rr

**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 №1**

**з дисципліни “ Основи програмування”**

**тема “МОДУЛІ”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-83**  **Мортіков Владислав Євгенович**  **варіант №16** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_”  2019  р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

**Мета роботи**

Навчитись створювати модулі в мові С.  
Навчитись реалізовувати основні колекції елементів різними способами.  
Навчитись розбивати код проекту на декілька файлів і компілювати його.

**Постановка завдання**

Створити у репозиторії **progbase2** директорію проекта labs/lab1/

1. Додати у директорію проекта текстовий файл data.txt, що містить набір будь-яких дробових значень, що записані через пробільні символи.
2. [Динамічний список](https://docs.google.com/presentation/d/18PQX143prl8iTkqzjQM7h1_GZQ1swnf17oQ0TLarKpw/edit" \l "slide=id.g4b564e1f14_0_41). Реалізувати список дробових чисел на основі динамічного масиву.  
   Зчитати всі дробові числа з текстового файлу і додати їх у список.  
   Вивести вміст списку у консоль.  
   Перенести всі числа, у яких нульова ціла частина, на початок списку.  
   Вивести вміст списку у консоль.
3. [Інтерфейс АТД](https://docs.google.com/presentation/d/18PQX143prl8iTkqzjQM7h1_GZQ1swnf17oQ0TLarKpw/edit" \l "slide=id.g4b564e1f14_0_65). Реалізувати дек дробових чисел на основі динамічного масиву

Створити два екземпляри даного типу даних.  
Переписати всі елементи непарних позицій елементи списку у початок першої деки, а всіх парних - у кінець другої.  
Вивести вміст обох екземплярів у консоль.  
Створити новий список і переписати у нього всі елементи з кінця першої деки, а потім з початку другої.  
Вивести вміст списку у консоль.

Весь код повинен бути розбитим на модулі відповідно до завдання:

1. main - головний модуль з функцією main()
2. list - модуль списку ([інтерфейс](https://docs.google.com/presentation/d/18PQX143prl8iTkqzjQM7h1_GZQ1swnf17oQ0TLarKpw/edit" \l "slide=id.g4b564e1f14_0_41))
3. deque - модуль даного АТД ([інтерфейс](https://docs.google.com/presentation/d/18PQX143prl8iTkqzjQM7h1_GZQ1swnf17oQ0TLarKpw/edit" \l "slide=id.g4b564e1f14_0_65))

При найменуванні модулів, типів та операцій обов'язково слідувати [рекомендаціям](https://docs.google.com/document/d/1cuPKtGcTTEXPeIWxXrxlBgx4WujB1qCKngj7useLFd0/edit).

Кожен модуль розміщувати у двох файлах (.h i .c). Компілювати проект за допомогою команди (з директорії проекта):

gcc \*.c

При створенні модулів допускається використання відкритих структур, але зовнішній код не повинен напряму доступатися до полів такої структури (перевіряється тимчасовим перенесенням відкритої структури у реалізацію, при чому єдиною помилкою має стати створення екземпляра на стеку).

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  #include "list.h"  #include "deque.h"  //------------------------------------------------  void modifyList(List \*list);  void fullDeque(Deque \*Deque, int dSize);  void modifyDeque(Deque \*firstDeque, Deque \*secondDeque, List \*mainlist);  void createListFromDequeu(Deque \*firstDeque, Deque \*secondDeque, List \*dequeList);  //------------------------------------------------  int main()  {  FILE \*myfile = fopen("data.txt", "r");  if (myfile == NULL)  {  return 1;  }  List mainList;  List\_init(&mainList);  float buf;  while (fscanf(myfile, "%f", &buf) == 1)  {  // printf("%f\n", buf);  float \*p = malloc(sizeof(float));  \*p = buf;  List\_add(&mainList, p);  }  fclose(myfile);  List\_print(&mainList);  modifyList(&mainList);  List\_print(&mainList);  puts("-----------------------------------");  Deque firstDeque;  Deque secondDeque;  List dequeList;    List\_init(&dequeList);  Deque\_init(&firstDeque);  Deque\_init(&secondDeque);  modifyDeque(&firstDeque, &secondDeque, &mainList);  createListFromDequeu(&firstDeque, &secondDeque, &dequeList);  Deque\_deinit(&firstDeque);  Deque\_deinit(&secondDeque);  List\_deinit(&mainList);  return 0;  }  void modifyList(List \*list)  {  int tmp = 0;  for (int i = 0; i < List\_size(list); i++)  {  if (\*((float \*)list->items[i]) > -1 && \*((float \*)list->items[i]) < 1)  {  float \*buf = list->items[tmp];  list->items[tmp] = list->items[i];  list->items[i] = buf;  tmp++;  }  }  }  void fullDeque(Deque \*Deque, int dSize)  {  // float e = rand()% (10 - 1 + 1 + 1);  for (int i = 0; i < dSize; i++)  {  float p = (float)i;  // scanf("%f", &p);  Deque\_pushBack(Deque, p);  }  Deque\_print(Deque);  }  void modifyDeque(Deque \*firstDeque, Deque \*secondDeque, List \*mainlist)  {  for (int i = 0; i < List\_size(mainlist); i++)  {  float buf = \*(float \*)List\_get(mainlist, i);  if (i % 2 == 0)  {  Deque\_pushFront(firstDeque, buf);  }  else  {  Deque\_pushBack(secondDeque, buf);  }  }  Deque\_print(firstDeque);  Deque\_print(secondDeque);  }  void createListFromDequeu(Deque \*firstDeque, Deque \*secondDeque, List \*dequeList)  {  while (!Deque\_isEmpty(firstDeque))  {  float \*buf = malloc(sizeof(float));  // float p = Deque\_popBack(firstDeque);  \*buf = Deque\_popBack(firstDeque);  List\_add(dequeList, buf);  }  while (!Deque\_isEmpty(secondDeque))  {  float \*buf = malloc(sizeof(float));  \*buf = Deque\_popFront(secondDeque);  List\_add(dequeList, buf);  }  List\_print(dequeList);  } |

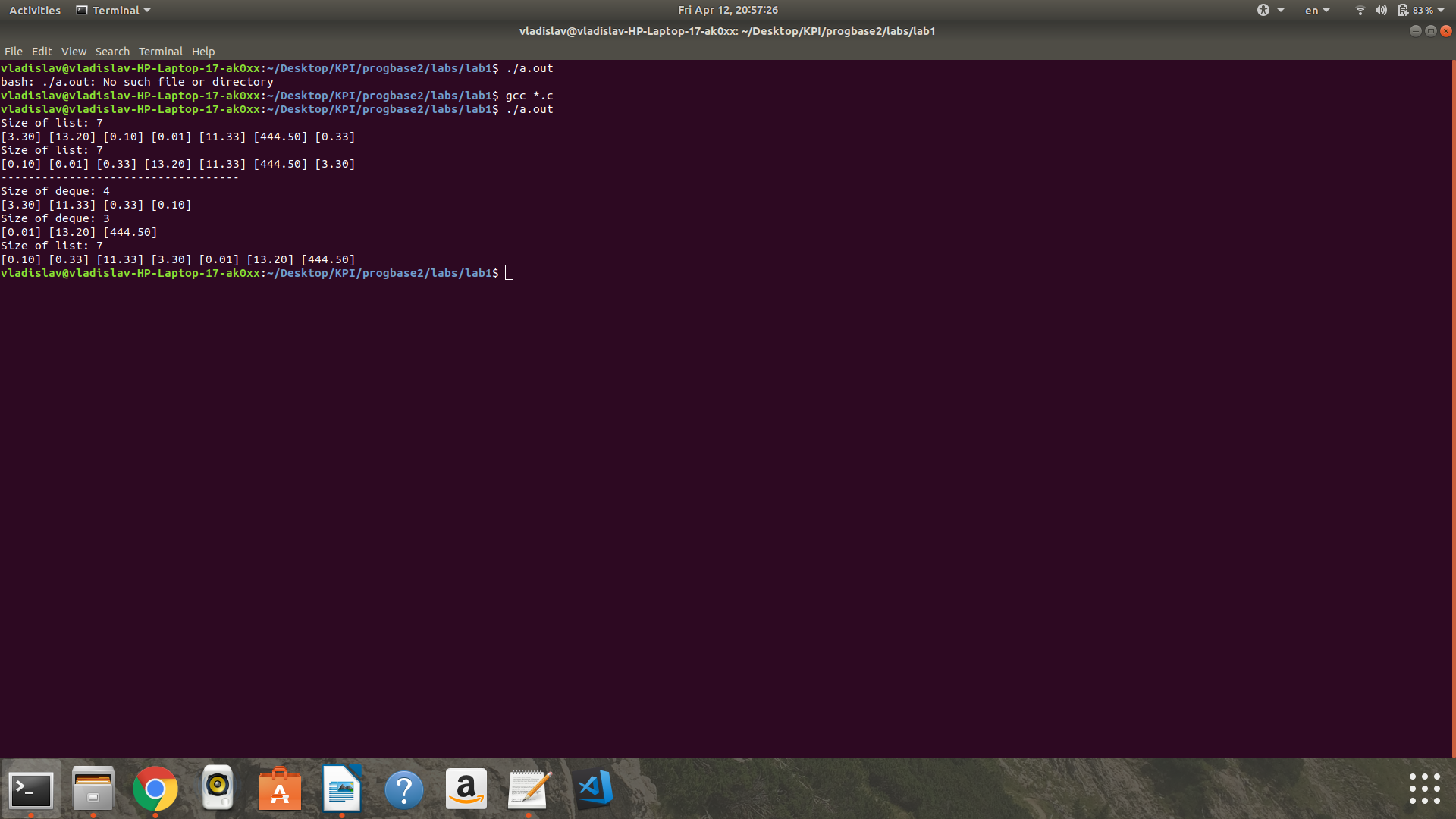
|  |
| --- |
| **deque.h** |
| #pragma once  #include <stdlib.h>  #include <stdbool.h>  #include <stdio.h>  struct \_\_Deque  {  float \*items;  int capacity;  int first;  int last;  };  typedef struct \_\_Deque Deque;  void Deque\_init(Deque \*self);  void Deque\_deinit(Deque \*self);  Deque \*Deque\_alloc(void);  void Deque\_free(Deque \*self);  void Deque\_pushBack(Deque \*self, float value);  float Deque\_popBack(Deque \*self);  void Deque\_pushFront(Deque \*self, float value);  float Deque\_popFront(Deque \*self);  size\_t Deque\_size(Deque \*self);  bool Deque\_isEmpty(Deque \*self);  void Deque\_print(Deque \*self); |

|  |
| --- |
| **deque.c** |
| #include "deque.h"  void Deque\_init(Deque \*self)  {  self->capacity = 20;  self->first = 10;  self->last = 10;  self->items = malloc(sizeof(float) \* self->capacity);  }  void Deque\_deinit(Deque \*self)  {  free(self->items);  }  void Deque\_free(Deque \*self)  {  free(self);  free(self->items);  }  void Deque\_pushFront(Deque \*self, float value)  {  if (self->first == -1)  {  self->first = 0;  self->last = 0;  }  else if (self->first == 0 || self->last == self->capacity - 1)  {  if (self->last + 1 < self->capacity)  {  int newCap = self->capacity + 1;  void \*newItems = realloc(self->items, sizeof(float) \* newCap);  if (newItems == NULL)  {  free(self->items);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  self->items = newItems;  self->capacity = newCap;  }  for (size\_t i = self->last; i >= self->capacity; i--)  {  self->items[i + 1] = self->items[i - 1];  }  self->items[self->last] = value;  self->last++;  }  else  {  self->first--;  }  self->items[self->first] = value;  }  void Deque\_pushBack(Deque \*self, float value)  {  self->items[self->last] = value;  self->last += 1;  if (self->last == self->capacity)  {  self->last = 0;  }  if (self->last == self->first)  {  fprintf(stderr, "Deque is full");  int newCap = self->capacity \* 2;  float \*newItems = realloc(self->items, sizeof(float) \* newCap);  if (newItems == NULL)  {  free(self->items);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  self->items = newItems;  self->capacity = newCap;  }  }  float Deque\_popFront(Deque \*self)  {  float value = self->items[self->first];  self->items[self->first] = -1;  if (self->first == self->last)  {  self->first = -1;  self->last = -1;  }  else if (self->first == self->capacity - 1)  {  self->first = 0;  }  else  {  self->first++;  }  return value;  }  float Deque\_popBack(Deque \*self)  {  float value = self->items[self->last - 1];  self->items[self->last] = -1;  if (self->first == self->last)  {  self->first = -1;  self->last = -1;  }  else if (self->last == 0)  {  self->last = self->capacity - 1;  }  else  {  self->last--;  }  return value;  }  size\_t Deque\_size(Deque \*self)  {  if (self->last >= self->first)  return self->last - self->first;  return self->capacity - self->first + self->last;  }  bool Deque\_isEmpty(Deque \*self)  {  if (self->first == self->last)  {  return true;  }  return false;  }  void Deque\_print(Deque \*self)  {  printf("Size of deque: %zu\n", Deque\_size(self));  for (int i = self->first; i < self->last; i++)  {  printf("[%.2f] ", self->items[i]);  }  puts("");  } |

|  |
| --- |
| **list.h** |
| #pragma once  #include <stdlib.h>  #include <stdio.h>  #include <stdbool.h>  struct \_\_List  {  void \*\* items;  size\_t size;  size\_t capacity;  };  typedef struct \_\_List List;  List \* List\_alloc (void);  void List\_free (List \* self);  void List\_init(List \* self);  void List\_deinit(List \* self);  void \* List\_get (List \* self, int index);  void List\_set (List \* self, int index, void \* value);  size\_t List\_size (List \* self);  void List\_insert (List \* self, int index, void \* value);  void List\_removeAt (List \* self, int index);  void List\_add (List \* self, void \* value);  void List\_remove (List \* self, void \* value);  int List\_indexOf (List \* self, void \* value);  bool List\_contains (List \* self, void \* value);  bool List\_isEmpty (List \* self);  void List\_clear (List \* self);  void List\_print (List \* self); |

|  |
| --- |
| **list.c** |
| #include "list.h"  List \*List\_alloc(void)  {  List \*list = malloc(sizeof(List));  List\_init(list);  return list;  }  void List\_free(List \* self)  {  List\_deinit(self);  free(self);  }  void List\_print(List \* self)  {  printf("Size of list: %zu\n", List\_size(self) );  for(int i = 0; i < List\_size(self); i++){  printf("[%.2f] ", \*((float\*)self->items[i]));  }  puts("");  }  void List\_init(List \* self)  {  self->capacity = 30;  self->size = 0;  self->items = malloc(sizeof(void \*) \* self->capacity);  }  void List\_deinit(List \* self)  {  free(self->items);  }  void \*List\_get(List \* self, int index)  {  return self->items[index];  }  void List\_set(List \* self, int index, void \* value)  {  }  size\_t List\_size(List \* self)  {  return self->size;  }  void List\_insert(List \* self, int index, void \* value)  {  if (index > self->size || index < 0)  {  fprintf(stderr, "ERROR: item cannot be inserted\n");  abort();  }  else if (index == self->size)  {  List\_add(self, value);  }  else if (index < self->size)  {  if (self->size + 1 < self->capacity)  {  int newcap = self->capacity + 1;  void \*newitems = realloc(self->items, sizeof(void \*) \* newcap);  if (newitems == NULL)  {  free(self->items);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  self->items = newitems;  self->capacity = newcap;  }  for (size\_t i = self->size; i >= index; i--)  {  self->items[i + 1] = self->items[i - 1];  }  self->items[index] = value;  self->size++;  }  }  void List\_removeAt(List \* self, int index)  {  if (index >= self->size || index < 0)  {  printf("ERROR: Index is bigger than list's size\n");  return;  }  free(self->items[index]);  for (size\_t i = index; i < self->size - 1; i++)  {  self->items[i] = self->items[i + 1];  }  self->size--;  }  void List\_add(List \* self, void \* value)  {  self->items[self->size] = value;  self->size += 1;  if (self->size == self->capacity)  {  int newcap = self->capacity \* 2;  void \*newitems = realloc(self->items, sizeof(void \*) \* newcap);  if (newitems == NULL)  {  free(self->items);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  self->items = newitems;  self->capacity = newcap;  }  }  void List\_remove(List \* self, void \* value)  {  for (size\_t i = 0; i < self->size; i++)  {  if (self->items[i] == value)  {  free(self->items[i]);  for (size\_t j = i; j < self->size; j++)  {  if (j != self->size - 1)  {  self->items[j] = self->items[j + 1];  }  }  self->size--;  return;  }  }  printf("ERROR: value that you want to remove isn't in list\n");  }  int List\_indexOf(List \* self, void \* value)  {  return -1;  }  bool List\_contains(List \* self, void \* value)  {  return false;  }  bool List\_isEmpty(List \* self)  {  return self->size == 0;  }  void List\_clear(List \* self)  {  self->size = 0;  } |

**￼Приклади результатів**

****

**Висновки**

Виконавши дану лабораторну роботу ми навчилися розподіляти програму за модулями, реалізовувати основні колекції елементів різними способами, розбивати код проекту на декілька файлів і компілювати його.

У процесі виконання програми, користувач може обрати один режим роботи.

Програма компілювалася за допомогою компілятора gcc.