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
	-	-	-	-	-	-

10. Замечания автора -

11. 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 _____