

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 №** 5

з дисципліни “ Основи програмування ”

Тема “**Бібліотеки і формати даних**”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-62  Лук’янець Михайло Олександрович  (*прізвище, ім’я, по батькові*)  варіант №15 |  | 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)** | |  |  |  | | Сумарний бал:   |  | | --- | |  | |

Київ 2017

**Мета роботи**

Навчитися використовувати та створювати статичні бібліотеки в ОС Linux. Навчитися працювати із текстовими форматами обміну даних XML та JSON.

#### **Завдання №1. Формати даних**

Встановити та налаштувати бібліотеки для роботи із форматами XML та JSON.

Підключити бібліотеки до проекту лабораторної роботи №4. Додати до модуля сховища даних можливість завантаження і збереження даних у файл у форматах XML та JSON (без модифікації коду інших модулів).

*/\* storage.h \*/*  
...  
**void** **Storage\_writeAsXml**(**const** **char** \* filePath, List \* list);  
List \* **Storage\_readAsXml**(**const** **char** \* filePath);  
  
**void** **Storage\_writeAsJson**(**const** **char** \* filePath, List \* list);  
List \* **Storage\_readAsJson**(**const** **char** \* filePath);  
...

Після цього додати у модулі CUI можливість автоматичного визначення формату даних у файлі, що відкривається користувачем (на основі його файлового розширення) із використанням відповідної функції конвертації для зчитування даних.

Аналогічно зберігати дані у відповідному форматі, коли користувач захоче зберегти дані у файл із розширенням .xml, .json чи .txt.

#### **Завдання №2. Бібліотеки**

Розділити код з лабораторної роботи №4 на бібліотечний і програмний (користувацький). У бібліотечний код винести всі модулі програми і створити на його основі статичну бібліотеку, яку підключати у основний проект. Описати конфігурацію CMake для компіляції і встановлення статичної бібліотеки.

**Тексти коду програм**

|  |
| --- |
| main.c |
| #include <progbase.h>  #include <string.h>  #include <ctype.h>  #include "interface.h"  #include "dino.h"  #include "totest.h"  #include "saves.h"  #include "list.h"  **int** main(**int** argc, **char** \* argv[]) {  **char** key = '\0';  **if**(argc > 1 && (!strcmp(argv[1], "-test"))){  test();  }  **const struct** conShift MAIN\_SHIFT = { 0, -10 + MAX\_SIZE};  **const char** \*START\_TEXT = "Hello User! Welcome to jurasic's park pre-alpha version 0.0.2.255\n" \  "You can choose what to do via pressing needed button. Press h for info\nPress q to quit program\n\0";  **const char** \*START\_INFO = "Press 1 to rewrite certain field\n"  "Press 2 to rewrite one dino\n"  "Press 3 to delete needed position\n"  "Press 4 to save data\n"  "Press 5 to copy data from one dino to another\n"  "Press 6 to find all dinos with mass less than X\n"  "Press 7 to add dino if possible\n"  "Press 8 to fill with some random data ALL fields(for tests only!)\n"  "\n"  "Press h to return\n\0";  redraw(MAIN\_SHIFT.cols , MAIN\_SHIFT.rows );  **int** flagMode = start("Press 1 for new array. Press 2 for existing array");  **char** readFileName[100] = "";  **if**(flagMode == 2){  **char** \* readFileTemp = getStringInter("read file name");  strcpy( readFileName , readFileTemp);  free(readFileTemp);  }  **struct** List \* list = readFile(readFileName);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  **while**(key != 'q'){  key = conGetChar();  **switch** (key) {  **case** '1':  ;  **char** \* number = getStringInter("number of a dino");  **char** \* toChange = getStringInter("new data");  **char** \* field = getStringInter("field to rewrite");  **if**(atoi(number) >= MAX\_SIZE || !isdigit(number[0])){  free(number);  free(toChange);  free(field);  **continue**;  }  changeField(List\_elementAt(list, atoi(number)), field, toChange, list);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  printResult("field was changed");  free(number);  free(toChange);  free(field);  **break**;  **case** '2':  ;  **char** \* numberOfDino = getStringInter("number of a dino");  **char** \* name = getStringInter("name");  **char** \* age = getStringInter("age");  **char** \* friend = getStringInter("number of friend");  **char** \* mass = getStringInter("mass");  **if**(atoi(numberOfDino) >= MAX\_SIZE || !isdigit(numberOfDino[0])){  printResult("Not correct number");  free(numberOfDino);  free(name);  free(age);  free(friend);  free(mass);  **continue**;  }  changeField(List\_elementAt(list, atoi(numberOfDino)), "name", name, list);  changeField(List\_elementAt(list, atoi(numberOfDino)), "age", age, list);  changeField(List\_elementAt(list, atoi(numberOfDino)), "friend", friend, list);  changeField(List\_elementAt(list, atoi(numberOfDino)), "mass", mass, list);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  printResult("Data was changed");  free(numberOfDino);  free(name);  free(age);  free(friend);  free(mass);  **break**;  **case** '3':  ;  **char** \* numberToDelete = getStringInter("number of a dino to delete");  **if**(!isdigit(numberToDelete[0])){  printResult("Not correct number");  free(numberToDelete);  **continue**;  }  delete(list, atoi(numberToDelete));  free(numberToDelete);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  printResult("Dino was deleted. Or not");  **break**;  **case** '4':  ;  **char** \* saveFileName = getStringInter("save file name");  **if**(!saveFile(list, saveFileName)){  printResult("Data saved");  } **else** {  printResult("Save was absolutely sabotaged");  }  free(saveFileName);  **break**;  **case** '5':  ;  **char** \* first = getStringInter("dino to rewrite");  **char** \* second = getStringInter("dino to copy");  **if**(!isdigit(first[0]) || !isdigit(second[0]) ||  atoi(first) >= MAX\_SIZE || atoi(second) >= MAX\_SIZE){  printResult("Not correct numbers");  free(first);  free(second);  **continue**;  }  rewrite(list, atoi(first), atoi(second));  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  printResult("Data copied");  free(first);  free(second);  **break**;  **case** '6':  ;  **char** \* massLower = getStringInter("mass");  **char** \* result = findByMass(list, atof(massLower));  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  **if**(strlen(result) == 0){  printResult("Not found");  } **else** {  printResult(result);  }  free(result);  free(massLower);  **break**;  **case** '7':  ;  **char** \* data = getStringInter("data in format name age mass number of friend");  cleanCanvas();  **if**(!writeFromStr(list, data)){  printResult("Added successfully");  } **else** {  printResult("Added not successfully");  }  printText(START\_TEXT);  printStructArr(list);  free(data);  **break**;  **case** '8':  fillRandList(list);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  printResult("Dinos were randomized");  **break**;  **case** 'h':  cleanCanvas();  printInfoText(START\_INFO);  cleanCanvas();  printText(START\_TEXT);  printStructArr(list);  **break**;  **default**: ;  }  }  List\_free(&list);  exitInter();  **return** 0;  } |

|  |
| --- |
| dino.h |
| //  // Created by tgifr on 03.12.16.  //  #include <pbconsole.h>  #include <progbase.h>  #include <string.h>  #include <time.h>  #include "list.h"  #ifndef LAB3\_DYNO\_H  #define LAB3\_DYNO\_H  #define MAX\_SIZE 10  #define NAMES\_SIZE 10  **const char** \* DIN0\_NAMES[10] ;  **struct** dino {  **char** name[100];  **int** age;  **double** mass;  **struct** dino \* friend;  **int** number;  **struct** dino \* next;  };  **int** writeFromStr(**struct** List \*list, **char** \*str);  **char** \* writeInStr(**struct** dino \* dyno);  **struct** List \* writeFromText(**char** \*text);  **char** \*writeInText(**struct** List \*list);  **void** delete(**struct** List \*list, **int** toDelete);  **void** rewrite(**struct** List \*list, **int** toRewrite, **int** toCopy);  **void** changeField(**struct** dino \*dyno, **char** \*field, **char** \*toChange, **struct** List \*list);  **char**\* findByMass(**struct** List \*list, **double** mass);  **void** initializeList(**struct** List \*list);  **void** fillRandList(**struct** List \*list);  #endif //LAB3\_DYNO\_H |

|  |
| --- |
| dino.c |
| //  // Created by tgifr on 03.12.16.  //  #include <ctype.h>  #include "dino.h"  #include "list.h"  **const char** \* DIN0\_NAMES[10] = { "Kvadrat", "Jurasic", "Star", "Katzchen", "Jet",  "Half", "life", "3", "Night\_Fury", "Moonshine"};  **int** writeFromStr(**struct** List \*list, **char** \*str){  **int** i = 0;  **char** test[100] = "";  strcpy(test, str);  **char** \* temp = strtok(test, " ");  **struct** dino \* tempDino = List\_elementAt(list, i);  **while**(strcmp(tempDino->name, "") || tempDino->age != 0 || tempDino->mass != 0.0){  i++;  tempDino = List\_elementAt(list, i);  **if**(i == MAX\_SIZE){  **return** EXIT\_FAILURE;  }  }  **if**(temp != NULL && isalpha(temp[0])){  strcpy(tempDino->name, temp);  temp = strtok(NULL, " ");  }  **if**(temp != NULL && isdigit(temp[0])){  tempDino->age = atoi(temp);  temp = strtok(NULL, " ");  }  **if**(temp != NULL && isdigit(temp[0])){  tempDino->mass = atof(temp);  temp = strtok(NULL, " ");  }  **if**(temp != NULL && atoi(temp) < MAX\_SIZE && atoi(temp) > -1 && isdigit(temp[0])){  tempDino->friend = List\_elementAt(list,atoi(temp));  }  **return** EXIT\_SUCCESS;  }  **char** \* writeInStr(**struct** dino \* dyno){  **char** \*result = malloc(200 \* **sizeof**(**char**));  **if**(result == NULL){  **return** 0;  }  sprintf(result, "%s %i %.3f %i\n", dyno->name, dyno->age, dyno->mass, dyno->friend->number);  **return** result;  }  **struct** List \* writeFromText(**char** \*text){  **char** temp[200] = "";  **struct** List \* list = createDinoList();  **if**(list == NULL){  **return** NULL;  }  initializeList(list);  **char** toCopy[2] = "";  toCopy[1] = '\0';  **for** (**int** i = 0; text[i] != '\0'; ++i) {  toCopy[0] = text[i];  strcat(temp, toCopy);  **if**(text[i] == '\n'){  temp[i] = '\0';  **if** (!writeFromStr(list, temp)) {  **break**;  }  strcpy(temp, "");  }  }  **return** list;  }  **char** \*writeInText(**struct** List \*list){  **char** \* temp;  **char** result[1000] = "";  **for** (**int** i = 0; i < MAX\_SIZE; ++i) {  temp = writeInStr(List\_elementAt(list, i));  strcat(result, temp);  free(temp);  }  temp = malloc((strlen(result) + 5 ) \* **sizeof**(**char**));  **if**(temp == NULL){  **return** 0;  }  strcpy(temp, result);  **return** temp;  }  **void** delete(**struct** List \*list, **int** toDelete){  **if**(toDelete >= MAX\_SIZE || toDelete < 0){  **return**;  }  **struct** dino \* temp = List\_elementAt(list, toDelete);  strcpy(temp->name, "");  temp->age = 0;  temp->mass = 0.0;  temp->friend = List\_elementAt(list, 0);  }  **void** rewrite(**struct** List \*list, **int** toRewrite, **int** toCopy){  **if**(toRewrite >= MAX\_SIZE || toCopy >= MAX\_SIZE || toCopy < 0 || toRewrite < 0){  **return**;  }  **struct** dino \* toRewriteD = List\_elementAt(list, toRewrite);  **struct** dino \* toCopyD = List\_elementAt(list, toCopy);  strcpy(toRewriteD->name, toCopyD->name);  toRewriteD->age = toCopyD->age;  toRewriteD->mass = toCopyD->mass;  toRewriteD->friend = toCopyD->friend;  }  **void** changeField(**struct** dino \*dyno, **char** \*field, **char** \*toChange, **struct** List \*list) {  **if**(!strcmp("name", field)){  strcpy(dyno->name, toChange);  } **else if**(!strcmp("age", field)){  dyno->age = atoi(toChange);  } **else if**(!strcmp("mass", field)){  dyno->mass = atof(toChange);  } **else if**(!strcmp("friend", field) && atoi(toChange) < MAX\_SIZE){  dyno->friend = List\_elementAt(list,atoi(toChange));  }  }  **char**\* findByMass(**struct** List \*list, **double** mass){  **char** temp[300] = "";  **for** (**int** i = 0; i < MAX\_SIZE; ++i) {  **struct** dino \* tempD = List\_elementAt(list, i);  **if**(tempD->mass < mass){  strcat(temp, tempD->name);  strcat(temp, " ");  }  }  **char** \* result = malloc((strlen(temp) + 5)\* **sizeof**(**char**));  **if**(result == NULL){  **return** "";  }  strcpy(result, temp);  **return** result;  }  **void** initializeList(**struct** List \*list){  **for**(**int** i = 0; i < MAX\_SIZE; i ++){  **struct** dino \* temp = List\_elementAt(list, i);  strcpy(temp->name ,"");  temp->number = i;  temp->friend = List\_elementAt(list, 0);  temp->age = 0;  temp->mass = 0.0;  }  }  **void** fillRandList(**struct** List \*list){  srand(time(0));  **for**(**int** i = 0; i < MAX\_SIZE; i ++){  **struct** dino \* temp = List\_elementAt(list, i);  strcpy(temp->name, DIN0\_NAMES[rand() % NAMES\_SIZE]);  temp->friend = List\_elementAt(list, rand() % MAX\_SIZE);  temp->age = (rand() % 1000) + 1;  temp->mass = 50.0 / ((rand() % 50) + 1) ;  }  } |

|  |
| --- |
| saves.h |
| //  // Created by tgifr on 03.12.16.  //  #include "dino.h"  #ifndef LAB3\_SAVES\_H  #define LAB3\_SAVES\_H  **int** Storage\_writeAsXml(**const char** \*filePath, List \*list);  List \* Storage\_readAsXml(**const char** \* filePath);  **int** Storage\_writeAsJson(**const char** \*filePath, List \*list);  List \* Storage\_readAsJson(**const char** \* filePath);  **int** Storage\_write(**const char** \*filePath, List \*list);  List \* Storage\_read(**const char** \* filePath);  **struct** List \* Storage\_readAsTxt(**const char** \*readFileName);  **int** Storage\_writeAsTxt(**struct** List \*list, **const char** \*saveFileName);  #endif //LAB3\_SAVES\_H |

|  |
| --- |
| saves.c |
| **//**  **// Created by tgifr on 03.12.16.**  **//**  **#include <stdio.h>**  **#include "../include/dino.h"**  **#include "../include/list.h"**  **#include <libxml/parser.h>**  **#include "../include/list.h"**  **#include <jansson.h>**  **#include "../include/saves.h"**  **enum{**  ***TXT*, *XML*, *JSON*, *NO\_EXTENSION* = -1**  **};**  **int readAllText(const char \* filename, char \* text) {**  **char line[100];**  **FILE \* fr = fopen(filename, "r");**  **if (fr == NULL) return 0;**  **while(fgets(line, 100, fr)) {**  **strcat(text, line);**  **}**  **fclose(fr);**  **return 1;**  **}**  **int analyzeExtension(const char \* str){**  **const char \* dot = str;**  **while(\*dot != '.' && \*dot != '\0'){**  **dot++;**  **}**  **if(\*dot != '.'){**  **return *NO\_EXTENSION*;**  **}**  **dot ++;**  **if(strcmp(dot, "xml") == 0)return *XML*;**  **else if(strcmp(dot, "json") == 0) return *JSON*;**  **else if(strcmp(dot, "txt") == 0) return *TXT*;**  **else return *NO\_EXTENSION*;**  **}**  **int Storage\_write(const char \*filePath, List \*list){**  **switch (analyzeExtension(filePath)){**  **case *NO\_EXTENSION*:**  **return EXIT\_FAILURE;**  **case *XML*:**  **return Storage\_writeAsXml(filePath, list);**  **case *JSON*:**  **return Storage\_writeAsJson(filePath, list);**  **case *TXT*:**  **return Storage\_writeAsTxt(list, filePath);**  **default:**  **return EXIT\_FAILURE;**  **}**  **}**  **List \* Storage\_read(const char \* filePath){**  **switch (analyzeExtension(filePath)){**  **case *XML*:**  **return Storage\_readAsXml(filePath);**  **case *JSON*:**  **return Storage\_readAsJson(filePath);**  **default:**  **return Storage\_readAsTxt(filePath);**  **}**  **}**  **int Storage\_writeAsXml(const char \*filePath, List \*list){**  **FILE \* write;**  **if((write = fopen(filePath, "w")) == NULL){**  **return EXIT\_FAILURE;**  **}**  **xmlDoc \* doc = NULL;**  **xmlNode \* rootNode = NULL;**  **xmlNode \* dinoNode = NULL;**  **char strBuf[4000];**  **doc = xmlNewDoc(BAD\_CAST "1.0");**  **// create xml tree**  **// create one root element**  **rootNode = xmlNewNode(NULL, BAD\_CAST "dinos");**  **xmlDocSetRootElement(doc, rootNode);**  **// student child**  **for (int j = 0; j < MAX\_SIZE; j ++) {**  **dino \* dino = List\_elementAt(list, j);**  **dinoNode = xmlNewChild(rootNode, NULL, BAD\_CAST "dino", NULL);**  **xmlNewChild(dinoNode, NULL, BAD\_CAST "name", BAD\_CAST dino->name);**  **sprintf(strBuf, "%i", dino->age); // copy number to string**  **xmlNewChild(dinoNode, NULL, BAD\_CAST "age", BAD\_CAST strBuf);**  **sprintf(strBuf, "%f", dino->mass); // copy number to string**  **xmlNewChild(dinoNode, NULL, BAD\_CAST "mass", BAD\_CAST strBuf);**  **sprintf(strBuf, "%i", dino->friend->number); // copy number to string**  **xmlNewChild(dinoNode, NULL, BAD\_CAST "friend", BAD\_CAST strBuf);**  **}**  **// copy xml contents to char buffer**  **xmlBuffer \* bufferPtr = xmlBufferCreate();**  **xmlNodeDump(bufferPtr, NULL, (xmlNode \*)doc, 0, 1);**  **fprintf(write, "%s", (const char \*)bufferPtr->content);**  **xmlBufferFree(bufferPtr);**  **xmlFreeDoc(doc);**  **fclose(write);**  **return EXIT\_SUCCESS;**  **}**  **List \* Storage\_readAsXml(const char \* filePath){**  **char xmlStr[3000];**  **if(readAllText(filePath, xmlStr) == 0);**  **List \* list = createDinoList();**  **initializeList(list);**  **xmlDoc \* xDoc = xmlReadMemory(xmlStr, (int)strlen(xmlStr), NULL, NULL, 0);**  **if (NULL == xDoc) {**  **printf("Error parsing xml");**  **return list;**  **}**  **xmlNode \* xRootEl = xmlDocGetRootElement(xDoc);**  **for (xmlNode \* xCur = xRootEl->children; NULL != xCur ; xCur = xCur->next) {**  **if (*XML\_ELEMENT\_NODE* == xCur->type) {**  **dino \* cur = dino\_new();**  **for (xmlNode \* xJ = xCur->children; NULL != xJ ; xJ = xJ->next) {**  **if (*XML\_ELEMENT\_NODE* == xJ->type) {**  **char \* content = (char \*)xmlNodeGetContent(xJ);**  **if (xmlStrcmp(xJ->name, BAD\_CAST "friend") == 0) {**  **cur->friend = List\_elementAt(list, atoi(content));**  **} else if (xmlStrcmp(xJ->name, BAD\_CAST "name") == 0) {**  **strcpy(cur->name, content);**  **} else if (xmlStrcmp(xJ->name, BAD\_CAST "age") == 0) {**  **cur->age = atoi( content);**  **} else if (xmlStrcmp(xJ->name, BAD\_CAST "mass") == 0) {**  **cur->mass = (float) atof( content);**  **} else {**  **printf("ERROR WHILE READING");**  **return list;**  **}**  **xmlFree(content);**  **}**  **}**  **char \* temp = writeInStr(cur);**  **free(cur);**  **writeFromStr(list, temp);**  **free(temp);**  **}**  **}**  **xmlFreeDoc(xDoc);**  **return list;**  **}**  **int Storage\_writeAsJson(const char \*filePath, List \*list){**  **FILE \* write;**  **if((write = fopen(filePath, "w")) == NULL){**  **return EXIT\_FAILURE;**  **}**  **if(filePath == NULL || list == NULL) return EXIT\_FAILURE;**  **// object construction**  **json\_t \* arr = json\_array();**  **for (int i = 0; i < List\_count(list); i++) {**  **dino \* cur = List\_elementAt(list, i);**  **json\_t \* curJ = json\_object();**  **json\_object\_set\_new(curJ, "name", json\_string(cur->name));**  **json\_object\_set\_new(curJ, "age", json\_integer(cur->age));**  **json\_object\_set\_new(curJ, "score", json\_real(cur->mass));**  **json\_object\_set\_new(curJ, "friend", json\_integer(cur->friend->number));**  **json\_array\_append(arr, curJ);**  **}**  **// create JSON document string**  **char \* jsonString = json\_dumps(arr, JSON\_INDENT(2));**  **fprintf(write, "%s", jsonString);**  **free(jsonString);**  **// decrease reference count (free's memory when count is 0')**  **json\_decref(arr);**  **fclose(write);**  **return EXIT\_SUCCESS;**  **}**  **List \* Storage\_readAsJson(const char \* filePath){**  **char jsonStr[3000];**  **if(readAllText(filePath, jsonStr) == 0);**  **List \* list = createDinoList();**  **initializeList(list);**  **json\_error\_t err;**  **json\_t \* jsonArr = json\_loads(jsonStr, 0, &err);**  **int index = 0;**  **json\_t \* value = NULL;**  **json\_array\_foreach(jsonArr, index, value) {**  **dino \* cur = dino\_new();**  **strcpy(cur->name, (char \*) json\_string\_value(json\_object\_get(value, "name")));**  **cur->mass = (float) json\_real\_value(json\_object\_get(value, "mass"));**  **cur->age = (int) json\_integer\_value(json\_object\_get(value, "age"));**  **cur->friend = List\_elementAt(list, (int) json\_integer\_value(json\_object\_get(value, "friend")));**  **char \* temp = writeInStr(cur);**  **writeFromStr(list, temp);**  **free(temp);**  **free(cur);**  **}**  **json\_decref(jsonArr);**  **return list;**  **}**  **struct List \* Storage\_readAsTxt(const char \*readFileName){**  **FILE \* read;**  **char temp[150] = "";**  **struct List \* list = createDinoList();**  **initializeList(list);**  **if((read = fopen(readFileName, "r")) == NULL){**  **return list;**  **}**  **while(!feof(read)){**  **fgets(temp, 150, read);**  **if(writeFromStr(list, temp)){**  **break;**  **}**  **}**  **fclose(read);**  **return list;**  **}**  **int Storage\_writeAsTxt(struct List \*list, const char \*saveFileName){**  **FILE \* save;**  **if((save = fopen(saveFileName, "w")) == NULL){**  **return EXIT\_FAILURE;**  **}**  **char \* toWrite = writeInText(list);**  **fprintf(save, "%s", toWrite);**  **free(toWrite);**  **fclose(save);**  **return EXIT\_SUCCESS;**  **}** |

|  |
| --- |
| **interface.h** |
| **//**  **// Created by tgifr on 03.12.16.**  **//**  **#include <pbconsole.h>**  **#include <progbase.h>**  **#include <string.h>**  **#include "dino.h"**  **#ifndef INTERFACE\_H**  **#define INTERFACE\_H**  **void redraw(int cols, int rows);**  **void clearAnim();**  **void printText(const char \*TEXT);**  **void printInfoText(const char \*HELP);**  **struct conShift{**  **short cols;**  **short rows;**  **};**  **void conDefPos();**  **void cleanInp();**  **void cleanCanvas();**  **char \* getStringInter(char \* whatGet);**  **void exitInter();**  **void printStructArr(struct List ( \*list));**  **int start(const char \*START\_TEXT);**  **void printResult(char \* result);**  **#endif //LAB3\_INTERFACE\_H** |

|  |
| --- |
| **interface.c** |
| **//**  **// Created by tgifr on 03.12.16.**  **//**  **#include "interface.h"**  **#include "list.h"**  **unsigned long int MILLIS\_WAIT = 30;**  **const struct consize DEF\_SIZE = {87, 25};**  **const char \* EXIT\_TEXT = " \_\_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \n"**  **" / \_\_\_\_|/ \_\_ \\ / \_\_ \\| \_\_ \\| \_ \\ \\ / / \_\_\_\_|\n"**  **" | | \_\_| | | | | | | | | | |\_) \\ \\\_/ /| |\_\_ \n"**  **" | | |\_ | | | | | | | | | | \_ < \\ / | \_\_| \n"**  **" | |\_\_| | |\_\_| | |\_\_| | |\_\_| | |\_) | | | | |\_\_\_\_ \n"**  **" \\\_\_\_\_\_|\\\_\_\_\_/ \\\_\_\_\_/|\_\_\_\_\_/|\_\_\_\_/ |\_| |\_\_\_\_\_\_|\n";**  **void redraw(int cols, int rows){**  **int i = 0;**  **conResize(DEF\_SIZE.rows + (unsigned short)rows, DEF\_SIZE.cols + (unsigned short)cols);**  **conClear();**  **const struct consize actSize = { DEF\_SIZE.cols + (unsigned short)cols, DEF\_SIZE.rows + (unsigned short)rows};**  **conSetAttr(*BG\_WHITE*);**  **conSetAttr(*FG\_INTENSITY\_BLUE*);**  **for(i = 1; i <= actSize.cols; i ++){**  **conMove(1, i );**  **printf("=");**  **conMove(actSize.rows - 5, i);**  **printf("=");**  **conMove(actSize.rows, i );**  **printf("=");**  **fflush(stdout);**  **sleepMillis(MILLIS\_WAIT/3);**  **}**  **for(i = 1; i <= actSize.rows; i ++){**  **conMove(i , 1);**  **printf("|");**  **conMove(i , actSize.cols);**  **printf("|");**  **fflush(stdout);**  **sleepMillis(MILLIS\_WAIT);**  **}**  **conReset();**  **conMove( actSize.rows - 4, 3);**  **printf("Print here: ");**  **}**  **void clearAnim(){**  **int i = 0;**  **int j = 0;**  **const struct consize size = conGetSize();**  **for(i = 1; i <= size.cols; i ++){**  **conSetAttr(*BG\_INTENSITY\_BLACK*);**  **for(j = 1; j <= size.rows; j ++){**  **conMove(j, i);**  **printf(" ");**  **conMove(j, i + 1);**  **printf(" ");**  **}**  **conSetAttr(*BG\_DEFAULT*);**  **if (i != 1) {**  **for(j = 1; j <= size.rows; j ++){**  **conMove(j, i - 1);**  **printf(" ");**  **}**  **}**  **if (i == size.cols) {**  **for(j = 1; j <= size.rows; j ++){**  **conMove(j, i);**  **printf(" ");**  **}**  **}**  **fflush(stdout);**  **sleepMillis(MILLIS\_WAIT);**  **}**  **conReset();**  **conDefPos();**  **}**  **void printText(const char \*TEXT){**  **int i = 0, yShift = 0;**  **conMove(3, 3);**  **for(i = 0; i < strlen(TEXT); i ++){**  **if(TEXT[i] == '\n'){**  **yShift++;**  **conMove(3 + yShift, 3);**  **} else printf("%c", TEXT[i]);**  **}**  **conDefPos();**  **}**  **void printInfoText(const char \*HELP){**  **char key = '\0';**  **cleanCanvas();**  **printText(HELP);**  **while (key != 'h'){**  **key = conGetChar();**  **}**  **cleanInp();**  **}**  **void conDefPos(){**  **struct consize actSize = conGetSize();**  **conMove(actSize.rows - 4, 3 + sizeof("Print here: "));**  **}**  **void cleanInp(){**  **int i = 0;**  **struct consize actSize = conGetSize();**  **for(i = sizeof("Print here: ") + 3; i < actSize.cols - 2; i++){**  **conMove(actSize.rows - 4, i);**  **printf(" ");**  **conMove(actSize.rows - 3, i - sizeof("Print here: "));**  **printf(" ");**  **}**  **conMove(actSize.rows - 6, 3 );**  **printf(" ");**  **conDefPos();**  **}**  **void cleanCanvas(){**  **int i = 0;**  **int j = 0;**  **conReset();**  **struct consize actSize = conGetSize();**  **for(j = 2; j < actSize.rows - 5; j ++){**  **for(i = 2; i < actSize.cols; i++){**  **conMove(j, i);**  **printf(" ");**  **}**  **}**  **conDefPos();**  **}**  **char \* getStringInter(char \* whatGet){**  **struct consize actSize = conGetSize();**  **conMove(actSize.rows - 6, 3 );**  **printf("Please, enter %s:", whatGet);**  **conDefPos();**  **char \* result = getString();**  **cleanInp();**  **return result;**  **}**  **void exitInter(){**  **clearAnim();**  **printText(EXIT\_TEXT);**  **sleepMillis(MILLIS\_WAIT \* 100);**  **conClear();**  **}**  **void printStructArr(struct List \*list){**  **for(int i = 0; i < MAX\_SIZE ; i ++){**  **conMove(6 + i, 3);**  **struct dino \* temp = List\_elementAt(list, i);**  **if(strcmp(temp->name, "") || temp->age != 0 || temp->mass != 0.0) {**  **printf("Dino number[%i]. Name: %s. Age: %i. Mass: %.3f. Friend: %s", i, temp->name, temp->age,**  **temp->mass, temp->friend->name);**  **}**  **}**  **conDefPos();**  **}**  **int start(const char \*START\_TEXT){**  **conMove(3,3);**  **char key = '\0';**  **printf("%s", START\_TEXT);**  **while(key != '1' && key != '2'){**  **key = conGetChar();**  **}**  **return key - '0';**  **}**  **void printResult(char \* result){**  **struct consize actSize = conGetSize();**  **conMove(actSize.rows - 6 , 3);**  **printf("%s", result);**  **conDefPos();**  **}** |

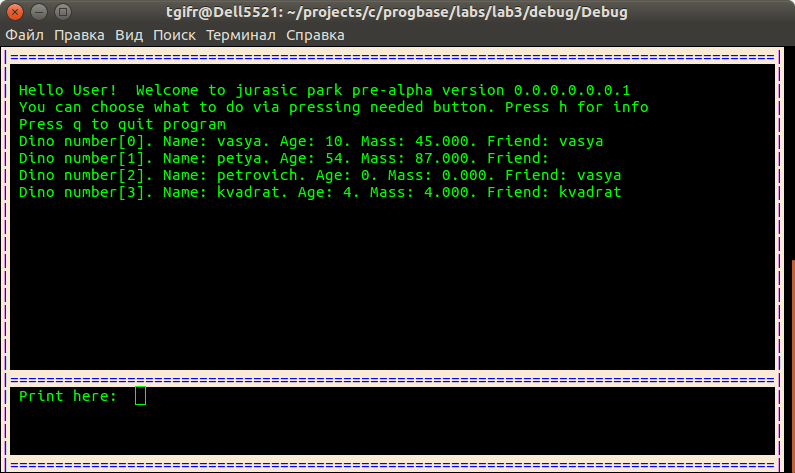
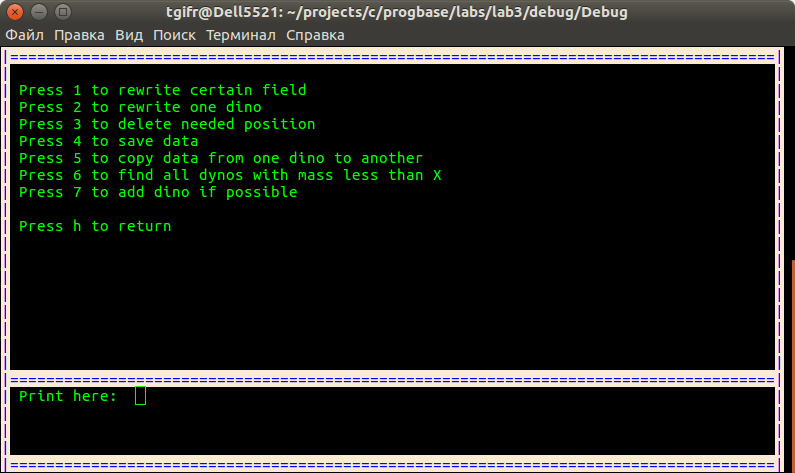
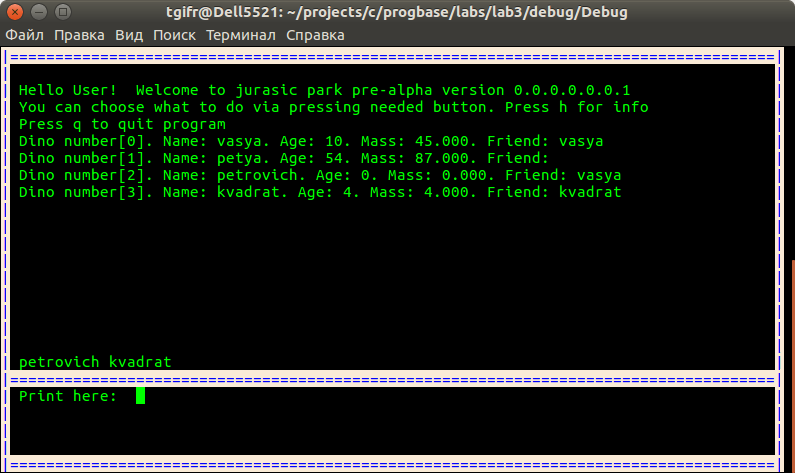
|  |
| --- |
| **totest.h** |
| **//**  **// Created by tgifr on 03.12.16.**  **//**  **#include <assert.h>**  **#ifndef LAB3\_TOTEST\_H**  **#define LAB3\_TOTEST\_H**  **void test();**  **#endif //LAB3\_TOTEST\_H** |

|  |
| --- |
| **totest.c** |
| **//**  **// Created by tgifr on 03.12.16.**  **//**  **#include <check.h>**  **#include "totest.h"**  **#include "dino.h"**  **#include "saves.h"**  **#include "list.h"**  **START\_TEST (createList\_void\_eqMAXSIZE){**  **struct List \* list1 = createDinoList();**  **ck\_assert(dino\_count(list1) == MAX\_SIZE);**  **List\_free(&list1);**  **}**  **END\_TEST**  **START\_TEST (initializeList\_void\_eqNULL){**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **ck\_assert(List\_elementAt(list1, 1) != NULL);**  **List\_free(&list1);**  **}**  **END\_TEST**  **START\_TEST (writeInStr\_dino\_eqZeroStr){**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **char \* tempStr = writeInStr(List\_elementAt(list1, 0));**  **ck\_assert(!strcmp(tempStr, " 0 0.000 0\n"));**  **List\_free(&list1);**  **free(tempStr);**  **}**  **END\_TEST**  **START\_TEST (writeFromStr\_str\_eqEnteredStr) {**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **ck\_assert(writeFromStr(list1, "Vasya 15 25 3") == 0);**  **List\_free(&list1);**  **}**  **END\_TEST**  **START\_TEST (writeFromStr\_strUncorrect\_AgeEqZero) {**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **writeFromStr(list1, "Vasya a5 5 3");**  **ck\_assert(List\_elementAt(list1, 0)->age == 0);**  **List\_free(&list1);**  **}**  **END\_TEST**  **START\_TEST (writeInStr\_dino\_eqEnteredStr){**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **writeFromStr(list1, "Vasya 15 25 3");**  **char \* tempStr = writeInStr(List\_elementAt(list1, 0));**  **ck\_assert(!strcmp(tempStr, "Vasya 15 25.000 3\n"));**  **free(tempStr);**  **List\_free(&list1);**  **}**  **END\_TEST**  **START\_TEST (writeInStr\_dinoWithUncorrectEnteredMass\_MassIsZero){**  **struct List \* list1 = createDinoList();**  **initializeList(list1);**  **writeFromStr(list1, "Vasya 15 blah 3");**  **char \* tempStr = writeInStr(List\_elementAt(list1, 0));**  **ck\_assert(!strcmp(tempStr, "Vasya 15 0.000 0\n"));**  **free(tempStr);**  **List\_free(&list1);**  **}**  **END\_TEST**  **Suite \*test\_suite() {**  **Suite \*s = suite\_create("Module");**  **TCase \*tc\_sample;**  **tc\_sample = tcase\_create("TestCase");**  **tcase\_add\_test(tc\_sample, createList\_void\_eqMAXSIZE);**  **tcase\_add\_test(tc\_sample, initializeList\_void\_eqNULL);**  **tcase\_add\_test(tc\_sample, writeInStr\_dino\_eqZeroStr);**  **tcase\_add\_test(tc\_sample, writeFromStr\_str\_eqEnteredStr);**  **tcase\_add\_test(tc\_sample, writeInStr\_dino\_eqEnteredStr);**  **tcase\_add\_test(tc\_sample, writeFromStr\_strUncorrect\_AgeEqZero);**  **tcase\_add\_test(tc\_sample, writeInStr\_dinoWithUncorrectEnteredMass\_MassIsZero);**  **suite\_add\_tcase(s, tc\_sample);**  **return s;**  **}**  **void test(){**  **Suite \*s = test\_suite();**  **SRunner \*sr = srunner\_create(s);**  **srunner\_set\_fork\_status(sr, *CK\_NOFORK*);**  **srunner\_run\_all(sr, *CK\_VERBOSE*);**  **srunner\_free(sr);**  **struct List \* list1 = createDinoList();**  **assert(dino\_count(list1) == MAX\_SIZE);**  **assert(List\_elementAt(list1, 0) != NULL);**  **List\_elementAt(list1, 2 )->next->age = 10;**  **assert(List\_elementAt(list1, 3)->age == 10);**  **initializeList(list1);**  **assert(List\_elementAt(list1, 2)->age == 0);**  **assert(List\_elementAt(list1, 3)->mass == 0.0);**  **assert(List\_elementAt(list1, 4)->friend == List\_elementAt(list1, 0));**  **assert(strcmp(List\_elementAt(list1, 7)->name, "") == 0);**  **assert(!writeFromStr(list1, "Vasya 10 15 1"));**  **assert(List\_elementAt(list1, 0)->age == 10);**  **assert(List\_elementAt(list1, 0)->mass == 15.0);**  **assert(List\_elementAt(list1, 0)->friend == List\_elementAt(list1, 0)->next);**  **assert(strcmp(List\_elementAt(list1, 0)->name, "Vasya") == 0);**  **char \* tempStr = writeInStr(List\_elementAt(list1, 0));**  **assert(strcmp(tempStr, "Vasya 10 15.000 1\n") == 0);**  **free(tempStr);**  **rewrite(list1, 1, 0);**  **assert(List\_elementAt(list1, 1)->age == 10);**  **assert(List\_elementAt(list1, 1)->mass == 15.0);**  **assert(List\_elementAt(list1, 1)->friend == List\_elementAt(list1, 1));**  **assert(strcmp(List\_elementAt(list1, 1)->name, "Vasya") == 0);**  **tempStr = writeInStr(List\_elementAt(list1, 1));**  **assert(strcmp(tempStr, "Vasya 10 15.000 1\n") == 0);**  **free(tempStr);**  **List\_free(&list1);**  **assert(list1 == NULL);**  **}** |

|  |
| --- |
| **list.c** |
| **//**  **// Created by tgifr on 19.02.17.**  **//**  **#include <assert.h>**  **#include "list.h"**  **#include "dino.h"**  **#include "dino.h"**  **struct List {**  **dino \* head;**  **};**  **struct List \* createDinoList(){**  **struct List \* result = malloc(sizeof(struct List));**  **result->head = dino\_new();**  **struct dino \* temp = result->head;**  **for (int i = 1; i < MAX\_SIZE; ++i) {**  **temp->next = dino\_new();**  **temp = temp->next;**  **}**  **return result;**  **}**  **int dino\_count(struct List \*self) {**  **int count = 0;**  **struct dino \* cur = self->head;**  **while (cur != NULL) {**  **count += 1;**  **cur = cur->next;**  **}**  **return count;**  **}**  **struct dino \* List\_elementAt(struct List \*self, int position) {**  **assert(position >= 0);**  **if (self == NULL) assert(0); // just to throw error**  **int i = 0;**  **struct dino \* cur = self->head;**  **while (cur->next != NULL && i != position) {**  **i += 1;**  **cur = cur->next;**  **}**  **if (i != position) assert(0);**  **return cur;**  **}**  **struct dino \*dino\_new() {**  **struct dino \* node = (struct dino\*)malloc(sizeof(struct dino));**  **node->next = NULL;**  **return node;**  **}**  **void List\_free(struct List \*\*self) {**  **assert(NULL != self);**  **struct dino \* temp;**  **struct dino \* cur = (\*self)->head;**  **while(cur!= NULL){**  **temp = cur;**  **cur = cur->next;**  **free(temp);**  **}**  **free(\*self);**  **\*self = NULL;**  **}** |

|  |
| --- |
| **list.h** |
| **//**  **// Created by tgifr on 19.02.17.**  **//**  **#ifndef LAB3\_LIST\_H**  **#define LAB3\_LIST\_H**  **typedef struct List List;**  **typedef struct dino dino;**  **int dino\_count(struct List \*self);**  **struct List \* createDinoList();**  **struct dino \* List\_elementAt(struct List \*self, int position);**  **struct dino \* dino\_new();**  **void List\_free(struct List \*\*self);**  **#endif //LAB3\_LIST\_H** |

**Приклади результатів**



**Висновки**

Виконавши дану лабораторну роботу було використано можливості мови С з викорситання динамічної пам’яті, структур данних, вказівників та файлових потоків.

Було виконано розділення програми на окремі модулі і їх модульне тестування.