

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №1**

з дисципліни

**«Об'єктно Орієнтоване Програмування»**

Тема: **«**Вказівники, рядки, масиви***»***

Виконав: студент ІI курсу

ФПМ групи КВ-61

Бідяк М. А.

Перевірив:

Київ-2017

**Завдання**

* 1. Написати функцію error\_t argz\_create\_sep (const char \*string, int sep, char \*\*argz, size\_t \*argz\_len). Функція перетворює null-terminated рядок string в argz вектор, розбиваючи рядок на елементи щораз, коли зустрічається символ sep. Вектор argz є null-terminated рядок, у якій елементи розділені символом ’\0’.

2. Написати функцію size\_t argz\_count (const char \*argz, size\_t arg\_len), що повертає число елементів argz. Для попереднього приклада результатом буде 4.

3. Написати функцію error\_t argz\_add (char \*\*argz, size\_t \*argz\_len, const char \*str), що додає рядок str у кінець argz, модифікуючи належним чином argz й argz\_len.

4. Написати функцію void argz\_delete (char \*\*argz, size\_t \*argz\_len, char \*entry), що видаляє елемент entry з argz, модифікуючи належним чином argz й argz\_len.

5. Написати функцію error\_t argz\_insert (char \*\*argz, size\_t \*argz\_len, char \*before, const char \*entry), що вставляє рядок entry перед елементом before, модифікуючи належним чином argz й argz\_len.

6. Написати функцію char \* argz\_next (char \*argz, size\_t argz\_len, const char \*entry), що забезпечує можливість переміщення по argz. Вона повертає покажчик на елемент argz, що слідує за елементом entry або 0, якщо таких немає. Якщо entry дорівнює 0, повертається покажчик на перший елемент argz. Таким чином, забезпечується одна із двох можливостей доступу до елементів argz.

7. Написати функцію error\_t argz\_replace(char \*\*argz, size\_t \*argz\_len, const char \*str, const char \*with), що заміщає рядок str елементом with, модифікуючи належним чином argz й argz\_len.

8. Написати функцію void argz\_print(const char \*argz, size\_t argz\_len) друку argz.

**Код програми**

**FunctionArgz.h**

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

\*file: argz.h

\*synopsis: declarations for argz functions, types, constants

\*author: Bidyak Mikhailo

\*written: 26/09/2017

\*last modified: 24/10/2017

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

#ifndef ARGZ\_H //checking the connection heder

#define ARGZ\_H

/\*

Preprocessor directives that use the program.

Standard libraries and features.

\*/

#include <stdio.h> // need for stantart functions (printf and etc..)

#include <string.h> //need for strlen, strcmp, strcpy and etc.

#include <conio.h> // need for getch();

#include <stdlib.h> // need for malloc, calloc, realloc, exit, free and etc.

/\*

Special defined type. If "OK" (1) -> all OK and function work.

If "ERROR" -> something wrong.

\*/

typedef enum { OK, ERROR } error\_t;

//list of prototype functions

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

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

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

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

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

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

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

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

#endif /\* ARGZ\_H \*/

**ArgzTest.cpp**

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

\*file: argz\_test.c

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

\*related files: argz.c and argz.h

\*author: Bidyak Mikhailo

\*written: 26/09/2017

\*last modified: 24/10/2017

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

#include "FunctionArgz.h"

/\*

CUSTOM CODE

\*/

int main(void)

{

//initial values

char \*argz, \*ptr = NULL;

size\_t argz\_len = 0;

if (argz\_create\_sep("Func;Doing;Good", ';', &argz, &argz\_len) == OK) //call the SEP function

{

printf("Func started (argz\_create\_sep(\"Fucn;Doing;Good\", ';', &argz, &argz\_len):\n");

argz\_print(argz, argz\_len); //call the print function

printf("argz\_len = %d\n\n", argz\_len); //print the length of the vector

printf("Number of '\\0' calculated by argz\_count(argz, argz\_len) = %d\n\n", argz\_count(argz, argz\_len)); //call the COUNT function

if (argz\_add(&argz, &argz\_len, "ADD") == OK) //call the ADD function

{

printf("argz\_add(&argz, &argz\_len, \"ADD\"):\n");

argz\_print(argz, argz\_len); //call the print function

printf("\nargz\_len = %d\n\n", argz\_len); //print the length of the vector

argz\_delete(&argz, &argz\_len, "Good"); //call the DELETE function

printf("argz\_delete(&argz, &argz\_len, \"Good\")\n");

argz\_print(argz, argz\_len); //call the print function

printf("argz\_len = %d\n\n", argz\_len); //print the length of the vector

}

else exit(1); // exit with code (1) if something wrong

if (argz\_insert(&argz, &argz\_len, "ADD", "Working") == OK){ //call the INSERT function

printf("argz\_insert(&argz, &argz\_len,\"ADD\", \"Working\"):\n");

argz\_print(argz, argz\_len); //call the print function

printf("argz\_len = %d\n\n", argz\_len); //print the length of the vector

printf("argz\_next(argz, argz\_len, entry): \n"); //print symbols by the "NEXT" function

for (ptr = argz; ptr; ptr = argz\_next(argz, argz\_len, ptr))

printf("%s\n", ptr);

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

}

else exit(1); // exit with code (1) if something wrong

if (argz\_replace(&argz, &argz\_len, "Doing", "DO") == OK){ //call the REPLACE function

printf("argz\_replace(&argz, &argz\_len, \"Doing\", \"DO\"):\n");

argz\_print(argz, argz\_len); //call the print function

printf("argz\_len = %d\n\n", argz\_len); //print the length of the vector

}

else exit(1);// exit with code (1) if something wrong

}

else exit(1); // exit with code (1) if something wrong

\_getch(); // wait for keyword

return 0;

}

**FunctionArgz.cpp**

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

\*file: argz.c

\*synopsis: The argz functions use malloc/realloc to allocate/grow argz vectors,

\*and so any argz vector creating using these functions may be freed by using

\*free; conversely, any argz function that may grow a string expects that string

\*to have been allocated \*using malloc those argz functions that only examine

\*their arguments or modify them in place \*will work on any sort of memory). All \*argz functions that do memory allocation have a return type of error\_t, and

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

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

\*related files: argz\_test.c and argz.h

\*author: Bidyak Mikhailo

\*written: 26/09/2017

\*last modified: 24/10/2017

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

#include "FunctionArgz.h"

/\*

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)

{

/\*

check pointers. if some of them NULL -> end the program or send ERROR

It's need for the correct program operation.

\*/

if (\*argz == NULL) { return ERROR; }

if (\*string == NULL) { return ERROR; }

if ((sep < 0)&(sep>127)) { return ERROR; }

unsigned int i = 0;

const unsigned int string\_size = strlen(string); // legth our string

\*argz\_len = string\_size + 1; //change the length of the line

if ((\*argz = (char\*)malloc(sizeof(char)\*(string\_size + 1))) == NULL) //allocates memory for a new array of elements

exit(1); //end if memory allocation failed

for (i = 0; i < string\_size; i++)

{

if (\*(string + i) != sep)

\*(\*argz + i) = \*(string + i);

else

\*(\*argz + i) = '\0';

}

\*(\*argz + string\_size) = '\0';

return OK;

}

//Function returns the number of elements '/0' in the argz vector.

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

{

if (\*argz == NULL) { return 1; }

unsigned int i = 0, count = 0;

for (i = 0; i < argz\_len; i++)

if (\*(argz + i) == '\0') count++;

return count;

}

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

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

{

/\*

check pointers. if some of them NULL -> end the program.

It's need for the correct program operation.

\*/

if (\*argz == NULL) { return ERROR; }

if (\*str == NULL) { return ERROR; }

unsigned int i = 0;

const unsigned int str\_size = strlen(str);

char \*tmp = (char\*)malloc(sizeof(char)\*(\*argz\_len + str\_size + 1)); //allocates memory for a new array of elements

if (\*tmp == NULL) { return ERROR; } //end if memory allocation failed

for (i = 0; i < \*argz\_len; i++)

\*(tmp + i) = \*(\*argz + i);

for (i = 0; i < str\_size; i++)

\*(tmp + i + (\*argz\_len)) = \*(str + i);

\*argz\_len += str\_size + 1; //change the number of characters

\*(tmp + (\*argz\_len) - 1) = '\0';

free(\*argz); //remove the memory by the pointer

\*argz = tmp; //point to a new memory cell

return OK;

}

/\*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.

Note that as destructive argz functions usually reallocate their argz argument,

pointers into argz vectors such as entry will then become invalid.

\*/

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

{

unsigned int i = 0, k = 0;

char \*ptr = NULL;// pointer for pass through the fields

const unsigned int entry\_size = strlen(entry); //The length of the word which we need to delete

/\*

check pointers. if some of them NULL -> end the program or send ERROR

It's need for the correct program operation.

\*/

if (entry == NULL) { return; }

if (\*argz == NULL) { return; }

char \*tmp = (char\*)malloc((\*argz\_len - entry\_size) \* sizeof(char)); //allocates memory for a new array of elements

if (\*tmp == NULL) { return; } //end if memory allocation failed

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) //check the "entry" str in our list of fields

if (strcmp(entry, ptr) == 0) k++; //if we found str in fields, k>0

if (k == 0) { exit(1);}

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) {

if (strcmp(ptr, entry) != 0) {

strcpy(tmp + i, ptr);

i += strlen(ptr) + 1;

}

}

free(\*argz); //remove the memory by the pointer

\*argz = tmp; //point to a new memory cell

\*argz\_len = \*argz\_len - entry\_size - 1; //change the number of characters

}

/\*

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, char \*entry)

{

char \*ptr = NULL;

if (argz == NULL) exit(1);

if (entry == NULL) return argz;

for (ptr = entry; ptr <= argz + argz\_len; ptr++)

{

if (ptr == argz + argz\_len - 1) return NULL;

else if (\*ptr == '\0') return ptr + 1;

}

return NULL;

}

/\*

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. If before is 0, entry is added

to the end instead (as if by argz\_add). Since the first element is in fact the same as \*argz, passing in \*argz as

the value of before will result in entry being inserted at the beginning.

\*/

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

/\*

check pointers. if some of them NULL -> end the program or return ERROR.

It's need for the correct program operation.

\*/

if (before == NULL) { return ERROR; }

if (entry == NULL) { return ERROR; }

if (\*argz == NULL) { return ERROR; }

unsigned int i = 0, k = 0;

const unsigned int entry\_size = strlen(entry); //The length of the word which we insert

char \*ptr = NULL; // pointer for pass through the fields

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) //check the entry str in our list of fields

if (strcmp(before, ptr) == 0) k++; //if we found str in fields, k>0 and we can continue the program. (we need to replace this field)

if (k == 0) { return ERROR; }

char \*tmp = (char\*)malloc((\*argz\_len + entry\_size + 1) \* sizeof(char)); //allocates memory for a new array of elements

if (\*tmp == NULL) { return ERROR; } //end if memory allocation failed

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) {

if (strcmp(ptr, before) != 0) {

strcpy(tmp + i, ptr);

i += strlen(ptr) + 1;

}

else {

strcpy(tmp + i, entry);

i += strlen(entry);

strcpy(tmp + i, "\0");

i += 1;

strcpy(tmp + i, ptr);

i += strlen(ptr) + 1;

}

}

free(\*argz); //remove the memory by the pointer

\*argz = tmp; //point to a new memory cell

\*argz\_len = \*argz\_len + entry\_size + 1; //change the number of characters

return OK;

}

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

unsigned int i = 0, k = 0; // variable for check the entry str in our list of fields

const unsigned int str\_size = strlen(str), with\_size = strlen(with); //size of str which we have to replace & size of word which we have to insert

char \*ptr = NULL; // pointer for pass through the fields

if (str == NULL) { return ERROR; }

if (with == NULL) { return ERROR; } //check poiters. if some of them NULL -> end the program or return ERROR.

if (\*argz == NULL) { return ERROR; } //It's need for the correct program operation.

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) //check the entry str in our list of fields

if (strcmp(str, ptr) == 0) k++; //if we found str in fields, k>0 and we can continue the program. (we need to replace this field)

if (k == 0) { return ERROR; }

char \*tmp = (char\*)malloc(((\*argz\_len - str\_size) + with\_size) \* sizeof(char)); //allocates memory for a new array of elements

if (\*tmp == NULL) { return ERROR; } //end if memory allocation failed

while (ptr = argz\_next(\*argz, \*argz\_len, ptr)) {

if (strcmp(ptr, str) != 0) {

strcpy(tmp + i, ptr);

i += strlen(ptr) + 1;

}

else {

strcpy(tmp + i, with);

i += with\_size + 1;

strcpy(tmp + i, ptr);

}

}

free(\*argz); //remove the memory by the pointer

\*argz = tmp; //point to a new memory cell

\*argz\_len = \*argz\_len - str\_size + with\_size; //change the number of characters

return OK;

}

/\*

prints argz vector

this function prints all elements in this lab

\*/

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

{

unsigned int i = 0;

for ( i = 0; i < argz\_len; i++) //deriving element by element

{

if (argz[i] == '\0') printf("\\0\n"); // prints "\0"

else printf("%c", argz[i]); // prints symbols

}

}

**Тестування програми**

