****

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота № 4**

з дисципліни “ Основи програмування ”

## тема: “Модульне програмування, динамічні структури даних”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Виконав  студент I курсу  групи КП-61  Щербина Вадим Олегович  (*прізвище, ім’я, по батькові*)  варіант № 23 | |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 2017 р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) | |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  |   Київ 2016 | | | Сумарний бал:   |  | | --- | |  | |

**Мета роботи**

Навчитися працювати зі структурами, файлами, а також вдосконалити та закріпити вміння роботи з вказівниками.

Застосувати на практиці динамічне видалення пам’яті.

Навчитися оформлювати консольну програму для зручності роботи користувача.

Освоїти мультифайлове програмування, роботу з файлами .c і .h.

Навчитися створювати і використовувати динамічні структури даних. Навчитися розбивати програмний код на модулі і перевіряти їх роботу за допомогою модульних тестів.

**Загальні вимоги до завдання**

(Модифіковане завдання із лабораторної роботи №3)

Реалізувати консольну програму для управління даними типу Мова програмування. Після входу у програму користувач має можливість:

* Створити новий \_список елементів типу Мова програмування.
* Зчитати список елементів типу Мова програмування із дискового файлу.
* Зберегти змінений список елементів на файлову систему, запропонувавши йому ввести назву дискового файлу, у який збережуться дані у довільному фрматі.

Після цього список можна модифікувати:

* Додати новий елемент на довільну позицію.
* Видалити елемент із вказаної позиції.
* Обрати довільний елемент і замітини всі, чи тільки деякі значення його полів на нові.
* Знайти всі мови програмування створені до року Х

**Постановка завдання**

#### **Завдання №1. Створення динамічної структури даних** Список

Потрібно створити список елементів типу за варіантом і замінити всі випадки використання масиву даних із ЛР№3 на список даних. Описати всі необхідні типи і функції типу і організувати їх у окремий модуль Список.

#### Завдання №2. Виділення модулів

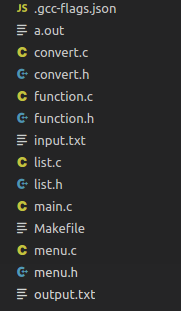
Розбити всю кодову базу програми на модулі:

1. Список даних
2. Модуль конвертації даних
3. Сховище даних (файлова система)
4. Консольний інтерфейс користувача (CUI)

#### Завдання №3. Модульні тести

За фопомогою фреймворку [Check][chack] (або аналогічного) описати модульні тести для тестування модуля конвертації даних.

**Структура програми**



**Тексти коду програм**

|  |
| --- |
| Makefile |
| *# Makefile for Lab\_4 project*    **all:**  gcc -c -g -std**=**c11 -Werror -Wall -pedantic-errors -I./include src/\*.c \*.c -I./ -I../include  gcc \*.o -o Lab\_4 -lprogbase  rm \*.o  gnome-terminal --tab -e "/bin/bash -c 'valgrind --leak-check=yes ./Lab\_4; exec /bin/bash -i'" |

|  |
| --- |
| list.h |
| **#ifndef LIST\_H**  **#define LIST\_H**    **#include** <stdio.h>  **#include** <stdbool.h>    **typedef** **struct** ListNode ListNode;  **typedef** **struct** List List;    List **\*** **List\_new**(**void**);  **void** **List\_free**(List **\*\*** self);    ListNode **\*** **ListNode\_new**(**void** **\*** data);  **void** **ListNode\_free**(ListNode **\*\*** self);  **void** **List\_freeData**(**void** **\*\*** selfPtr);    **void** **List\_addFirst**(List **\*** self, **void** **\*** data);  **void** **List\_addLast**(List **\*** self, **void** **\*** data);  **int** **List\_insert**(List **\*** self, **int** position, **void** **\*** data);    **void** **List\_removeFirst**(List **\*** self);  **void** **List\_removeLast**(List **\*** self);  **void** **List\_removeAt**(List **\*** self, **int** position);  **int** **List\_count**(List **\*** self);    ListNode **\*** **List\_elementAt**(List **\*** self, **int** position);    **void** **\*** **List\_getProgLang**(ListNode **\*** self);  **void** **List\_overwriteData**(ListNode **\*** self, **void** **\*** data);  **void** **List\_clear**(List **\*** self);    **bool** **List\_strToNode**(List **\*** self, **int** position, **const** **char** **\*** str);  **bool** **List\_overwriteNode**(List **\*** self, **int** position, **const** **char** **\*** str);  **int** **List\_overwriteNodeFromNode**(List **\*** self, **int** firstIndex, **int** secondIndex);      **#endif** |

|  |
| --- |
| convert.h |
| **#ifndef CONVERT\_H**  **#define CONVERT\_H**    **#include** "list.h"    **typedef** **struct** ProgLang ProgLang;    ProgLang **\*** **ProgLang\_new**(**const** **char** **\***);  **void** **ProgLang\_free**(**void** **\*\*** selfPtr);    **char** **\*** **ProgLang\_toString**(ProgLang **\*** self, **char** **\*** buffer);    **void** **ProgLang\_print**(ProgLang **\*** self);    **const** **char** **\*** **ProgLang\_getFullname**(ProgLang **\*** self);  **int** **ProgLang\_getDate**(ProgLang **\*** self);  **float** **ProgLang\_getMarketShare**(ProgLang **\*** self);  **const** **char** **\*** **ProgLang\_getMajorImplementation**(ProgLang **\*** self);  **float** **ProgLang\_getSatisfactionIndex**(ProgLang **\*** self);    **void** **ProgLang\_setFullname**(ProgLang **\*** self, **const** **char** **\*** newName);  **void** **ProgLang\_setDate**(ProgLang **\*** self, **int** newDate);  **void** **ProgLang\_setMarketShare**(ProgLang **\*** self, **float** newMarketShare);  **void** **ProgLang\_setMajorImplementation**(ProgLang **\*** self, **const** **char** **\*** newValue);  **void** **ProgLang\_setSatisfactionIndex**(ProgLang **\*** self, **float** newIndex);    **bool** **ProgLang\_overwriteField**(ListNode **\*** self, **int** field, **const** **char** **\*** str);  **#endif** |
| storage.h |
| **#ifndef STORAGE\_H**  **#define STORAGE\_H**    **#include** <stdbool.h>  **#include** "list.h"    List **\*** **Storage\_load**(**const** **char** **\*** fileName);  **bool** **Storage\_save**(**const** **char** **\*** fileName, List **\*** self);    **void** **Storage\_printList**(List **\*** self, **FILE** **\*** fout);  **void** **Storage\_printNode**(ListNode **\*** node, **FILE** **\*** fout);    **#endif** |
| cui.h |
| **#ifndef CUI\_H**  **#define CUI\_H**    **#include** <stdio.h>  **#include** "list.h"    **void** **CUI\_consoleImage** (**void**);  **void** **CUI\_menuChoice**(**char** command, **int** **\*** choice, **int** limit, **char** menu[][300], **int** length);  **void** **CUI\_menuTasks**(**char** menu [][300], **int** clear, **int** choice, **int** prev, **int** length);  **void** **CUI\_mainMenu**(**char** menu [][300], **int** clear, **int** choice, **int** prev, **int** length);  **void** **CUI\_endProg**(**void**);    **void** **CUI\_printList**(List **\*** self);  **void** **CUI\_printNode**(ListNode **\*** node);  **void** **CUI\_controlPrint**(List **\*** self);  **void** **CUI\_printNodeInMenu**(ListNode **\*** node, **int** x, **int** y);  **void** **CUI\_printStr**(**const** **char** **\*** str , **int** x, **int** y, **const** **char** **\*** color);  **void** **CUI\_listInfo**(**int** numberOfNode);  **void** **CUI\_findLanguages**(List **\*** self, **int** date);    **#endif** |
| valid.h |
| **#ifndef VALID\_H**  **#define VALID\_H**    **#include** "list.h"  **#include** <stdbool.h>      **bool** **isValidStr**(**const** **char** **\*** str);    **bool** **checkInt**(**const** **char** **\*** str);  **bool** **checkFloat**(**const** **char** **\*** str);    **#endif** |
| list.c |
| **#include** <stdio.h>  **#include** <stdlib.h>  **#include** <assert.h>  **#include** <stdbool.h>    **#include** <convert.h>  **#include** <list.h>  **#include** <valid.h>  **#include** <cui.h>    **#define throw(MSG) assert(0 && MSG);**    **struct** ListNode {  **void** **\*** data;  ListNode **\*** next;  };    **struct** List {  ListNode **\*** head;  };    List **\*** **List\_new**(**void**) {  **struct** List **\*** list **=** (**struct** List **\***)malloc(**sizeof**(**struct** List));  list**->**head **=** NULL;  **return** list;  }    **void** **List\_free**(List **\*\*** self) {  assert(NULL **!=** self);  free(**\***self);  **\***self **=** NULL;  }      **struct** ListNode **\*** **ListNode\_new**(**void** **\*** data) {  **struct** ListNode **\*** node **=** (**struct** ListNode **\***) malloc(**sizeof**(**struct** ListNode));  node**->**next **=** NULL;  node**->**data **=** data;  **return** node;  }    **void** **ListNode\_free**(ListNode **\*\*** self) {  assert(NULL **!=** self);  List\_freeData(**&**(**\***self)**->**data);  free(**\***self);  **\***self **=** NULL;  }    **void** **List\_freeData**(**void** **\*\*** selfPtr) {  assert(NULL **!=** selfPtr);  free(**\***selfPtr);  **\***selfPtr **=** NULL;  }    **void** **List\_addFirst**(List **\*** self, **void** **\*** data) {  **struct** ListNode **\*** node **=** ListNode\_new(data);  node**->**next **=** self**->**head;  self**->**head **=** node;  }    **void** **List\_addLast**(List **\*** self, **void** **\*** data) {  **struct** ListNode **\*** node **=** ListNode\_new(data);  **if** (self**->**head **==** NULL) {  self**->**head **=** node;  **return**;  }  **struct** ListNode **\*** cur **=** self**->**head;  **while** (cur**->**next **!=** NULL) {  cur **=** cur**->**next;  }  cur**->**next **=** node;  }    **int** **List\_insert**(List **\*** self, **int** position, **void** **\*** data) {  assert(position **>=** 0);  **if** (position **==** 0) {  List\_addFirst(self, data);  **return** 0;  }  **struct** ListNode **\*** node **=** ListNode\_new(data);  **int** i **=** 0;  **struct** ListNode **\*** cur **=** self**->**head;  **while** (cur**->**next **!=** NULL **&&** i **!=** position **-** 1) {  cur **=** cur**->**next;  i**++**;  }  **if** (i **!=** position **-** 1) **return** 1;  node**->**next **=** cur**->**next;  cur**->**next **=** node;  **return** 0;  }    **void** **List\_removeFirst**(**struct** List **\*** self) {  **struct** ListNode **\*** node **=** self**->**head;  **if** (node **==** NULL) throw("NULL reference");  self**->**head **=** node**->**next;  ListNode\_free(**&**node);  }      **void** **List\_removeLast**(List **\*** self) {  **struct** ListNode **\*** cur **=** self**->**head;  **if** (cur **==** NULL) throw("NULL reference");  **if** (cur**->**next **==** NULL) {  ListNode\_free(**&**cur);  self**->**head **=** NULL;  }  **while** (cur**->**next**->**next **!=** NULL) {  cur **=** cur**->**next;  }  ListNode\_free(**&**(cur**->**next));  cur**->**next **=** NULL;  }    **void** **List\_removeAt**(List **\*** self, **int** index) {  assert(index **>=** 0);  assert(index **<** List\_count(self));  **if** (index **==** 0) {  List\_removeFirst(self);  **return**;  }  **int** i **=** 0;  **struct** ListNode **\*** cur **=** self**->**head;  **while** (cur**->**next**->**next **!=** NULL **&&** i **!=** index **-** 1) {  i **+=** 1;  cur **=** cur**->**next;  }  **if** (i **!=** index **-** 1) throw("Index out of bounds");  **struct** ListNode **\*** node **=** cur**->**next;  cur**->**next **=** node**->**next;  ListNode\_free(**&**node);  }    **int** **List\_count**(**struct** List **\*** self) {  **int** count **=** 0;  **struct** ListNode **\*** cur **=** self**->**head;  **while** (cur **!=** NULL) {  count **+=** 1;  cur **=** cur**->**next;  }  **return** count;  }    **struct** ListNode **\*** **List\_elementAt**(List **\*** self, **int** position) {  assert(position **>=** 0);  **if** (self**->**head **==** NULL) throw("list->head = NULL");  **struct** ListNode **\*** cur **=** self**->**head;  **int** count **=** 0;  **while** (cur **!=** NULL **&&** count **!=** position) {  cur **=** cur**->**next;  count**++**;  }  **return** cur;  }    **void** **\*** **List\_getProgLang**(ListNode **\*** self) {  **return** self**->**data;  }    **void** **List\_overwriteData**(ListNode **\*** self, **void** **\*** data) {  **if** (self **==** NULL) throw("NULL reference");  ProgLang\_free(**&**self**->**data);  self**->**data **=** data;  }    **void** **List\_clear**(List **\*** self) {  **struct** ListNode **\*** cur **=** self**->**head;  **while** (cur **!=** NULL) {  **struct** ListNode **\*** node **=** cur;  cur **=** cur**->**next;  ListNode\_free(**&**node);  }  }    **bool** **List\_strToNode**(List **\*** self, **int** position, **const** **char** **\*** str) {  **if** (isValidStr(str) **==** false) {  **return** false;  }  List\_insert(self, position, ProgLang\_new(str));  **return** true;  }    **bool** **List\_overwriteNode**(List **\*** self, **int** position, **const** **char** **\*** str) {  **if** (isValidStr(str) **==** false) {  **return** false;  }  **struct** ListNode **\*** node **=** List\_elementAt(self, position);  ProgLang **\*** new **=** ProgLang\_new(str);  List\_overwriteData(node, new);  **return** true;  }    **int** **List\_overwriteNodeFromNode**(List **\*** self, **int** firstIndex, **int** secondIndex) {  **if** (firstIndex **<** 0 **||** firstIndex **>=** List\_count(self)) {  **return** 1;  } **else**  **if** (secondIndex **<** 0 **||** secondIndex **>=** List\_count(self)) {  **return** 2;  }  **else** {  **char** str[100];  ProgLang\_toString(List\_getProgLang(List\_elementAt(self, secondIndex)), str);  ProgLang **\*** data **=** ProgLang\_new(str);  List\_overwriteData(List\_elementAt(self, firstIndex), data);  }  **return** 0;  } |

|  |
| --- |
| convert.c |
| **#include** <stdlib.h>  **#include** <stdio.h>  **#include** <string.h>  **#include** <convert.h>  **#include** <assert.h>    **#include** <list.h>  **#include** <stdbool.h>  **#include** <valid.h>    **struct** ProgLang {  **char** fullname[200];  **int** date\_of\_occurrence;  **float** market\_share;  **struct** {  **char** major\_implementation[200];  **float** satisfaction\_index;  } add\_info;  };    ProgLang **\*** **ProgLang\_new**(**const** **char** **\*** str) {  ProgLang **\*** self **=** malloc(**sizeof**(ProgLang));  sscanf(str, "%s %i %f %s %f", self**->**fullname, **&**self**->**date\_of\_occurrence, **&**self**->**market\_share, \  self**->**add\_info.major\_implementation, **&**self**->**add\_info.satisfaction\_index);  **return** self;  }    **void** **ProgLang\_free**(**void** **\*\*** selfPtr) {  assert(NULL **!=** selfPtr);  free(**\***selfPtr);  **\***selfPtr **=** NULL;  }    **char** **\*** **ProgLang\_toString**(ProgLang **\*** self, **char** **\*** buffer) {  sprintf(buffer, "%s %i %.2f %s %.2f", self**->**fullname, self**->**date\_of\_occurrence, self**->**market\_share, \  self**->**add\_info.major\_implementation, self**->**add\_info.satisfaction\_index);  **return** buffer;  }    **void** **ProgLang\_print**(ProgLang **\*** self) {  printf("Fullname: %s\n", self**->**fullname);  printf("Date of occurrence: %i\n", self**->**date\_of\_occurrence);  printf("Market share: %.2f\n", self**->**market\_share);  printf("Major implementation: %s\n", self**->**add\_info.major\_implementation);  printf("Satisfaction index: %.2f\n", self**->**add\_info.satisfaction\_index);  }    **const** **char** **\*** **ProgLang\_getFullname**(ProgLang **\*** self) {  **return** self**->**fullname;  }    **int** **ProgLang\_getDate**(ProgLang **\*** self) {  **return** self**->**date\_of\_occurrence;  }    **float** **ProgLang\_getMarketShare**(ProgLang **\*** self) {  **return** self**->**market\_share;  }    **const** **char** **\*** **ProgLang\_getMajorImplementation**(ProgLang **\*** self) {  **return** self**->**add\_info.major\_implementation;  }    **float** **ProgLang\_getSatisfactionIndex**(ProgLang **\*** self) {  **return** self**->**add\_info.satisfaction\_index;  }    **void** **ProgLang\_setFullname**(ProgLang **\*** self, **const** **char** **\*** newName) {  sscanf(newName, "%s", self**->**fullname);  }    **void** **ProgLang\_setDate**(ProgLang **\*** self, **int** newDate) {  self**->**date\_of\_occurrence **=** newDate;  }    **void** **ProgLang\_setMarketShare**(ProgLang **\*** self, **float** newMarketShare) {  self**->**market\_share **=** newMarketShare;  }    **void** **ProgLang\_setMajorImplementation**(ProgLang **\*** self, **const** **char** **\*** newValue) {  sscanf(newValue, "%s", self**->**add\_info.major\_implementation);  }    **void** **ProgLang\_setSatisfactionIndex**(ProgLang **\*** self, **float** newIndex) {  self**->**add\_info.satisfaction\_index **=** newIndex;  }    **bool** **ProgLang\_overwriteField**(**struct** ListNode **\*** self, **int** field, **const** **char** **\*** str) {  ProgLang **\*** data **=** List\_getProgLang(self);  **switch**(field) {  **case** 0**:**  ProgLang\_setFullname(data, str);  **break**;  **case** 1**:**  **if** (checkInt(str) **==** true)  ProgLang\_setDate(data, atoi(str));  **else** **return** false;  **break**;  **case** 2**:**  **if** (checkFloat(str) **==** true)  ProgLang\_setMarketShare(data, atof(str));  **else** **return** false;  **break**;  **case** 3**:**  ProgLang\_setMajorImplementation(data, str);  **break**;  **case** 4**:**  **if** (checkFloat(str) **==** true **&&** atof(str) **<=** 1)  ProgLang\_setSatisfactionIndex(data, atof(str));  **else** **return** false;  **break**;  }  **return** true;  } |
| [storage.c](https://bitbucket.org/Valzavator/progbase2/src/5553ab88b9079e5ac6cbb344c427ecd625c894fd/labs/lab4/src/storage.c?at=master) |
| **#include** <stdio.h>  **#include** <string.h>  **#include** <list.h>  **#include** <cui.h>  **#include** <assert.h>  **#include** <storage.h>  **#include** <stdbool.h>  **#include** <convert.h>    **#define throw(MSG) assert(0 && MSG);**    List **\*** **Storage\_load**(**const** **char** **\*** fileName) {  List **\*** self;  **FILE** **\*** fin **=** fopen(fileName, "r");  **char** buffer[200];  **if** (fin **==** NULL) **return** NULL;  self **=** List\_new();  **while**(fgets(buffer, 200, fin) **!=** NULL) {  buffer[strlen(buffer) **-** 1] **=** '\0';  List\_strToNode(self, List\_count(self), buffer);  }  fclose(fin);  **return** self;  }    **bool** **Storage\_save**(**const** **char** **\*** fileName, List **\*** self) {  **FILE** **\*** fout **=** fopen(fileName, "w");  **if** (fout **==** NULL) {  **return** false;  }  Storage\_printList(self, fout);  fclose(fout);  **return** true;  }    **void** **Storage\_printList**(List **\*** self, **FILE** **\*** fout) {  **int** count **=** List\_count(self);  **int** i **=** 0;  **for** (i **=** 0; i **<** count; i**++**) {  Storage\_printNode(List\_elementAt(self, i), fout);  }  }    **void** **Storage\_printNode**(ListNode **\*** node, **FILE** **\*** fout) {  **if** (node **==** NULL) throw("NULL reference");  **char** buffer[100];  ProgLang\_toString(List\_getProgLang(node), buffer);  fprintf(fout, "%s\n", buffer);  } |
| cui.c |
| **#include** <pbconsole.h>  **#include** <assert.h>  **#include** <string.h>  **#include** <cui.h>  **#include** <list.h>  **#include** <convert.h>    **#define throw(MSG) assert(0 && MSG);**    **void** **CUI\_consoleImage** (**void**) {  **const** **char** colorsTable[16][2] **=** {  {'1', BG\_YELLOW},  {'2', BG\_BLUE},  };  **int** i;  **int** j;  **int** colorPairIndex **=** 0;  **int** colorsTableLength **=** **sizeof**(colorsTable) **/** **sizeof**(colorsTable[0]);  **int** image[8][38] **=** {  { '1','1','2','2','2','2', '2','2','2','2', '2','2','1','1','1','2','2', '2','2','2','2', '1','1','1','1','1','2', '2','2','2','2', '1','1','2','2','2','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '2','1','1','1','1','1','2', '2','2','2','2', '1','1','2','2','1','1', '2','2','2','2', '1','1','2','2','2','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '1','1','1','2','1','1','1', '2','2','2','2', '1','1','2','2','1','1', '2','2','2','2', '1','1','2','2','2','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '1','1','2','2','2','1','1', '2','2','2','2', '1','1','1','1','1','2', '2','2','2','2', '1','1','2','2','2','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '1','1','2','2','2','1','1', '2','2','2','2', '1','1','2','2','1','1', '2','2','2','2', '1','1','1','1','1','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '1','1','1','1','1','1','1', '2','2','2','2', '1','1','2','2','1','1', '2','2','2','2', '2','2','2','2','2','1','1'},  { '1','1','2','2','2','2', '2','2','2','2', '1','1','2','2','2','1','1', '2','2','2','2', '1','1','2','2','1','1', '2','2','2','2', '2','2','2','2','2','1','1'},  { '1','1','1','1','1','1', '2','2','2','2', '1','1','2','2','2','1','1', '2','2','2','2', '1','1','1','1','1','1', '2','2','2','2', '2','2','2','2','2','1','1'},  };  **for** (i **=** 0; i **<** 10; i**++**) {  **for**(j **=** 0; j **<** 118; j**++**) {  conMove(i **+** 2, j **+** 2);  conSetAttr(BG\_BLUE);  putchar(' ');  conReset();  }  }  **for** (i **=** 0; i **<** 8; i**++**)  {  **for** (j **=** 0; j **<** 38; j**++**)  {  **char** colorCode **=** '\0';  **char** color **=** '\0';  */\* get current color code from colorsPalette \*/*  colorCode **=** image[i][j];  */\* find corresponding color in table \*/*  **for** (colorPairIndex **=** 0; colorPairIndex **<** colorsTableLength; colorPairIndex**++**)  {  **char** colorPairCode **=** colorsTable[colorPairIndex][0];  **char** colorPairColor **=** colorsTable[colorPairIndex][1];  **if** (colorCode **==** colorPairCode)  {  color **=** colorPairColor;  **break**; */\* we have found our color, break the loop \*/*  }  }  */\* print space with founded color background \*/*  conSetAttr(color);  conMove(3 **+** i, 41 **+** j);  putchar(' ');  }  conReset();  puts("");  }  conSetAttr(BG\_RED);  **for** (i **=** 0; i **<** 37; i**++**) {  conMove(i **+** 1, 1);  putchar(' ');  conMove(i **+** 1, 121);  putchar(' ');  }  **for** (i **=** 0; i **<** 120; i**++**) {  conMove(1, i **+** 1);  putchar(' ');  conMove(12, i **+** 1);  putchar(' ');  conMove(22, i **+** 1);  putchar(' ');  conMove(38, i **+** 1);  putchar(' ');  }  conSetAttr(FG\_BLACK);  conSetAttr(BG\_WHITE);  conMove(12, 45);  printf("CHOOSE TASK (use \"w\" or \"s\")");  conMove(22, 56);  printf("DISPLAY");  conReset();  }    **void** **CUI\_menuChoice**(**char** command, **int** **\*** choice, **int** limit, **char** menu[][300], **int** length) {  **int** prev **=** 1;  **if** (command **==** 'w') {  **\***choice **=** **\***choice **-** 1;  prev **=** **\***choice **+** 1;  }  **if** (command **==** 's') {  **\***choice **=** **\***choice **+** 1;  prev **=** **\***choice **-** 1;  }  **if** (**\***choice **>** limit) {  **\***choice **=** 0;  }  **if** (**\***choice **<** 0) {  **\***choice **=** limit;  }    **if** (prev **>** limit) {  prev **=** 0;  }  **if** (prev **<** 0) {  prev **=** limit;  }  **if** (limit **==** 10) {  CUI\_menuTasks(menu, 0, **\***choice, prev, length);  } **else**  **if** (limit **==** 1) {  CUI\_mainMenu(menu, 0, **\***choice, prev, length);  } **else**  **if** (limit **==** 4) {  conMove(24, 55);  printf("%s", menu[**\***choice]);  }  }    **void** **CUI\_menuTasks**(**char** menu [][300], **int** clear, **int** choice, **int** prev, **int** length) {  **int** i **=** 0;  **enum** {x **=** 14, y1 **=** 8, y2 **=** 58};  **if** (clear **==** 1) {  conClear();  CUI\_consoleImage();  **for** (i **=** 0; i **<** length; i**++**) {  **if** (i **/** 5 **==** 0) {  conMove(x **+** i, y1);  } **else**  **if** (i **==** 10) {  conMove(20, 51);  } **else** {  conMove(x **+** i **-** 5, y2);  }  puts(menu[i]);  }  }  **if** (prev **/** 5 **==** 0) {  conMove(x **+** prev, y1);  } **else**  **if** (prev **==** 10) {  conMove(20, 51);  } **else** {  conMove(x **+** prev **-** 5, y2);  }  puts(menu[prev]);  conSetAttr(BG\_BLACK);  **if** (choice **/** 5 **==** 0) {  conMove(x **+** choice, y1);  } **else**  **if** (choice **==** 10) {  conMove(20, 51);  } **else** {  conMove(x **+** choice **-** 5, y2);  }  puts(menu[choice]);  conReset();  }    **void** **CUI\_mainMenu**(**char** menu [][300], **int** clear, **int** choice, **int** prev, **int** length) {  **int** i **=** 0;  **enum** {x **=** 14, y **=** 44};  **if** (clear **==** 1) {  conClear();  CUI\_consoleImage();  **for** (i **=** 0; i **<** length; i**++**) {  conMove(x **+** i, y);  puts(menu[i]);  }  }  conMove(x **+** prev, y);  puts(menu[prev]);  conSetAttr(BG\_BLACK);  conMove(x **+** choice, y);  puts(menu[choice]);  conReset();  }    **void** **CUI\_endProg**(**void**) {  **const** **char** colorsTable[16][2] **=** {  {'0', BG\_BLACK},  {'1', BG\_INTENSITY\_BLACK},  {'2', BG\_BLUE},  {'3', BG\_YELLOW},  {'4', BG\_GREEN},  {'5', BG\_MAGENTA},  {'6', BG\_CYAN},  {'7', BG\_INTENSITY\_CYAN},  {'8', BG\_RED},  {'9', BG\_INTENSITY\_RED},  {'B', BG\_INTENSITY\_YELLOW},  {'A', BG\_INTENSITY\_GREEN},  {'C', BG\_INTENSITY\_BLUE},  {'D', BG\_INTENSITY\_MAGENTA},  {'E', BG\_WHITE},  {'F', BG\_INTENSITY\_WHITE}  };  **int** i, j;  **int** colorPairIndex **=** 0;  **int** colorsTableLength **=** **sizeof**(colorsTable) **/** **sizeof**(colorsTable[0]);  **char** image[28][28] **=** {  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { 'F','F','F','F','F','F','F','F','F','F','0','0','0','0','0','0','0','0','0','0','0','F','F','F','F','F','F','F' },  { 'F','F','F','F','F','F','F','F','0','0','F','F','F','F','F','F','F','F','F','F','0','0','0','0','F','F','F','F' },  { 'F','F','F','F','F','0','0','0','F','F','F','F','F','F','F','0','F','F','F','F','F','F','F','0','0','0','0','F' },  { 'F','F','F','0','0','F','F','F','F','F','F','F','F','F','F','0','F','F','F','F','F','F','F','F','F','F','0','0' },  { 'F','0','0','F','F','F','F','F','F','0','F','F','F','F','F','0','F','F','F','F','F','F','F','F','F','F','F','F' },  { '0','F','F','F','F','F','F','F','0','F','0','C','F','F','F','F','0','F','F','0','F','F','F','F','F','F','F','F' },  { 'F','0','0','F','F','F','F','F','0','0','0','C','0','F','F','F','0','F','F','0','F','F','F','F','F','F','F','F' },  { 'F','F','F','0','F','F','F','F','0','0','C','C','F','0','F','F','F','0','F','F','0','F','F','F','F','F','F','F' },  { 'F','F','F','F','0','F','F','F','F','0','C','C','F','0','F','F','F','0','F','F','0','F','F','F','F','F','F','F' },  { 'F','F','F','F','0','F','F','F','F','F','C','C','0','3','0','F','F','F','0','F','F','0','F','F','F','F','F','F' },  { 'F','F','F','0','F','F','F','F','F','0','F','C','F','0','F','F','F','F','0','0','0','0','F','F','F','F','F','F' },  { 'F','0','0','F','F','F','F','F','0','F','0','C','F','0','F','F','F','F','F','F','F','F','F','F','F','F','F','F' },  { '0','F','F','F','F','F','F','F','0','0','0','C','0','F','F','F','F','F','0','0','0','0','0','0','0','0','0','0' },  { 'F','0','0','F','F','F','F','F','C','C','C','C','F','F','F','F','F','F','F','F','F','F','F','F','0','0','F','F' },  { 'F','F','F','0','0','0','0','0','C','C','C','C','0','0','0','0','0','0','0','0','0','0','0','0','0','F','F','F' },  { 'F','F','F','F','F','C','C','C','C','C','C','C','C','C','C','F','F','F','F','F','F','F','F','F','F','F','F','F' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' },  { '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1' }  };  **int** length\_i **=** **sizeof**(image) **/** **sizeof**(image[0]);  **int** length\_j **=** **sizeof**(image[0]) **/** **sizeof**(image[0][0]);  conClear();  **for** (i **=** 2; i **<** length\_i **-** 2; i**++**) {  **for** (j **=** 0; j **<** length\_j; j**++**) {  **char** colorCode **=** '\0';  **char** color **=** '\0';  */\* get current color code from colorsPalette \*/*  colorCode **=** image[i][j];  */\* find corresponding color in table \*/*  **for** (colorPairIndex **=** 0; colorPairIndex **<** colorsTableLength; colorPairIndex**++**) {  **char** colorPairCode **=** colorsTable[colorPairIndex][0];  **char** colorPairColor **=** colorsTable[colorPairIndex][1];  **if** (colorCode **==** colorPairCode) {  color **=** colorPairColor;  **break**; */\* we have found our color, break the loop \*/*  }  }  */\* print space with founded color background \*/*  conSetAttr(color);  conMove(i **+** 8, j **+** 46);  putchar(' ');  }  conReset();  puts("");  }  puts("");  conSetAttr(FG\_RED);  **for** (i **=** 0; i **<** 125; i**++**) {  conMove(34, 1 **+** i);  putchar('=');  conMove(36, 1 **+** i);  putchar('=');  }  CUI\_printStr("Goodbye", 35, 55, "FG\_YELLOW");  conShowCursor();  conMove(37, 1);  }    **void** **CUI\_printList**(List **\*** self) {  **int** count **=** List\_count(self);  **int** i **=** 0;  **if** (count **==** 0) {  CUI\_printStr("Список порожній!", 1, 54, "FG\_YELLOW");  **return**;  }  **for** (i **=** 0; i **<** count; i**++**) {  CUI\_printNode(List\_elementAt(self, i));  }  }    **void** **CUI\_printNode**(ListNode **\*** node) {  **if** (node **==** NULL) throw("NULL reference");  ProgLang\_print(List\_getProgLang(node));  puts("");  }    **void** **CUI\_controlPrint**(List **\*** self) {  **char** commandExit **=** '1';  **int** position **=** 0;  **int** index **=** 0;  **int** countNode **=** 0;  **while**(commandExit **!=** '\n') {  conClear();  index **=** position;  **while**(List\_elementAt(self, index) **!=** NULL **&&** countNode **<=** 5) {  CUI\_printNode(List\_elementAt(self, index));  index**++**;  countNode**++**;  }  countNode **=** 0;  CUI\_printStr("Натисніть \"Enter\" щоб продовжити", 37, 48, "FG\_GREEN");  commandExit **=** conGetChar();  **if** (commandExit **==** 'w') {  **if**(position **!=** 0) position**--**;  }  **if** (commandExit **==** 's') {  **if** (position **!=** List\_count(self) **-** 1 **&&** List\_count(self) **-** position **>=** 6) position**++**;  }  }  }    **void** **CUI\_printNodeInMenu**(ListNode **\*** node, **int** x, **int** y) {  **if** (node **!=** NULL) {  **const** **char** **\*** fullname **=** ProgLang\_getFullname(List\_getProgLang(node));  **int** date **=** ProgLang\_getDate(List\_getProgLang(node));  **float** marketShare **=** ProgLang\_getMarketShare(List\_getProgLang(node));  **const** **char** **\*** majorImplementation **=** ProgLang\_getMajorImplementation(List\_getProgLang(node));  **float** satisfactionIndex **=** ProgLang\_getSatisfactionIndex(List\_getProgLang(node));  conMove(x, y);  printf("Fullname: %s", fullname);  conMove(x **+** 1, y);  printf("Date of occurrence: %i", date);  conMove(x **+** 2, y);  printf("Market share: %.2f", marketShare);  conMove(x **+** 3, y);  printf("Major implementation: %s", majorImplementation);  conMove(x **+** 4, y);  printf("Satisfaction index: %.2f", satisfactionIndex);  } **else** {  conMove(x, y);  conSetAttr(FG\_YELLOW);  printf("Вузла з даним індексом не існує!\n");  conReset();  }  }    **void** **CUI\_printStr**(**const** **char** **\*** str , **int** x, **int** y, **const** **char** **\*** color) {  conMove(x, y);  **if** (strcmp(color, "FG\_GREEN") **==** 0) {  conSetAttr(FG\_GREEN);  } **else**  **if** (strcmp(color, "FG\_YELLOW") **==** 0) {  conSetAttr(FG\_YELLOW);  }  printf("%s", str);  conReset();  }    **void** **CUI\_listInfo**(**int** numberOfNode) {  conMove(1, 48);  conSetAttr(BG\_WHITE);  conSetAttr(FG\_BLACK);  printf(" Number of Nodes: %i ", numberOfNode);  conReset();  }    **void** **CUI\_findLanguages**(List **\*** self, **int** date) {  **int** i **=** 0;  List **\*** list **=** List\_new();  **for** (i **=** 0; i **<** List\_count(self); i**++**) {  ListNode **\*** cur **=** List\_elementAt(self, i);  **int** curDate **=** ProgLang\_getDate(List\_getProgLang(cur));  **if** (curDate **<** date) {  **char** buffer[100];  ProgLang\_toString(List\_getProgLang(cur), buffer);  ProgLang **\*** new **=** ProgLang\_new(buffer);  List\_addLast(list, new);  }  }  **if** (List\_count(list) **!=** 0) {  CUI\_controlPrint(list);  List\_clear(list);  } **else** {  CUI\_printStr("Список порожній!", 1, 54, "FG\_YELLOW");  CUI\_printStr("Натисніть \"Enter\" щоб продовжити", 37, 48, "FG\_GREEN");  conGetChar();  }  List\_free(**&**list);  } |
| valid.c |
| **#include** <stdio.h>  **#include** <ctype.h>  **#include** <stdbool.h>    **bool** **isValidStr**(**const** **char** **\*** str) {  **char** buffer[200];  **int** date\_of\_occurrence **=** 0;  **float** market\_share **=** 0;  **float** satisfaction\_index **=** 0;  sscanf(str, "%s %i %f %s %f", buffer, **&**date\_of\_occurrence, **&**market\_share, \  buffer, **&**satisfaction\_index);  **if** (date\_of\_occurrence **<=** 0 **||** market\_share **<** 0.01 **||** satisfaction\_index **<** 0.01 **||** satisfaction\_index **>** 1) {  **return** false;  }  **return** true;  }    **bool** **checkInt**(**const** **char** **\*** str) {  **int** i **=** 0;  **if** (**\*** str **==** '\0' **||** **\*** str **==** '\n') {  **return** 1;  }  **while** (**\***(str **+** i) **!=** '\0' **&&** **\***(str **+** i) **!=** '\n') {  **if** (isdigit(**\***(str **+** i)) **==** 0) {  **return** false;  }  i**++**;  }  **return** true;  }    **bool** **checkFloat**(**const** **char** **\*** str) {  **int** i **=** 0;  **int** flag **=** 0;  **if** (**\*** str **==** '\0' **||** **\*** str **==** '\n') {  **return** false;  }  **while** (**\***(str **+** i) **!=** '\0' **&&** **\***(str **+** i) **!=** '\n') {  **if** (isdigit(**\***(str **+** i)) **==** 0 **&&** **\***(str **+** i) **!=** '.') {  **return** false;  }  **if** (**\***(str **+** i) **==**'.' **&&** flag **==** 1) {  **return** false;  }  **if** (**\***(str **+** i) **==** '.' **&&** flag **==** 0) {  flag **=** 1;  }  i**++**;  }  **return** true;  } |
| main.c |
| **#include** <stdio.h>  **#include** <stdlib.h>  **#include** <string.h>  **#include** <pbconsole.h>  **#include** <stdbool.h>    **#include** <valid.h>  **#include** <cui.h>  **#include** <list.h>  **#include** <storage.h>  **#include** <convert.h>    **int** **main**(**int** argc, **char** **const** **\***argv[]) {  **char** menu[2][300] **=** {  {"1. Створити новий масив даних."},  {"2. Зчитати масив даних із файлу."}  };  **char** menu\_funct [11][300] **=** {  {"1. Видалити вузол із вказаної позиції."},  {"2. Перезаписати дані у вказаній позиції."},  {"3. Перезаписати обране поле даних вузла."},  {"4. Записати дані у строку."},  {"5. Записати дані зі строки в вузол."},  {"6. Створити декілька вузлів із тексту."},  {"7. Створити текст із списку."},  {"8. Перезаписати вузол даними із іншого вузла."},  {"9. Знайти всі мови програмування створені до вказаного року."},  {"10. Записати список структур у текстовий файл."},  {"Вихід"}  };  **char** field[5][300] **=** {  {"fullname"},  {"date\_of\_occurrence"},  {"market\_share"},  {"major\_implementation"},  {"satisfaction\_index"},  };  **enum** {x **=** 23, y **=** 3};  **int** i **=** 0;  **int** j **=** 0;  **char** buffer[200];  **char** command\_1 **=** '1';  **char** command\_2 **=** '1';  **char** commandExit **=** '\0';  **int** choice **=** 0;  **int** choice\_1 **=** 0;  **int** choice\_2 **=** 0;  **int** position **=** 0;  **int** result;  **int** firstIndex;  **int** secondIndex;  List **\*** list;  conResize(37,120);  conHideCursor();  CUI\_mainMenu(menu, 1, 0, 0, **sizeof**(menu)**/sizeof**(menu[0]));  conMove(25, 41);  puts("Вузол містить 5 полів:\n\n\t\t\t\t\t 1. Назва мови програмування.\n\t\t\t\t\t\  2. Дата створення.\n\t\t\t\t\t 3. Доля ринку.\n\t\t\t\t\t 4. Основні реалізації.\n\t\t\t\t\t 5. Індекс задоволеності мовою \  програмування.\n\n\t\t\t\t\tПри створені структури дані записуються через пробіл!");  **while** (command\_1 **!=** '0') {  command\_1 **=** conGetChar();  CUI\_menuChoice(command\_1, **&**choice\_1, 1, menu, **sizeof**(menu)**/sizeof**(menu[0]));  **if** (command\_1 **==** '\n') {  **for** (i **=** 0; i **<** 13; i**++**) {  **for** (j **=** 0; j **<** 118; j**++**) {  conMove(23 **+** i, 2 **+** j);  putchar(' ');  }  }  **switch**(choice\_1) {  **case** 0**:**  list **=** List\_new();  command\_1 **=** '0';  **break**;  **case** 1**:**  conMove(x, y);  printf("Введіть назву файлу, з якого бажаєте зчитати масив структур: ");  conShowCursor();  fgets(buffer, 200, stdin);  conHideCursor();  buffer[strlen(buffer) **-** 1] **=** '\0';  list **=** Storage\_load(buffer);  conMove(x **+** 2, y);  **if** (list **!=** NULL) {  printf("Масив структур успішно зчитаний з файла %s.", buffer);  command\_1 **=** '0';  } **else** CUI\_printStr("Помилка файла зчитування або файл пустий!", x **+** 2, y, "FG\_YELLOW");  **break**;  }  }  }  CUI\_menuTasks(menu\_funct, 1, choice\_2, 1, **sizeof**(menu\_funct)**/sizeof**(menu\_funct[0]));  **if** (list **==** NULL) list **=** List\_new();  **while** (command\_2 **!=** '0') {  CUI\_listInfo(List\_count(list));  command\_2 **=** conGetChar();  CUI\_menuChoice(command\_2, **&**choice\_2, 10, menu\_funct, **sizeof**(menu\_funct)**/sizeof**(menu\_funct[0]));  **if** (command\_2 **==** '\n') {  **for** (i **=** 0; i **<** 14; i**++**) {  **for** (j **=** 0; j **<** 118; j**++**) {  conMove(23 **+** i, 2 **+** j);  putchar(' ');  }  }  **switch**(choice\_2) {  **case** 0**:**  conMove(x, y);  printf("Введіть індекс вузла, який бажаєте видалити: ");  conShowCursor();  fgets(buffer, 100, stdin);  conHideCursor();  **if** (checkInt(buffer) **==** true) {  position **=** atoi(buffer);  **if** (position **<** List\_count(list)) {  List\_removeAt(list, position);  CUI\_printStr("Вузол успішно видалено!", x **+** 2, y, "FG\_GREEN");  } **else** CUI\_printStr("Вузла з даним індексом не існує!", x **+** 2, y, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  **break**;  **case** 1**:**  conShowCursor();  conMove(x, y);  printf("Введіть індекс вузла, який бажаєте перезаписати: ");  fgets(buffer, 100, stdin);  **if** (checkInt(buffer) **==** true) {  position **=** atoi(buffer);  **if** (position **<** List\_count(list)) {  conMove(x **+** 1, y);  printf("Введіть дані через пробіл, які бажаєте занести в вузол: ");  fgets(buffer, 200, stdin);  buffer[strlen(buffer) **-** 1] **=** '\0';  **if** (List\_overwriteNode(list, position, buffer) **==** true) {  CUI\_printStr("Вузол був успішно презаписаний!", x **+** 3, y, "FG\_GREEN");  } **else** CUI\_printStr("Неправильні вхідні дані!", x **+** 3, y, "FG\_YELLOW");  } **else** CUI\_printStr("Вузла з даним індексом не існує!", x **+** 3, y, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  conHideCursor();  **break**;  **case** 2**:**  choice **=** 0;  commandExit **=** '\0';  conShowCursor();  conMove(x, y);  printf("Введіть індекс вузла: ");  fgets(buffer, 100, stdin);  **if** (checkInt(buffer) **==** true) {  position **=** atoi(buffer);  **if** (position **<** List\_count(list)) {  conMove(x **+** 1, y);  printf("Оберіть поле, яке бажаєте перезаписати(\"w\" or \"s\"): ");  conMove(24, 55);  printf("%s", field[0]);  **while** (commandExit **!=** '\n') {  commandExit **=** conGetChar();  **for** (**int** i **=** 0; i **<** 40; i**++**) {  conMove(24, 55 **+** i);  putchar(' ');  }  CUI\_menuChoice(commandExit, **&**choice, 4, field, **sizeof**(field)**/sizeof**(field[0]));  }  conMove(x **+** 2, y);  printf("Введіть нове значення вказаного поля: ");  fgets(buffer, 100, stdin);  **if** (ProgLang\_overwriteField(List\_elementAt(list, position), choice, buffer) **==** true) {  CUI\_printStr("Поле вузла перезаписане!", x **+** 3, y, "FG\_GREEN");  } **else** CUI\_printStr("Неправильні дані для запису в поле!", x **+** 3, y, "FG\_YELLOW");  } **else** CUI\_printStr("Вузла з даним індексом не існує!", x **+** 2, y, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  conHideCursor();  **break**;  **case** 3**:**  conMove(x, y);  printf("Введіть індекс вузла: ");  conShowCursor();  fgets(buffer, 200, stdin);  conHideCursor();  **if** (checkInt(buffer) **==** true **&&** atoi(buffer) **<** List\_count(list)) {  position **=** atoi(buffer);  CUI\_printNodeInMenu(List\_elementAt(list, position), x **+** 2, y);  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  **break**;  **case** 4**:**  conMove(x, y);  printf("Введіть позицію нового вузла в інтервалі [0, %i]: ", List\_count(list));  fgets(buffer, 100, stdin);  **if** (checkInt(buffer) **==** true **&&** atoi(buffer) **<=** List\_count(list)) {  position **=** atoi(buffer);  conMove(x **+** 2, y);  printf("Введіть дані через пробіл, які бажаєте занести в новий вузол: ");  conShowCursor();  conMove(x **+** 4, y);  fgets(buffer, 200, stdin);  conHideCursor();  buffer[strlen(buffer) **-** 1] **=** '\0';  **if** (List\_strToNode(list, position, buffer) **==** true) {  CUI\_printStr("Новий вузол успішно створений!", x **+** 4, y **+** 30, "FG\_GREEN");  } **else** CUI\_printStr("Неправильні вхідні дані!", x **+** 4, y **+** 30, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказано позицію вузла!", x **+** 2, y, "FG\_YELLOW");  **break**;  **case** 5**:**  commandExit **=** '1';  conMove(x, y);  conShowCursor();  printf("Введіть дані в список через пробіл: ");  **while**(commandExit **!=** 'q') {  conMove(36, 3);  **for** (**int** j **=** 0; j **<** 100; j**++**) {  conMove(35 , 2 **+** j);  putchar(' ');  conMove(x **+** 2 , 2 **+** j);  putchar(' ');  }  conMove(x **+** 2, y);  fgets(buffer, 100, stdin);  **if** (List\_strToNode(list, List\_count(list), buffer) **==** false) CUI\_printStr("Невірні вхідні дані!", x **+** 2, y **+** 30, "FG\_YELLOW");  **else** CUI\_printStr("Новий вузол створений!", x **+** 2, y **+** 30, "FG\_GREEN");  CUI\_printStr("Натисніть \"q\", щоб припинити створенняя структур чи Enter, щоб продовжити", 35, 23, "FG\_GREEN");  commandExit **=** conGetChar();  }  **for** (**int** i **=** 0; i **<** 100; i**++**) {  conMove(35, 3 **+** i);  putchar(' ');  }  CUI\_printStr("Текст успішно занесений до списку", 35, 38, "FG\_GREEN");  conHideCursor();  **break**;  **case** 6**:**  conMove(1, 1);  **if** (List\_count(list) **!=** 0) {  CUI\_controlPrint(list);  } **else** {  conClear();  CUI\_printStr("Список порожній!", 1, 54, "FG\_YELLOW");  CUI\_printStr("Натисніть \"Enter\" щоб продовжити", 37, 48, "FG\_GREEN");  conGetChar();  }  CUI\_menuTasks(menu\_funct, 1, choice\_2, 0, **sizeof**(menu\_funct)**/sizeof**(menu\_funct[0]));  **break**;  **case** 7**:**  conMove(x, y);  conShowCursor();  printf("Введіть індекс вузла, яку бажаєте перезаписати: ");  fgets(buffer, 200, stdin);  **if** (checkInt(buffer) **==** true) {  firstIndex **=** atoi(buffer);  conMove(x **+** 1, y);  printf("Введіть індекс вузла з якої виконується перезапис: ");  fgets(buffer, 200, stdin);  **if** (checkInt(buffer) **==** true) {  secondIndex **=** atoi(buffer);  conMove(x **+** 2, y);  result **=** List\_overwriteNodeFromNode(list, firstIndex, secondIndex);  **if** (result **==** 0) {  CUI\_printStr("Перезапис виконано успішно!", x **+** 2, y, "FG\_GREEN");  } **else**  **if** (result **==** 1) {  CUI\_printStr("Вузла, який ви бажаєте перезаписати не існує!", x **+** 2, y, "FG\_YELLOW");  } **else** CUI\_printStr("Вузла з якого виконується перезапис не існує!", x **+** 2, y, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  } **else** CUI\_printStr("Неправильно вказаний індекс!", x **+** 2, y, "FG\_YELLOW");  conHideCursor();  **break**;  **case** 8**:**  conMove(x, y);  printf("Введіть рік: ");  conShowCursor();  fgets(buffer, 200, stdin);  conHideCursor();  **if** (checkInt(buffer) **==** true) {  position **=** atoi(buffer);  conClear();  conMove(1, 1);  CUI\_findLanguages(list, position);  CUI\_menuTasks(menu\_funct, 1, 8, 0, **sizeof**(menu\_funct)**/sizeof**(menu\_funct[0]));  } **else** CUI\_printStr("Неправильно вказаний рік!", x **+** 2, y, "FG\_YELLOW");  **break**;  **case** 9**:**  conMove(x, y);  printf("Введіть назву файлу, в який бажаєте записати список: ");  conShowCursor();  fgets(buffer, 200, stdin);  buffer[strlen(buffer) **-** 1] **=** '\0';  **if** (Storage\_save(buffer, list) **==** true) {  conMove(x **+** 2, y);  conSetAttr(FG\_GREEN);  printf("Масив структур записаний у файл \"%s\".", buffer);  conReset();  } **else** CUI\_printStr("Помилка файла запису", x **+** 2, y, "FG\_YELLOW");  conHideCursor();  **break**;  **case** 10**:**  command\_2 **=** '0';  **break**;  }  }  }  List\_clear(list);  List\_free(**&**list);  CUI\_endProg();  **return** EXIT\_SUCCESS;  } |

**Приклади результатів**

**Головне меню**

Гортання пунктів в меню здійснюється кнопками “w” та “s”. Щоб перейти до завдання необхідно нажати Enter.

Для оптимізації програми головне меню і меню кожного завдання будується один раз при переході між завданнями та головним меню. У інших випадках просто перемальовується певні строки зображення. Таким чином при гортанні в меню всі зображення, крім строк залишаються статичними.

**Завдання 1**

Користувач гортає пункти меню та виконує бажані дії зі списком структур. Всі вхідні дані перевіряються на валідність.

**Завдання 2**

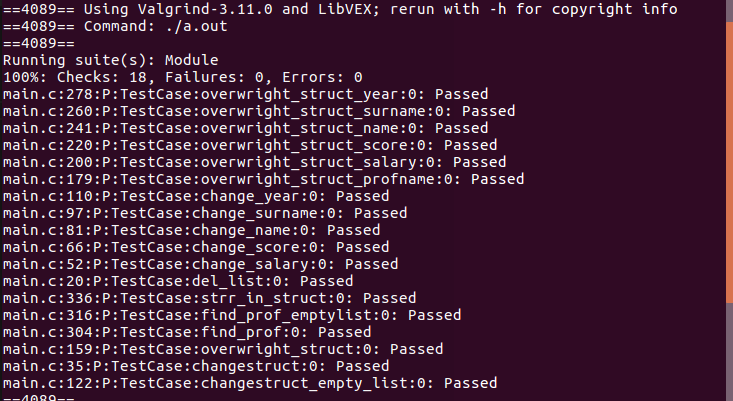


Було створено чотири модулі:

1. Список даних
2. Модуль конвертації даних
3. Сховище даних (файлова система)
4. Консольний інтерфейс користувача (CUI)

**Завдання 3**

Модульні тести розташовані в папці tests. Компілювати файл за допомогою опцій: make(у файлi Makefile уже прописані необхідні команди).



**Висновок**

Виконавши дану лабораторну роботу були отримані навички роботи зі структурами, файлами, а також вдосконалені та закріплені вміння роботи з вказівниками.

Було застосовано на практиці динамічне видалення пам’яті та освоєно мультифайлове програмування, роботу з файлами .c і .h.

Виконавши дану лабораторну роботу були отримані навики роботи зі статичними масивами різних типів даних мови програмування С.

Було створено консольну програму, яка поєднує в собі всі завдання. Для зручності роботи користувача було створено меню, а у станах, де необхідно користувачу виконати певні дії (наприклад: «Змінити значення елементу за вказаними індексами на задане») всі вхідні данні перевіряються на валідність. У разі некоректних вхідних даних виводиться повідомлення про помилку.

Були оповановані навички створення модулів та їх перевірка за допомогою модульних тестів.

Компіляція всього коду відбувалася за допомогою Malefile.

