t \*stk)

{

assert(stk != NULL);

return (stk->head == NULL);

}

string pop\_off\_stack(stack\_t \*stk)

{

assert(stk != NULL);

assert(stk->head != NULL); // Stack must not be empty

stack\_item\_t \*p = stk->head;

stk->head = stk->head->next;

string ret = p->data;

delete p;

return ret;

}

string peek\_top\_of\_stack(stack\_t \*stk)

{

assert(stk != NULL);

assert(stk->head != NULL); // Stack must not be empty

return stk->head->data;

}

bool string\_eq(char first[], char second[])

{ // return true iff two strings are identical, character for character

int firstLen = getLenChars(first);

int secondLen = getLenChars(second);

if (firstLen != secondLen) return false;

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

{

if (first[i] != second[i]) return false;

}

return true;

}

void toUpper(char input[])

{

int length = getLenChars(input);

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

{

if ((input[i] >= 'a') && (input[i] <= 'z'))

{// if the character is lowercase

input[i] -= 32; // maker uppercase

}

}

}

bool string\_eq\_nocase(char first[], char second[])

{ // a case insensitive check for string equality

toUpper(first);

toUpper(second);

if ( string\_eq(first, second) ) return true;

return false;

}

void strip\_dup\_spaces(char str[])

{ // removes all consecutive spaces from str

int len = usedLen(str);

for (int i = 0; i < len-1; i++)

{

while((str[i] == ' ') && (str[i+1] == ' ') )

{

for (int j = i; j < len-1; j++)

{

str[j] = str[j+1];

}

}

}

}

bool contains\_sub\_str(char haystack[], char needle[])

{ // return true iff needle appears within haystack

int len2 = getLenChars(needle);

int len1 = usedLen(haystack);

char test[len2];

test[len2] = '\0';

for (int i = 0; i < len1-len2; i++)

{

if (haystack[i] == needle[0])

{

for(int j = 0; j < len2; j++)

{

test[j] = haystack[i+j];

}

if (string\_eq(test, needle)) return true;

}

}

return false;

}

int index\_sub\_str(char haystack[], char needle[])

{ // return true iff needle appears within haystack

int len2 = getLenChars(needle);

int len1 = usedLen(haystack);

char test[len2];

test[len2] = '\0';

for (int i = 0; i < len1-len2; i++)

{

if (haystack[i] == needle[0])

{

for(int j = 0; j < len2; j++)

{

test[j] = haystack[i+j];

}

if (string\_eq(test, needle)) return i;

}

}

return -1;

}

bool del\_first\_occur(char input[], char cut[])

{ // if cut appears in input, remove it and return true;

// otherwise leave input as is and return false

if (!contains\_sub\_str(input, cut)) return false;

int lenIn = usedLen(input);

int lenCut = getLenChars(cut);

int indexCut = index\_sub\_str(input, cut);

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

{

for (int j = indexCut; j < lenIn; j++)

{

input[j] = input[j+1];

}

}

return true;

}