МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №1**

**з дисципліни: «Об’єктно-орієнтоване програмування»**

Виконав:

Студент групи КВ-31

Волонтир О.О.

Перевірив:

Київ 2014

**Текст програми**

**Частина перша**

* Файл *project1.c*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: project1.c

\*synopsis: This file is used to test the functions located in file "str\_work.c"

\*and declared in file "string.h"

\*author: volontyr alexandr

\*written: 03/09/2014

\*last modified: 06/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include "string.h"

int main(){

char \*str1 = NULL;

char \*str2 = NULL;

char \*str3 = NULL;

char \*str4 = NULL;

char \*str5 = NULL;

int \*arr\_size;

double \*array = NULL;

arr\_size = NULL;

printf("Enter the first line\n");

str1 = put\_str();

printf("Enter the second line\n");

str2 = put\_str();

printf("result = %d\n", substr(str1, str2));

printf("max general length = %d\n", subseq(str1, str2));

printf("Enter the word you want to check for palindromicity\n");

str3 = put\_str();

printf("polyndrom or not ('1' or '0'): %d\n", ispal (str3));

printf("Enter any word\n");

str4 = put\_str();

printf("polyndrom maden of the word = %s\n", makepal(&str4));

printf("Enter the line of some numbers separated with a semicolon\n");

str5 = put\_str();

printf("Array of these numbers: ");

array = txt2double(str5, &arr\_size);

if (\*arr\_size != 0) {

for (int i = 0, len = \*arr\_size; i < len; i++) {

printf("%f\t", array[i]);

}

printf("\n");

}

else {

printf("convertation error\n");

}

return 0;

}

* Файл *str\_work.c*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: str\_work.c

\*synopsis: The string functions operate with strings:

\*check for containing any substring or check for palyndromicity;

\*they also change the strings by adding some characters.

\*These functions are declared in the include file "string.h".

\*related files: none

\*author: volontyr alexandr

\*written: 03/09/2014

\*last modified: 06/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <malloc.h>

#include <string.h>

#include <stdlib.h>

#include "string.h"

typedef enum { false, true } bool;

/\*

The put\_str function gets a string from a keyboard and returns it.

\*/

char\* put\_str(){

int i;

char \*string = (char\*)malloc(sizeof(char)\* 255);

string = (char\*)realloc(string, sizeof(char)\* 2);

string[0] = getchar();

i = 1;

while ((string[i] = getchar()) != '\n') {

++i;

string = (char\*)realloc(string, (i + 1)\*sizeof(char));

}

string[i] = '\0';

return string;

}

/\*

The substr function searches string2 in string1 and returns a number of string2 position

in string1 or returns -1 if string1 doesn't contain string2.

\*/

int substr(const char \*string1, const char \*string2) {

int i, j, result = -1;

bool flag;

for (i = 0; string1[i] != '\0'; i++) {

if (string1[i] == string2[0]) {

flag = true;

for (j = 1; string2[j] != '\0'; j++) {

if (string1[i + j] != string2[j]) {

flag = false;

break;

}

}

if (flag == true) {

result = i;

break;

}

}

}

return result;

}

/\*

The subseq function returns maximal length of common subsequence among such strings as string1 and string2.

\*/

int subseq(const char \*string1, const char \*string2) {

int k, max\_len = 0;

for (int i = 0; string1[i] != '\0'; i++) {

for (int j = 0; string2[j] != '\0'; j++) {

if (string1[i] == string2[j]) {

k = 1;

while ((string1[k + i] == string2[k + j]) && (string1[k + i] != '\0' || string2[k + j] != '\0')) {

k++;

}

if (k > max\_len) {

max\_len = k;

}

}

}

}

return max\_len;

}

/\*

The ispal function checks if string is palyndrom. It retruns 1 if it's true or 0 if it's false.

\*/

char ispal(const char \*string) {

char result = 1;

int length = strlen(string) - 1;

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

if (string[i] != string[length - i]) {

result = 0;

break;

}

}

return result;

}

/\*

The makepal function makes a palyndrom of string adding to it minimal number of characters.

The function returns a pointer to this string.

\*/

char\* makepal(char \*\*string) {

int i = strlen(\*string) - 1, length = strlen(\*string) - 1, pos = -1;

char \*str = (char\*)malloc(sizeof(char)\* i);

if (ispal(\*string) == 1) {

return \*string;

}

str[0] = (\*string)[length];

while (i > 0) {

i--;

str[length - i] = (\*string)[i];

str[length - i + 1] = '\0';

if (ispal(str) == 1) {

pos = i - 1;

}

}

if (pos == -1) {

pos = length - 1;

}

\*string = (char\*)realloc(\*string, sizeof(char)\*(length + pos + 1));

for (i = pos; i >= 0; i--) {

(\*string)[length + pos - i + 1] = (\*string)[i];

}

(\*string)[length + pos + 2] = '\0';

return \*string;

}

/\*

The txt2double function converts separated numbers with semicolons located in string and adds these

numbers to an array. Parameter size takes a value of the array length if convertation is successful or 0 if it isn't.

The function returns a pointer to the array of numbers.

\*/

double\* txt2double(const char \*string, int \*\*size) {

int num\_elem = 0;

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

if (string[i] == ';') {

num\_elem++;

}

}

num\_elem++;

double \*arr = (double\*)malloc((num\_elem)\*sizeof(double));

char \*str = (char\*)malloc(sizeof(char)\*(strlen(string)));

int k = 0, j = 0;

for (int unsigned i = 0; i <= strlen(string); i++) {

if (string[i] != ';' && string[i] != '\0') {

str[k] = string[i];

k++;

}

else {

str[k] = '\0';

if ((atof(str) != 0.0) || (atof(str) == 0.0 && \*str == '0')) {

arr[j] = atof(str);

free(str);

str = (char\*)malloc(sizeof(char)\*(strlen(string)));

}

else {

\*size = 0;

}

j++;

k = 0;

}

}

free(str);

if (size != NULL) {

\*size = &num\_elem;

}

return arr;

}

* Header - file *string.h*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: string.h

\*synopsis: declarations for string functions

\*author: volontyr alexandr

\*written: 03/09/2014

\*last modified: 06/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_STRING\_H

#define \_STRING\_H

/\* function prototypes \*/

char\* put\_str();

int substr(const char \*string1, const char \*string2);

int subseq(const char \*string1, const char \*string2);

char ispal(const char \*string);

char\* makepal(char \*\*string);

double\* txt2double(const char \*string, int \*\*size);

#endif

**Частина друга**

* Файл *argztest.c*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: argztest.c

\*synopsis: This file is used to test the functions located in file "argz.c"

\*and declared in file "argz.h"

\*author: volontyr alexandr

\*written: 06/09/2014

\*last modified: 11/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "argz.h"

int main() {

char \*string = NULL;

char \*entry = NULL;

char \*before = NULL;

char \*entry2 = NULL;

char \*entry3 = NULL;

char \*str = NULL;

char \*with = NULL;

char \*add\_str = NULL;

char \*argz;

int argz\_len = 0;

int symbol;

argz = NULL;

printf("Enter some string to create vector argz:\n");

string = put\_str();

printf("Enter a symbol to separate the strings\n");

symbol = getchar();

printf("The result is %d\n", argz\_create\_sep(string, symbol, &argz, &argz\_len));

argz\_print(argz, argz\_len);

printf("number of elemnts in vector argz = %d\n\n", argz\_count(argz, argz\_len));

printf("Enter any string to add it to the argz:\n");

add\_str = put\_str();

printf("The result of adding is %d\n", argz\_add(&argz, &argz\_len, add\_str));

argz\_print(argz, argz\_len);

printf("Enter the string to delete it from argz\n");

entry = put\_str();

argz\_delete(&argz, &argz\_len, entry);

argz\_print(argz, argz\_len);

printf("Enter any string to insert it into argz\nand the string to insert it before\n");

entry2 = put\_str();

before = put\_str();

printf("the result of inserting is %d\n", argz\_insert(&argz, &argz\_len, before, entry2));

argz\_print(argz, argz\_len);

system("pause");

system("cls");

printf("Enter any element of vector argz\n");

entry3 = put\_str();

printf("The next element is \n%s\n", argz\_next(argz, argz\_len, entry3));

printf("Enter the string you want to replace and the string to replace witn\n");

str = put\_str();

with = put\_str();

printf("The result of replacing is %d\n", argz\_replace(&argz, &argz\_len, str, with));

argz\_print(argz, argz\_len);

}

* Файл*argz.c*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: argz.c

\*synopsis: The argz functions create argz vectors,

\*they also operate with the vector argz in such way as adding

\*some element to the end of argz vector, deleting or inserting some element in argz and so on.

\*Argz functions that do memory allocation have a return type of error\_t, and

\*return 0 for success, and ENOMEM if an allocation error occurs.

\*These functions are declared in the include file "argz.h".

\*related files: none

\*author: volontyr alexandr

\*written: 06/09/2014

\*last modified: 11/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

#include "argz.h"

#pragma warning(disable : 4018)

typedef enum {false, true} bool;

/\*

The describtion of this function is in file "str\_wrok.c"

\*/

char\* put\_str(){

int i;

char \*string = (char\*)malloc(sizeof(char)\*255);

string = (char\*)realloc(string, sizeof(char)\*2);

string[0] = getchar();

i = 1;

while ((string[i] = getchar()) != '\n') {

++i;

string = (char\*)realloc(string, (i + 1)\*sizeof(char));

}

string[i] = '\0';

return string;

}

/\*

The describtion of this function is in file "str\_wrok.c"

\*/

int substr(const char \*string1, const char \*string2) {

int i, j, result = -1;

bool flag;

for (i = 0; string1[i] != '\0'; i++) {

if (string1[i] == string2[0]) {

flag = true;

for (j = 1; string2[j] != '\0'; j++) {

if (string1[i + j] != string2[j]) {

flag = false;

break;

}

}

if (flag == true) {

result = i;

break;

}

}

}

return result;

}

/\*

The argz\_create\_sep function converts the null-terminated string string into an

argz vector (returned in argz and argz len) by splitting it into elements at every

occurrence of the character sep.

\*/

error\_t argz\_create\_sep(const char \*string, int sep, char \*\*argz, size\_t \*argz\_len) {

error\_t err;

bool flag = false;

char \*str = (char\*)malloc(sizeof(char)\*(strlen(string)));

int step = 0, last\_pos = 0, num\_sep = 0;

int unsigned i = 0;

for (i; i <= strlen(string); i++) {

str[i + step] = string[i];

if (string[i] == '=' && string[i + 1] != sep && string[i + 1] != '\0') {

flag = true;

}

if (string[i] == sep || string[i] == '\0') {

if (flag == false) {

step = last\_pos - i;

}

else {

last\_pos = i;

flag = false;

str[i + step] = '\0';

num\_sep++;

}

}

}

if (num\_sep == 0) { step--; }

str = (char\*)realloc(str, sizeof(char)\*(i + step));

str[i + step - 1] = '\0';

\*argz\_len = i + step - num\_sep + 1;

if (str[0] != '\0') {

\*argz = str;

}

if (\*argz != NULL) {

err = OK;

}

else { err = ENOMEM ; }

return err;

}

/\*

Returns the number of elements in the argz vector.

\*/

size\_t argz\_count(const char \*argz, size\_t arg\_len) {

int num\_elem = 0;

for (int unsigned i = 0; i < arg\_len; i++) {

if (argz[i] == '\0') {

++num\_elem;

if (argz[i + 1] != '\0') { arg\_len++; }

}

}

return num\_elem;

}

/\*

The argz\_add function adds the string str to the end of the argz vector \*argz,

and updates \*argz and \*argz\_len accordingly.

\*/

error\_t argz\_add(char \*\*argz, size\_t \*argz\_len, const char \*str) {

int num\_el = argz\_count(\*argz, \*argz\_len) + \*argz\_len - 1;

int pos = substr(str, "="), i = 1;

if ((pos > 0) && str[pos + 1] != '\0' && str[pos - 1] != '\n') {

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

\*argz = (char\*)realloc(\*argz, sizeof(char)\*(i + num\_el + 1));

(\*argz)[(i - 1) + num\_el] = str[i];

(\*argz\_len)++;

}

\*argz = (char\*)realloc(\*argz, sizeof(char)\*(i + num\_el));

(\*argz)[i + num\_el - 1] = '\0';

return OK;

}

else { return ENOMEM; }

}

/\*

The check\_in\_argz function checks if string str is an element of vector argz or not.

It returns a number of str position in vector argz if str is an element of the vector or

-1 if it isn't.

\*/

int check\_in\_argz(const char \*argz, size\_t argz\_len, const char \*str) {

bool flag = false;

int num\_el = argz\_count(argz, argz\_len);

int pos\_str = 0, length = strlen(str);

for (int i = 0; i <= argz\_len + num\_el && flag == false; i++) {

if (argz[i] == str[0] && (i == 0 || argz[i - 1] == '\0')) {

pos\_str = i;

for (int j = 1; j <= length; j++) {

if (argz[j + pos\_str] != str[j]) { break; }

else {

if (str[j] == '\0') {

flag = true;

break;

}

}

}

}

}

if (flag == true) { return pos\_str; }

else { return -1; }

}

/\*

If entry points to the beginning of one of the elements in the argz vector \*argz,

the argz\_delete function will remove this entry and reallocate \*argz,

modifying \*argz and \*argz\_len accordingly.

\*/

void argz\_delete(char \*\*argz, size\_t \*argz\_len, const char \*entry) {

int num\_el = argz\_count(\*argz, \*argz\_len);

int i = 0, pos\_entry = 0, length = strlen(entry);

pos\_entry = check\_in\_argz(\*argz, \*argz\_len, entry);

if (pos\_entry != -1) {

for (i = pos\_entry + length; i <= \*argz\_len + num\_el; i++) {

(\*argz)[i - length - 1] = (\*argz)[i];

}

\*argz\_len -= length;

}

}

/\*

The argz\_insert function inserts the string entry into the argz vector \*argz at a point just before

the existing element pointed to by before,

reallocating \*argz and updating \*argz and \*argz\_len.

\*/

error\_t argz\_insert(char \*\*argz, size\_t \*argz\_len, char \*before, const char \*entry) {

int num\_el = argz\_count(\*argz, \*argz\_len);

int i = 0, pos\_before = 0, length = strlen(before);

int pos = substr(entry, "=");

pos\_before = check\_in\_argz(\*argz, \*argz\_len, before);

if (pos > 0 && entry[pos + 1] != '\0' && pos\_before != -1) {

\*argz = (char\*)realloc(\*argz, sizeof(char)\*(\*argz\_len + num\_el + length + 1));

for (i = \*argz\_len + num\_el; i >= pos\_before; i--) {

(\*argz)[i + strlen(entry) + 1] = (\*argz)[i];

}

for (i = 0; i <= strlen(entry); i++) {

(\*argz)[i + pos\_before] = entry[i];

}

\*argz\_len += strlen(entry);

return OK;

}

else { return ENOMEM; }

}

/\*

The argz\_next function provides a convenient way of iterating over the elements in the argz vector argz.

It returns a pointer to the next element in argz after the element entry,

or 0 if there are no elements following entry. If entry is 0, the first element of argz is returned.

\*/

char\* argz\_next(char \*argz, size\_t argz\_len, const char \*entry) {

char \*string = (char\*)malloc(sizeof(char)\*argz\_len);

int num\_el = argz\_count(argz, argz\_len);

int i = 0, pos\_entry = 0, length = strlen(entry);

pos\_entry = check\_in\_argz(argz, argz\_len, entry);

if (entry[0] == '\n') { length -= 1; }

if (pos\_entry != -1 || entry[0] == '\n') {

for (i = pos\_entry + length + 1; argz[i] != '\0'; i++) {

string = (char\*)realloc(string, sizeof(char)\*(i - pos\_entry - length));

string[i - pos\_entry - length - 1] = argz[i];

}

string[i - pos\_entry - length - 1] = '\0';

if (string[0] == '\0') { string = NULL; }

}

else {

string = NULL;

}

if (entry[length - 1] == argz[argz\_len + num\_el - 3]) { string = NULL; }

return string;

}

/\*

The argz\_replace function replaces the string str in argz with string with, reallocating argz as

necessary.

\*/

error\_t argz\_replace(char \*\*argz, size\_t \*argz\_len, const char \*str, const char \*with) {

error\_t err;

int pos\_str = 0;

char \*string = (char\*)malloc(sizeof(char)\*(\*argz\_len));

pos\_str = check\_in\_argz(\*argz, \*argz\_len, str);

if (pos\_str != -1) {

string = argz\_next(\*argz, \*argz\_len, str);

if (string == NULL) {

string = (char\*)realloc(string, sizeof(char)\*(strlen(str)));

string = str;

}

err = argz\_insert(&(\*argz), &(\*argz\_len), string, with);

if (err == OK) {

argz\_delete(&(\*argz), &(\*argz\_len), str);

}

}

else { err = ENOMEM; }

return err;

}

/\* The argz\_print function prints argz vector \*/

void argz\_print(const char \*argz, size\_t argz\_len) {

int num\_el = argz\_count(argz, argz\_len);

printf("The vector argz:\t");

for (int i = 0; i <= argz\_len + num\_el - 2; i++) {

printf("%c", argz[i]);

}

printf("\nThe length of argz = %d\n", argz\_len);

}

* Файл*argz.h*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: argz.h

\*synopsis: declarations for argz functions and types

\*author: volontyr alexandr

\*written: 06/09/2014

\*last modified: 11/09/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_ARGZ\_H

#define \_ARGZ\_H

typedef enum { OK, ENOMEM } error\_t;

/\* function prototypes \*/

char\* put\_str();

error\_t argz\_create\_sep(const char \*, int, char \*\*, size\_t \*);

size\_t argz\_count(const char \*, size\_t);

error\_t argz\_add(char \*\*, size\_t \*, const char \*);

void argz\_delete(char \*\*, size\_t \*,const char \*);

error\_t argz\_insert(char \*\*, size\_t \*, char \*, const char \*);

char \* argz\_next(char \*, size\_t, const char \*);

error\_t argz\_replace(char \*\*, size\_t \*, const char \*, const char \*);

void argz\_print(const char \*, size\_t);

#endif