

**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 ”**

|  |  |  |
| --- | --- | --- |
| **Виконала**  **студентка I курсу**  **групи КП-83**  **Снітко Маріанна Дмитрівна**  **(прізвище, ім’я, по батькові)**  **варіант № 22** |  | **Перевірив**  “**\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2018**

**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 від'ємні числа 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

**Тексти коду програм**

|  |
| --- |
| **deque.c** |
| #include "deque.h"  void Deque\_init(Deque \*self)  {  self->capacity = 20;  self->first = 10;  self->last = 10;  self->items = malloc(sizeof(double) \* 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, double 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(double) \* 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, double 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;  double \*newItems = realloc(self->items, sizeof(double) \* newCap);  if (newItems == NULL)  {  free(self->items);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  self->items = newItems;  self->capacity = newCap;  }  }  double Deque\_popFront(Deque \*self)  {  double 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;  }  double Deque\_popBack(Deque \*self)  {  double value = self->items[self->last];  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 == -1)  {  return true;  }  return false;  } |

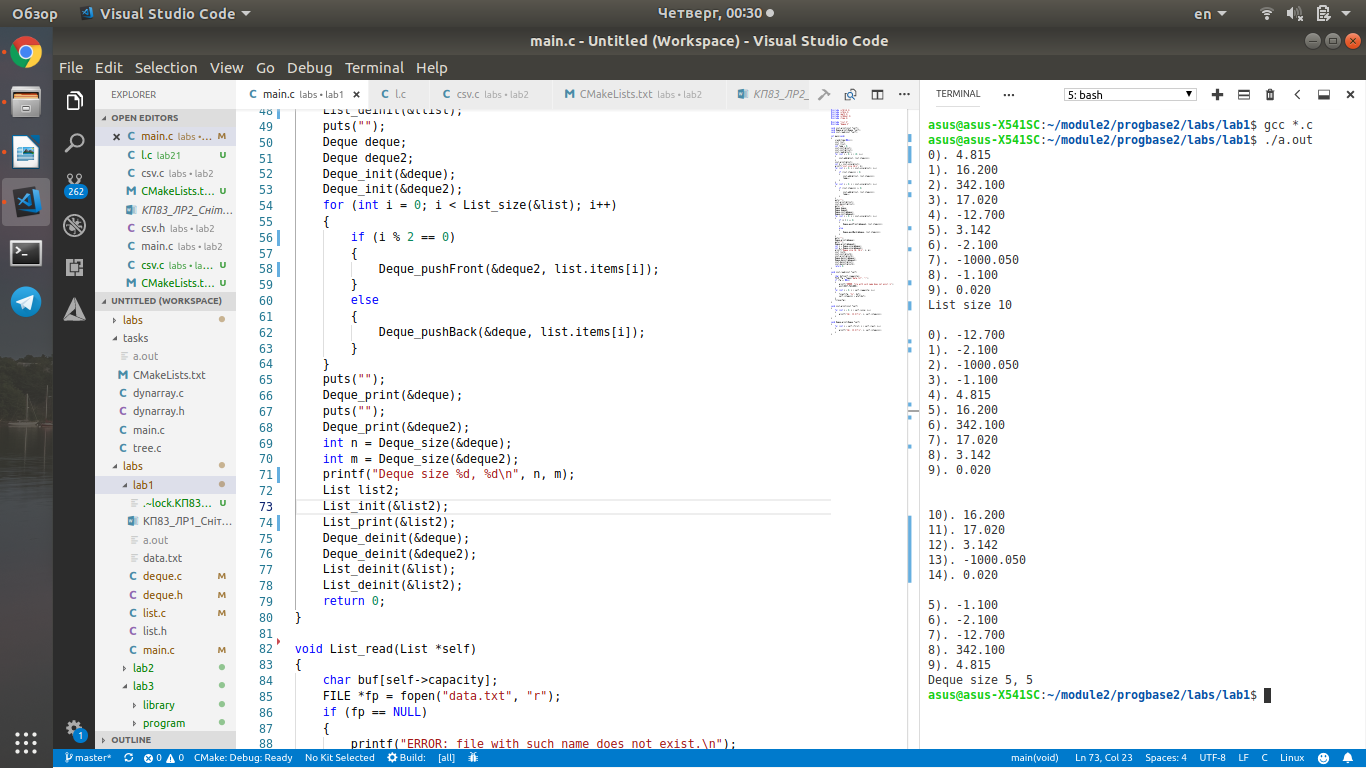
|  |
| --- |
| **deque.h** |
| #pragma once  #include <stdlib.h>  #include <stdbool.h>  struct \_\_Deque  {  double \*items;  int capacity;  int first;  int last;  };  typedef struct \_\_Deque Deque;  void Deque\_init (Deque \*self);  void Deque\_deinit (Deque \*self);  Deque \*Deque\_alloc (void); 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

|  |
| --- |
| **list.c** |
| #include "list.h"  void List\_init(List \*self)  {  self->capacity = 10;  self->size = 0;  self->items = malloc(sizeof(double) \* self->capacity);  if (self->items == NULL)  {  fprintf(stderr, "Allocating memory error\n");  abort();  }  }  void List\_deinit(List \*self)  {  free(self->items);  }  List \*List\_alloc(void)  {  List \*self = malloc(sizeof(List));  List\_init(self);  return self;  }  void List\_free(List \*self)  {  List\_deinit(self);  free(self);  }  size\_t List\_size(List \*self)  {  return self->size;  }  double List\_get(List \*self, int index)  {  return self->items[index];  }  void List\_set(List \*self, int index, double value)  {  self->items[index] = value;  }  void List\_insert(List \*self, int index, double 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(double) \* 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 < 0 || index > self->size)  {  abort();  }  else  {  for (int i = index - 1; i < self->size - 1; i++)  {  self->items[i] = self->items[i + 1];  }  self->size--;  }  }  void List\_add(List \*self, double 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(double) \* 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, double 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 does not exist\n");  }  int List\_indexOf(List \*self, double value)  {  for (size\_t i = 0; i < self->size; i++)  {  if (self->items[i] == value)  {  return i;  }  }  printf("ERROR: value does not exist\n");  return -1;  }  bool List\_contains(List \*self, double value)  {  for (size\_t i = 0; i < self->size; i++)  {  if (self->items[i] == value)  {  return 1;  }  }  return 0;  }  bool List\_isEmpty(List \*self)  {  if (self->size == 0)  {  return 1;  }  else  {  return 0;  }  }  void List\_clear(List \*self)  {  self->size = 0;  } |

|  |
| --- |
| **list.h** |
| #pragma once  #include <stdlib.h>  #include <stdbool.h>  #include <stdio.h>  struct \_\_List  {  double \*items;  size\_t capacity;  size\_t size;  };  typedef struct \_\_List List;  void List\_init (List \*self); // allocate items, set capacity and size  void List\_deinit (List \*self); // free allocated items  List \*List\_alloc (void); // allocate new List on heap, init it  void List\_free (List \*self); // deinit self, free it  size\_t List\_size (List \*self); // return number of items in list  double List\_get (List \*self, int index); // return self->items[index]  void List\_set (List \*self, int index, double value); // set self->items[index]  void List\_insert (List \*self, int index, double value); // insert, shift right  void List\_removeAt (List \*self, int index); // remove and shift left  void List\_add (List \*self, double value); // insert back  void List\_remove (List \*self, double value); // remove first by value  int List\_indexOf (List \*self, double value); // find index by value  bool List\_contains (List \*self, double value); // check by value  bool List\_isEmpty (List \*self); // check if list has any items  void List\_clear (List \*self); // make list empty |

|  |
| --- |
| **main.c** |
| #include <stdlib.h>  #include <stdio.h>  #include <math.h>  #include <stdbool.h>  #include <time.h>  #include "list.h"  #include "deque.h"  void List\_print(List \*self);  void Deque\_print(Deque \*self);  void List\_read(List \*self);  int main(void)  {  srand(time(NULL));  List list;  List llist;  int temp = 0;  List\_init(&list);  List\_init(&llist);  List\_read(&list);  for (int i = 0; i < 10; i++)  {  List\_add(&list, List\_get(&list,i));  }  List\_print(&list);  int k = List\_size(&list);  printf("List size %d\n", k);  for (int i = 0; i < List\_size(&list); i++)  {  if (List\_get(&list,i) < 0)  {  List\_add(&llist, List\_get(&list,i));  temp++;  }  }  for (int i = 0; i < List\_size(&list); i++)  {  if (List\_get(&list,i) >= 0)  {  List\_add(&llist, List\_get(&list,i));  temp++;  }  }  puts("");  List\_print(&llist);  List\_deinit(&llist);  puts("");  Deque deque;  Deque deque2;  Deque\_init(&deque);  Deque\_init(&deque2);  for (int i = 0; i < List\_size(&list); i++)  {  if (i % 2 == 0)  {  Deque\_pushFront(&deque2, List\_get(&list,i));  }  else  {  Deque\_pushBack(&deque, List\_get(&list,i));  }  }  puts("");  Deque\_print(&deque);  puts("");  Deque\_print(&deque2);  int n = Deque\_size(&deque);  int m = Deque\_size(&deque2);  printf("Deque size %d, %d\n", n, m);  List list2;  List\_init(&list2);  List\_print(&list2);  Deque\_deinit(&deque);  Deque\_deinit(&deque2);  List\_deinit(&list);  List\_deinit(&list2);  return 0;  }  void List\_read(List \*self)  {  char buf[self->capacity];  FILE \*fp = fopen("data.txt", "r");  if (fp == NULL)  {  printf("ERROR: file with such name does not exist.\n");  exit(EXIT\_FAILURE);  }  for (int i = 0; i < self->capacity; i++)  {  fscanf(fp, "%s", buf);  self->items[i] = atof(buf);  }  fclose(fp);  }  void List\_print(List \*self)  {  for (int i = 0; i < self->size; i++)  {  printf("%d). %3.3lf\n", i, self->items[i]);  }  }  void Deque\_print(Deque \*self)  {  for (int i = self->first; i < self->last; i++)  {  printf("%d). %3.3lf\n", i, self->items[i]);  }  } |

**Приклади результатів**

****

**Висновок**

Протягом виконання цієї лабораторної роботи я ознайомилася із програмуванням на мові С у спеціалізованому редакторі коду.

Більше того, під час виконання лабораторної роботи я навчилася створювати модулі в мові С. Також мною було засвоєно реалізацію основних колекцій елементів різними способами. Окрім цього, я навчилася розбивати код проекту на декілька файлів і компілювати його.

Компіляція всього коду відбувалася за допомогою компілятора gcc.

Отже, мета лабораторної роботи досягнена, всі завдання виконані і їх розвʼязання наведені.