Отчёт по лабораторных работах №25-26.

по курсу «Языки и методы программирования».

Выполнил студент группы М8О-111Б-23: Тимофеева Ирина
№ по списку 21

Работа выполнена: «18» мая 2024 г.

Преподаватель: каф. 806 Никулин Сергей Петрович

Входной контроль знаний с оценкой: _____

Отчет сдан «18» мая 2024 г.

Итоговая оценка: _____

Подпись преподавателя: _____

1.Тема: абстрактные типы данных, рекурсия, модульное программирование на ЯП Си. Автоматизация сборки программ модульной структуры с использованием утилиты make.

2. Цель работы: применение различных сортировок к различным типам данных и обучение по работе с утилитой make.

3. Задание: АТД - дек, процедура - слияние двух стеков, деков, списков или очередей, упорядоченных по возрастанию, с сохранением порядка, метод - сортировка слиянием.

4. Оборудование: Оборудование ПЭВМ студента, если использовалось:
Процессор AMD Ryzen 5 5600H, ОП 16 ГБ, SSD 250 ГБ, мониторы 15" Full Hd Display и 27" BenQ
BL2780T. Другие устройства: принтер Canon MG4520S, мышь Logitech g403, наушники HyperX Cloud

5. Программное обеспечение: Программное обеспечение ПЭВМ студента, если использовалось:
Операционная система семейства Ubuntu, наименование версия VirtualBox Ubuntu 20.04.3
LTS,
интерпретатор команд bash версия 5.0.17. Система программирования C. Редактор текстов
VI версия 8.1

6. Идея, метод, алгоритм решения задачи [в формах: словесной, псевдокода, графической (блок-схема,
диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:
Продумать основную модульную структуру будущей программы.

Написать код.
Написать make файл.
Протестировать.

7. Сценарий выполнения работы (план работы, первоначальный текст программы в черновике (можно
на отдельном листе) и тесты либо соображения по тестированию).

Makefile:

```
# makefile
deck: deck.o sort.o
	cc -o sorting deck.o sort.o
deck.o: sort.c deck.c deck.h
	cc -c sort.c deck.c
deck.c:
	sorting
```

Заголовочный файл deck.h:

```
#ifndef _DECK_H_
#define _DECK_H_
#define N 12

typedef int ValueType;

// структура-дек
typedef struct {
   ValueType a[N];
   int head;      //индекс первого элемента
   int tail;      //индекс последнего элемента
   size_t size;
} Deck;

// итератор для навигации по элементам массива дека
typedef ValueType* iterator;
// итератор начало массива
iterator array_begin(ValueType a[]);
// итератор конец массива
iterator array_end(ValueType a[], int n);
// следующий элемент массива
iterator array_next(iterator i);

// проверка на нечетность числа
int odd(int arg);
```

```
// инициализация дека
Deck* deck_create();
// удаление дека
void deck_destroy(Deck* d);
// пустое значение
ValueType EmptyValue();
// индекс следующего за indx элемента дека
int next_index(int indx);
// индекс предыдущего перед indx элемента дека
int prev_index(int indx);
// добавление в начало
void push_front(Deck* d, ValueType x);
// добавление в конец
void push_back(Deck* d, ValueType x);
// удаление с конца
ValueType pop_front(Deck* d);
// удаление с конца
ValueType pop_back(Deck* d);
// печать всех элементов дека
void DeckPrint(Deck* d);
// копирование элементов массива дека
ValueType* CopyData(Deck* deck, int L, int R);
// сортировка массива слиянием
void SortData(ValueType* arr, int n);
// слияние массива с элементами дека
void MergeData(Deck* deck, ValueType arr[], int a, int b);
// Функция сортировки слиянием
void DeckMergeSort(Deck* deck, int L, int R);

#endif
```

Файл с функциями дека deck.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "deck.h"

iterator array_begin(ValueType a[]){ return &a[0]; } // начало массива
iterator array_end(ValueType a[], int n){ return &a[n]; } // конец массива
iterator array_next(iterator i) { return ++i; } // следующий элемент массива

// проверка на нечетность числа
int odd(int arg) { return arg % 2; }

// инициализация дека
Deck* deck_create()
```

```
{
  Deck* deck = (Deck*)malloc(sizeof(Deck));
  deck->head = deck->tail = deck->size = 0;
  int i;
  for (i = 0; i < N; i++) deck->a[i] = -1;
  return deck;
}

// удаление дека
void deck_destroy(Deck* d)
{
        free(d);
}

// пустое значение
ValueType EmptyValue(){  return -1; }

// индекс следующего за indx элемента дека
int next_index(int indx)
{
        if (indx>=N-1)
                return 0;
        else
                return indx+1;
}

// индекс предыдущего перед indx элемента дека
int prev_index(int indx)
{
        if (indx <= 0)
                return N-1;
        else
                return indx-1;
}

// добавление в начало
void push_front(Deck* d, ValueType x)
{
        if (d->size<N)
        {
        if (d->size > 0) d->head = prev_index(d->head);
        d->a[d->head] = x;
        d->size++;
        }
        else
```

```
		printf("no space for add data!\n");
}

// добавление в конец
void push_back(Deck* d, ValueType x)
{
	if (d->size<N)
	{
	if (d->size > 0) d->tail = next_index(d->tail);
	d->a[d->tail] = x;
	d->size++;
	}
	else
		printf("no space for add data!\n");
}

// удаление с конца
ValueType pop_front(Deck* d)
{
	ValueType res = 0;
	if (d->size > 0)
	{
		res = d->a[d->head];
		d->a[d->head] = EmptyValue();
		if (d->head == d->tail)d->tail = prev_index(d->tail);
		d->head = next_index(d->head);
		d->size--;
	return res;
	}
	else
	{
		d->head = d->tail = 0;
	return EmptyValue();
   }
}

// удаление с конца
ValueType pop_back(Deck* d)
{
	ValueType res = 0;
   if (d->size > 0)
	{
		res = d->a[d->tail];
		d->a[d->tail] = EmptyValue();
	d->tail = prev_index(d->tail);
```

```
        d->size--;
        return res;
    }
        else
        {
                d->head = d->tail = 0;
        return EmptyValue();
    }
}

// печать всех элементов дека
void DeckPrint(Deck* d)
{
        int i;
        for (i = 0; i < N; i++)
        {
                if (d->a[i]>=0)
                        printf("[%4d]", d->a[i]);
                else
                        printf("[    ]");
        }
        printf("\n");
}

// копирование элементов массива дека
ValueType* CopyData(Deck* deck, int L, int R)
{
        int size = R - L + 1;
        ValueType *tmp = (ValueType*)malloc(size * sizeof(ValueType));
        int i;
        iterator it = array_end(deck->a, L);
        for (i = 0; i < size; i++)
        {
                tmp[i] = *it;
                if (i < size-1) it = array_next(it);
        }
        return tmp;
}

// сортировка массива слиянием
void SortData(ValueType* arr, int n)
{
        if (n < 2) return;
        int mid = (n / 2); // определяем середину последовательности
        if (odd(n)) mid++;
```

```
	int i = 0;  // начало первого пути
	int j = mid; // начало второго пути
	ValueType *tmp = (ValueType*)malloc(n*sizeof(ValueType)); // дополнительный
массив
	//ValueType tmp[n];
	int step;
	for (step = 0; step < n; step++) // для всех элементов дополнительного массива
	{
	// записываем в формируемую последовательность меньший из элементов двух
путей
	// или остаток первого пути если j > r
	if (j > n-1 || (i < mid && arr[i] < arr[j]))
	{
		tmp[step] = arr[i];
		i++;
		}
	else
	{
		tmp[step] = arr[j];
		j++;
	}
	}
	// переписываем сформированную последовательность в исходный массив
	for (step = 0; step < n; step++) arr[step] = tmp[step];
}

// слияние массива с элементами дека
void MergeData(Deck* deck, ValueType arr[], int a, int b)
{
	int i;
	iterator it = array_end(deck->a, a);
	for (i = 0; i < b - a + 1; i++)
	{
		*it = arr[i];
		if (i < b - a) it = array_next(it);
	}
}

// Функция сортировки слиянием
void DeckMergeSort(Deck* deck, int L, int R)
{
  if (L == R) return; // границы сомкнулись
  int mid = (L + R) / 2; // определяем середину последовательности
  // и рекурсивно вызываем функцию сортировки для каждой половины
  DeckMergeSort(deck, L, mid);
```

```
  DeckMergeSort(deck, mid+1, R);
  int size = R-L+1;
  ValueType* tmp = CopyData(deck, L, R);
  SortData(tmp, size);
  MergeData(deck, tmp, L, R);
}
```

Файл decksort.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "deck.h"

int main()
{
  // инициализация генератора случайных чисел
  srand(time(NULL));
  // создаине дека
  Deck* FirstDeck = deck_create();

  int i;
  char c;
  int done = 0;
  while (!done)
  {
     printf("Main menu:\n");
     printf("1 - make deck\n");
     printf("2 - print deck\n");
     printf("3 - sort deck\n");
     printf("0 - exit\n");
     printf("Enter command: ");
     c = getchar();
     getchar();
     printf("\n-----------------------------------------------\n");
     switch (c)
     {
     case '0':
        done = 1;
        break;
     case '1':
           // заполнение дека случайными числами
           for (i=0; i<N; i++)
              push_back(FirstDeck, rand() % 100);
           if (FirstDeck->size > 0)
```

```
            printf("Deck is created!\n");
          else
          printf("Error of deck creating!\n");
        break;
    case '2':
        printf("Deck is:\n");
          DeckPrint(FirstDeck);
        printf("\n");
        break;
    case '3':
        DeckMergeSort(FirstDeck, 0, N-1);
        printf("Deck sort completed!\n");
        break;
    default:
        printf("incorrect input!\n");
        printf("\n");
    }
  }

 deck_destroy(FirstDeck);
 return 0;
}
```

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми
примерами, подписанный преподавателем) + makefile.

```
#листинг программы работы с деком
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ cat deck.h
#ifndef _DECK_H_
#define _DECK_H_
#define N 12

typedef int ValueType;

// структура-дек
typedef struct {
   ValueType a[N];
   int head;      //индекс первого элемента
   int tail;      //индекс последнего элемента
   size_t size;
} Deck;

// итератор для навигации по элементам массива дека
typedef ValueType* iterator;
```

```
// итератор начало массива
iterator array_begin(ValueType a[]);
// итератор конец массива
iterator array_end(ValueType a[], int n);
// следующий элемент массива
iterator array_next(iterator i);

// проверка на нечетность числа
int odd(int arg);

// инициализация дека
Deck* deck_create();
// удаление дека
void deck_destroy(Deck* d);
// пустое значение
ValueType EmptyValue();
// индекс следующего за indx элемента дека
int next_index(int indx);
// индекс предыдущего перед indx элемента дека
int prev_index(int indx);
// добавление в начало
void push_front(Deck* d, ValueType x);
// добавление в конец
void push_back(Deck* d, ValueType x);
// удаление с конца
ValueType pop_front(Deck* d);
// удаление с конца
ValueType pop_back(Deck* d);
// печать всех элементов дека
void DeckPrint(Deck* d);
// копирование элементов массива дека
ValueType* CopyData(Deck* deck, int L, int R);
// сортировка массива слиянием
void SortData(ValueType* arr, int n);
// слияние массива с элементами дека
void MergeData(Deck* deck, ValueType arr[], int a, int b);
// Функция сортировки слиянием
void DeckMergeSort(Deck* deck, int L, int R);

#endif

irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ cat deck.c
#include <stdio.h>
#include <stdlib.h>
#include "deck.h"
```

```
iterator array_begin(ValueType a[]){ return &a[0]; } // начало массива
iterator array_end(ValueType a[], int n){ return &a[n]; } // конец массива
iterator array_next(iterator i) { return ++i; } // следующий элемент массива

// проверка на нечетность числа
int odd(int arg) { return arg % 2; }

// инициализация дека
Deck* deck_create()
{
  Deck* deck = (Deck*)malloc(sizeof(Deck));
  deck->head = deck->tail = deck->size = 0;
  int i;
  for (i = 0; i < N; i++) deck->a[i] = -1;
  return deck;
}

// удаление дека
void deck_destroy(Deck* d)
{
        free(d);
}

// пустое значение
ValueType EmptyValue(){  return -1; }

// индекс следующего за indx элемента дека
int next_index(int indx)
{
        if (indx>=N-1)
                return 0;
        else
                return indx+1;
}

// индекс предыдущего перед indx элемента дека
int prev_index(int indx)
{
        if (indx <= 0)
                return N-1;
        else
                return indx-1;
}
```

```
// добавление в начало
void push_front(Deck* d, ValueType x)
{
	if (d->size<N)
	{
	if (d->size > 0) d->head = prev_index(d->head);
	d->a[d->head] = x;
	d->size++;
	}
	else
		printf("no space for add data!\n");
}

// добавление в конец
void push_back(Deck* d, ValueType x)
{
	if (d->size<N)
	{
	if (d->size > 0) d->tail = next_index(d->tail);
	d->a[d->tail] = x;
	d->size++;
	}
	else
		printf("no space for add data!\n");
}

// удаление с конца
ValueType pop_front(Deck* d)
{
	ValueType res = 0;
	if (d->size > 0)
	{
		res = d->a[d->head];
		d->a[d->head] = EmptyValue();
		if (d->head == d->tail)d->tail = prev_index(d->tail);
		d->head = next_index(d->head);
		d->size--;
      return res;
	}
	else
	{
		d->head = d->tail = 0;
      return EmptyValue();
   }
}
```

```
// удаление с конца
ValueType pop_back(Deck* d)
{
        ValueType res = 0;
   if (d->size > 0)
        {
                res = d->a[d->tail];
                d->a[d->tail] = EmptyValue();
      d->tail = prev_index(d->tail);
      d->size--;
      return res;
   }
        else
        {
                d->head = d->tail = 0;
      return EmptyValue();
   }
}

// печать всех элементов дека
void DeckPrint(Deck* d)
{
        int i;
        for (i = 0; i < N; i++)
        {
                if (d->a[i]>=0)
                        printf("[%4d]", d->a[i]);
                else
                        printf("[    ]");
        }
        printf("\n");
}

// копирование элементов массива дека
ValueType* CopyData(Deck* deck, int L, int R)
{
        int size = R - L + 1;
        ValueType *tmp = (ValueType*)malloc(size * sizeof(ValueType));
        int i;
        iterator it = array_end(deck->a, L);
        for (i = 0; i < size; i++)
        {
                tmp[i] = *it;
                if (i < size-1) it = array_next(it);
```

```
        }
        return tmp;
}

// сортировка массива слиянием
void SortData(ValueType* arr, int n)
{
        if (n < 2) return;
        int mid = (n / 2); // определяем середину последовательности
        if (odd(n)) mid++;
        int i = 0;  // начало первого пути
        int j = mid; // начало второго пути
        ValueType *tmp = (ValueType*)malloc(n*sizeof(ValueType)); // дополнительный
массив
        //ValueType tmp[n];
        int step;
        for (step = 0; step < n; step++) // для всех элементов дополнительного массива
        {
        // записываем в формируемую последовательность меньший из элементов двух
путей
        // или остаток первого пути если j > r
        if (j > n-1 || (i < mid && arr[i] < arr[j]))
        {
                tmp[step] = arr[i];
                i++;
                }
        else
        {
                tmp[step] = arr[j];
                j++;
        }
        }
        // переписываем сформированную последовательность в исходный массив
        for (step = 0; step < n; step++) arr[step] = tmp[step];
}

// слияние массива с элементами дека
void MergeData(Deck* deck, ValueType arr[], int a, int b)
{
        int i;
        iterator it = array_end(deck->a, a);
        for (i = 0; i < b - a + 1; i++)
        {
                *it = arr[i];
                if (i < b - a) it = array_next(it);
```

```
        }
}

// Функция сортировки слиянием
void DeckMergeSort(Deck* deck, int L, int R)
{
  if (L == R) return; // границы сомкнулись
  int mid = (L + R) / 2; // определяем середину последовательности
  // и рекурсивно вызываем функцию сортировки для каждой половины
  DeckMergeSort(deck, L, mid);
  DeckMergeSort(deck, mid+1, R);
  int size = R-L+1;
  ValueType* tmp = CopyData(deck, L, R);
  SortData(tmp, size);
  MergeData(deck, tmp, L, R);
}

# листинг основной программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ cat decksort.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "deck.h"

int main()
{
  // инициализация генератора случайных чисел
  srand(time(NULL));
  // создаине дека
  Deck* FirstDeck = deck_create();

  int i;
  char c;
  int done = 0;
  while (!done)
  {
     printf("Main menu:\n");
     printf("1 - make deck\n");
     printf("2 - print deck\n");
     printf("3 - sort deck\n");
     printf("0 - exit\n");
     printf("Enter command: ");
     c = getchar();
     getchar();
     printf("\n-----------------------------------------------\n");
```

```
    switch (c)
    {
    case '0':
       done = 1;
       break;
    case '1':
          // заполнение дека случайными числами
          for (i=0; i<N; i++)
            push_back(FirstDeck, rand() % 100);
          if (FirstDeck->size > 0)
            printf("Deck is created!\n");
          else
          printf("Error of deck creating!\n");
       break;
    case '2':
       printf("Deck is:\n");
          DeckPrint(FirstDeck);
       printf("\n");
       break;
    case '3':
       DeckMergeSort(FirstDeck, 0, N-1);
       printf("Deck sort completed!\n");
       break;
    default:
       printf("incorrect input!\n");
       printf("\n");
    }
   }

  deck_destroy(FirstDeck);
  return 0;
}

#компиляция программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26/decksort$ make
cc -c decksort.c deck.c
decksort.c: In function ‘main’:
decksort.c:9:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-
declaration]
   9 |   srand(time(NULL));
     |         ^~~~
cc -o sorting deck.o decksort.o
```

```
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ cat makefile
# makefile
deck: deck.o decksort.o
	cc -o sorting deck.o decksort.o
deck.o: decksort.c deck.c deck.h
	cc -c decksort.c deck.c
deck.c:
	sorting

# компиляция и сборка программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ ls
deck.c  deck.h  makefile  decksort.c
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ make
cc -c decksort.c deck.c
decksort.c: In function ‘main’:
decksort.c:9:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-
declaration]
   9 |   srand(time(NULL));
     |         ^~~~
cc -o sorting deck.o decksort.o
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ ls
deck.c  deck.h  deck.o  decksort.c  decksort.o  makefile  sorting

#запуск программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab26$ ./sorting
Main menu:
1 - make deck
2 - print deck
3 - sort deck
0 - exit
Enter command: 1

---------------------------------------------
Deck is created!
Main menu:
1 - make deck
2 - print deck
3 - sort deck
0 - exit
Enter command: 2

---------------------------------------------
Deck is:
[  50][  35][  91][  50][  94][  17][  20][  33][  29][  54][  22][  34]
```

```
Main menu:
1 - make deck
2 - print deck
3 - sort deck
0 - exit
Enter command: 3

---------------------------------------------
Deck sort completed!
Main menu:
1 - make deck
2 - print deck
3 - sort deck
0 - exit
Enter command: 2

---------------------------------------------
Deck is:
[  17][  20][  22][  29][  33][  34][  35][  50][  50][  54][  91][  94]

Main menu:
1 - make deck
2 - print deck
3 - sort deck
0 - exit
Enter command: 0
```

9. Дневник отладки должен содержать дату и время сеансов отладки и основные события (ошибки в
сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки
приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в
написании и отладке программы.

10. Замечания автора по существу работы:

11. Выводы: в ходе лабораторной работы я применила различные сортировки к различным типам данных и поработала с утилитой make.

Подпись студента _____