**Отчет по лабораторной работе № 25/26** по курсу “Языки и методы программирования”

Студент группы М80-103Б-21 Фадеев Денис Вадимович, № по списку 22

Контакты e-mail: denfad2003@mail.ru, telegram: @Denissimo\_f

Работа выполнена: «7» апреля 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20\_\_ г., итоговая оценка \_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **Тема: Автоматизация сборки программ модульной структуры на языке Си с использованием утилиты make. Абстрактные типы данных. Рекурсия. Модульное программирование на языке Си.**
  2. **Цель работы:** Изучить работу с утилитой make. Создать и отладить модуль с реализацией заданного абстрактного типа данных.
  3. **Задание:** вариант №3/5 Создать тип данных дек. Сортировка слиянием.
  4. **Оборудование** :

Процессор *Intel Core i5-6200U @ 4x 2.30GHz* с ОП 16 Гб, НМД 512 Гб. Монитор *1920x1080*

**5. Программное обеспечение.**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия 20*.04 focal*

интерпретатор команд: *bash* версия 5.0.16(1)-release.

Редактор текстов *emacs* версия 26.3

**6. Идея, метод, алгоритм**

Необходимо написать отдельный файл с Си кодом, в котором будут реализованы основные методы для работы с деком, а также разработать необходимый тип данных Unit, выполняющий роль элементов дека. Для сборки проекта напишем Makefile.

**7.Сценарий выполнения работы**

1. По заданному шаблону создадим заголовочный файл deck.h. Кроме заданных методов создадим две структуры: deck и unit. В deck хранятся ссылки на передний и задний элементы дека. В unit хранится значение элемента (value) и ссылка на следующий элемент deck. Таким образом мы можем рекурсивно или в цикле проходиться по деку.
2. В Makefile создадим четыре цели: all, exe, lib и clean. Цель all зависит от exe и lib. В lib создаётся библиотека libDeck.so, где реализован сам дек. Затем в цели exe этот .so файл линкуется с main.c файлом и создаётся исполняемый файл prog. В clean удаляются исполняемый файл и файл библиотеки.
3. Алгоритм сортировки слиянием реализован с учётом особенностей дека.

Выполнение программы:

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 1**

**Enter key of first unit: 5**

**Deck created**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 2**

**Enter key of unit: 6**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 3**

**Enter key of unit: 9**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 2**

**Enter key of unit: 1**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 2**

**Enter key of unit: -5**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 6**

**Deck: -5 1 6 5 9**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 2**

**Enter key of unit: 67**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 6**

**Deck: 67 -5 1 6 5 9**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 7**

**Sorting =======**

**Deck: -5 67**

**Deck: 1 6**

**Deck: 5 9**

**============**

**Deck: -5 1 6 67**

**Deck: 5 9**

**============**

**Deck: -5 1 5 6 9 67**

**============**

**Deck sorted**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 8**

**Deck is empty: 0**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 9**

**Deck size: 6**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 5**

**Key of back unit: 67Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 6**

**Deck: -5 1 5 6 9**

**Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter:**

**4**

**Key of front unit: -5Working with deck. Enter code of command.**

**Code 1: Create deck Code 2: Push unit to front Code 3: Push unit to back**

**Code 4: Pop unit from front Code 5: Pop unit from back Code 6: Show deck**

**Code 7: Sort deck Code 8: Is deck empty Code 9: Deck size**

**Code 10: Exit**

**Enter: 6**

**Deck: 1 5 6 9**

Работа с Makefile:

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % ls**

**Makefile main.c**

**deck.c prog**

**deck.h Фадеев Денис ЛР№23.docx**

**libDeck.so**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % make clean**

**rm prog libDeck.so**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % ls**

**Makefile main.c**

**deck.c Фадеев Денис ЛР№23.docx**

**deck.h**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % make**

**gcc -shared -fPIC deck.c -o libDeck.so**

**gcc main.c -L. -lDeck -o prog**

**main.c:11:9: warning: expression result unused [-Wunused-value]**

**\*str++;**

**^~~~~~**

**1 warning generated.**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % ls**

**Makefile main.c**

**deck.c prog**

**deck.h Фадеев Денис ЛР№23.docx**

**libDeck.so**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % make clean**

**rm prog libDeck.so**

**(base) denisfadeev@MacBook-Pro-Denis lab26:25 % ls**

**Makefile main.c**

**deck.c Фадеев Денис ЛР№23.docx**

**deck.h**

**8. Распечатка протокола**

**deck.c**

#include "deck.h"

deck \* deck\_create() {

deck \*d = malloc(sizeof(deck));

d->left = NULL;

d->right = NULL;

return d;

}

bool deck\_is\_empty(deck \*d) {

if(d == NULL) return true;

bool empty;

empty = (d->left == NULL) ? true : false;

return empty;

}

void deck\_push\_front(deck \*d, int value) {

if(deck\_is\_empty(d)) {

unit \*u = (unit \*)malloc(sizeof(unit));

u->value = value;

u->next = NULL;

d->left = u;

d->right = u;

}

else {

unit \*u = (unit \*)malloc(sizeof(unit));

u->value = value;

u->next = d->left;

d->left = u;

}

}

void deck\_push\_back(deck \*d, int value) {

if(deck\_is\_empty(d)) {

unit \*u = (unit \*)malloc(sizeof(unit));

u->value = value;

u->next = NULL;

d->left = u;

d->right = u;

}

else {

unit \*u = (unit \*)malloc(sizeof(unit));

u->value = value;

u->next = NULL;

d->right->next = u;

d->right = u;

}

}

unit \* deck\_pop\_front(deck \*d) {

if(deck\_is\_empty(d)) {

printf("Deck is empty\n");

return NULL;

}

else if(d->left->next != NULL) {

unit \*u = d->left;

d->left = u->next;

u->next = NULL;

return u;

}

else {

unit \*u = d->left;

d->left = NULL;

d->right = NULL;

u->next = NULL;

return u;

}

}

unit \* deck\_pop\_back(deck \*d) {

if(deck\_is\_empty(d)) {

printf("Deck is empty\n");

return NULL;

}

else if(d->left->next != NULL) {

unit \*u = d->right;

d->right = deck\_find\_prev(d->left, u);

d->right->next = NULL;

u->next = NULL;

return u;

}

else {

unit \*u = d->right;

d->left = NULL;

d->right = NULL;

u->next = NULL;

return u;

}

}

unit \* deck\_find\_prev(unit \*front, unit \*f) {

if(front->next == f) return front;

else return deck\_find\_prev(front->next, f);

}

void deck\_print(deck \*d) {

unit \*u = d->left;

if(deck\_is\_empty(d)) {

printf("Deck is empty\n");

}

else {

printf("Deck: ");

while(u != NULL) {

printf("%d ", u->value);

u = u->next;

}

printf("\n");

}

}

size\_t deck\_size(deck \*d) {

size\_t s = 0;

if(d == NULL) return 0;

else {

unit \*u = d->left;

while(u != NULL) {

u = u->next;

s++;

}

return s;

}

}

deck \* deck\_sort(deck \*d) {

int step = 2;

int n =(int)deck\_size(d);

deck \*\*t = malloc(sizeof(deck) \* n);

for(int i = 0; i<n; i++) {

deck \*f = deck\_create();

deck\_push\_front(f, deck\_pop\_front(d) -> value);

t[i] = f;

}

int b = n;

printf("Sorting =======\n");

while(b > 1) {

int i = 0;

int k = 0;

while(i < b) {

deck \*f = deck\_create();

int count = (int)deck\_size(t[i]) + (int)deck\_size(t[i+1]);

for(int j = 0; j<count; j++) {

if(deck\_is\_empty(t[i])) {

unit \*u = deck\_pop\_front(t[i+1]);

deck\_push\_back(f, u->value);

free(u);

}

else if (deck\_is\_empty(t[i+1])) {

unit \*u = deck\_pop\_front(t[i]);

deck\_push\_back(f, u->value);

free(u);

}

else if(t[i]->left->value > t[i+1]->left->value) {

unit \*u = deck\_pop\_front(t[i+1]);

deck\_push\_back(f, u->value);

free(u);

}

else {

unit \*u = deck\_pop\_front(t[i]);

deck\_push\_back(f, u->value);

free(u);

}

}

deck\_print(f);

t[k] = f;

k++;

i+=step;

}

b = b / step + (b % step);

printf("============\n");

}

return t[0];

}

}

**deck.h**

#ifndef deck\_h

#define deck\_h

#include <stdio.h>

#include "stdlib.h"

#include <stdbool.h>

typedef struct unit{

int value;

struct unit \*next;

}unit;

typedef struct {

unit \*left;

unit \*right;

} deck;

deck \* deck\_create(void);

bool deck\_is\_empty(deck \*d);

void deck\_push\_front(deck \*d, int value);

void deck\_push\_back(deck \*d, int velue);

unit \* deck\_pop\_front(deck \*d);

unit \* deck\_pop\_back(deck \*d);

unit \* deck\_find\_prev(unit \*front, unit \*f);

void deck\_print(deck \*d);

size\_t deck\_size(deck \*d);

deck \* deck\_sort(deck \*d);

#endif

**main.c**

#include <stdio.h>

#include "stdlib.h"

#include "stdbool.h"

#include "string.h"

#include "deck.h"

bool isInt(const char\*str) {

while(\*str) {

if((\*str< '0' || \*str > '9') && \*str != '-' && \*str != '.')

return false;

\*str++;

}

return true;

}

int main(int argc, const char \* argv[]) {

deck \*d = NULL;

bool working = true;

while(working) {

int input = 0;

char h[] = "";

while(input == 0){

printf("Working with deck. Enter code of command.\nCode 1: Create deck\t\tCode 2: Push unit to front\t\tCode 3: Push unit to back\nCode 4: Pop unit from front\t\tCode 5: Pop unit from back\t\tCode 6: Show deck\nCode 7: Sort deck\tCode 8: Is deck empty\tCode 9: Deck size\nCode 10: Exit\nEnter: ");

scanf("%s", h);

if(!strcmp("1",h)){

input = 1;

}

else if(!strcmp("2",h)) {

input = 2;

}

else if(!strcmp("3",h)) {

input = 3;

}

else if(!strcmp("4",h)) {

input = 4;

}else if(!strcmp("5",h)) {

input = 5;

}else if(!strcmp("6",h)) {

input = 6;

}else if(!strcmp("7",h)) {

input = 7;

}else if(!strcmp("8",h)) {

input = 8;

}else if(!strcmp("9",h)) {

input = 9;

}else if(!strcmp("10",h)) {

input = 10;

}else {

printf("Incomprehensible input\n");

}

}

switch(input){

case 1:

{

if(d != NULL)

printf("Deck already exists\n");

else {

char c[] = "";

int key = 0;

printf("Enter key of first unit: ");

scanf("%s", c);

if(isInt(c)){

key = atoi(c);

d = deck\_create();

deck\_push\_front(d, key);

printf("Deck created\n");

}

else {

printf("Not a number\n");

}

}

break;

}

case 2:

{

if(d != NULL){

char f[100] = "";

int key = 0;

printf("Enter key of unit: ");

scanf("%s", f);

if(isInt(f)){

key = atoi(f);

deck\_push\_front(d, key);

} else printf("Not a number\n");

}

else printf("Deck not exists\n");

break;

}

case 3:

{

if(d != NULL){

char f[100] = "";

int key = 0;

printf("Enter key of unit: ");

scanf("%s", f);

if(isInt(f)){

key = atoi(f);

deck\_push\_back(d, key);

} else printf("Not a number\n");

}

else printf("Deck not exists\n");

break;

}

case 4:

{

if(d != NULL){

printf("Key of front unit: %d", deck\_pop\_front(d) -> value);

}

else printf("Deck not exists\n");

break;

}

case 5: {

if(d != NULL){

printf("Key of back unit: %d", deck\_pop\_back(d) -> value);

}

else printf("Deck not exists\n");

break;

}

case 6:

if(d != NULL) {

deck\_print(d);

}

else printf("Deck not exists\n");

break;

case 7:

if(d != NULL) {

d = deck\_sort(d);

printf("Deck sorted\n");

}

else printf("Deck not exists\n");

break;

case 8:

if(d != NULL) {

printf("Deck is empty: %d\n", deck\_is\_empty(d));

}

else printf("Deck not exists\n");

break;

case 9:

if(d != NULL) {

printf("Deck size: %zu\n", deck\_size(d));

}

else printf("Deck not exists\n");

break;

case 10:

printf("Stop program\n");

working = false;

break;

}

}

return 0;

}

Makefile

all: exe lib

exe: deck.h main.c lib

gcc main.c -L. -lDeck -o prog

lib: deck.h deck.c

gcc -shared -fPIC deck.c -o libDeck.so

clean:

-rm prog libDeck.so

**9. Дневник отладки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. Или Дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | - | - | - | - | - | - |

1. **Замечания автора** -
2. **Выводы**

Результатом лабораторной работы стала программа и файл для её сборки. В процессе выполнения задания были изучены основы работы с утилитой make. Синтаксис Makefile отдалённо напоминает Dockerfile для системы контейниризации, но задачи файлов разные. Определённо, утилита make значительно упрощает жизнь при отладки программ и их сборки. Реализация абстрактного типа данных дек оказалось не самой лёгкой задачей, по началу не было понятно как вообще его реализовывать, в какой форме должна быть описана данная структура. Но в итоге, использовав рекурсии и итерационные методы, я смог добиться выполнения всего функционала. Алгоритм сортировки у меня получился достаточно хитрым, в нём активно используется работа с памятью и ссылки.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_