

Московский Авиационный Институт  
(Национальный Исследовательский Университет)  
**Факультет прикладной математики и физики**  
Кафедра вычислительной математики и программирования

**КУРСОВАЯ РАБОТА**  
**по курсу**  
**"Информатика"**  
"Архитектура ЭВМ, системное программное обеспечение"  
**II семестр**  
Задание 9 "Сортировка и поиск"

Студент: Пашкевич А. Р.

Группа: 08-107, № по списку 10

Руководитель: Ридли М. К.

Ридли А. Н.

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

**Москва, 2017**

## **АННОТАЦИЯ**

В данном документе описывается программа, написанная в соответствии с постановкой задачи на курсовое проектирование по теме "Сортировка и поиск" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". Данная программа предназначена для обработки таблиц с ключом вещественного типа и данными в виде изображения ASCII-графики. Входными данными является текстовый файл содержащий изображение ASCII-графики в виде нумерованных строк. Для проверки работы программы разработан тестовый пример. Результаты тестирования доказывают, что программа правильно выполняет все операции по обработке входных данных и формирования выходных данных.

## 1. ПОСТАНОВКА ЗАДАЧИ

Составить программу на Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Программа должна вводить значения элементов неупорядоченной таблицы и проверять работу процедуры сортировки в трех случаях:

1. элементы таблицы с самого начала упорядочены
2. элементы таблицы расставлены в обратном порядке
3. элементы таблицы не упорядочены

В последнем случае можно использовать встроенные процедуры генерации псевдослучайных чисел.

Для каждого вызова процедуры сортировки необходимо печатать исходное состояние таблицы и результаты сортировки.

После выполнения сортировки программа должна вводить ключи и для каждого из них выполнять поиск в упорядоченной таблице с помощью процедуры двоичного поиска и печатать найденные элементы, если они присутствуют в таблице.

В процессе отладки и тестирования рекомендуется использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода. Тестовые данные необходимо заранее поместить в текстовые файлы.

**В качестве текста для записей таблицы взять** — изображение ASCII-графики

**Вариант сортировки** — 4) Шейкер-сортировка.

**Структура таблицы:** тип ключа — вещественный, хранение данных и ключей — отдельно.

## 2. МЕТОД РЕАЛИЗАЦИИ

- 1) Создать набор тестовых файлов, содержащих изображение ASCII-графики.
- 2) Использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода.
- 3) Произвести шейкерную сортировку по ключу таблицы.
- 4) Распечатать отсортированную таблицу.
- 5) Расставить элементы таблицы в обратном порядке.
- 6) Расставить элементы таблицы в псевдослучайном порядке.
- 7) Осуществить двоичный поиск элемента по ключу.
- 8) Разработать меню действий

## 3. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Данная программа осуществляет шейкерную сортировку и двоичный поиск по ключу в таблице. После этого программа распечатывает результат сортировки или выводит найденную по ключу строку.

Сортировка перемешиванием, или ***Шейкерная сортировка***, или двунаправленная (англ. Cocktail sort) — разновидность пузырьковой сортировки. Метод можно описать так: границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

#### **4. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ**

Необходимое программное и аппаратное обеспечение: компилятор gcc

Операционная система: GNU/Linux, UNIX, MS Windows

Язык: Си

Система программирования: Си

Способ вызова и загрузки: bash

#### **4. ОРГАНИЗАЦИЯ ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ**

##### **Входные данные:**

На первой строке находится число *M*, указывающее количество записей в таблице. На следующих *M* строках находятся пары ключ-значение, разделенные знаком табуляции.

Типы ключа — вещественный, значения — изображение ASCII-графики.

Далее до конца файла находятся ключи, которые нужно искать в таблице.

##### **Выходные данные:**

Таблица, состоящая из тех же строк, что и входная, но расположенных в отсортированном порядке.

Для каждого ключа, который нужно было найти в таблице вывести соответствующие значения, разделенные знаком табуляции или "Not found", если такого ключа в таблице нет.

#### **5. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ**

```
typedef struct {  
    float key;  
} key_t;  
  
typedef struct {  
    char data[MAXLEN];  
} data_t;
```

#### **3. ОРГАНИЗАЦИЯ ИСХОДНОГО КОДА**

Программа состоит из одного файла **cocktail\_sort.c**

Исполняемый файл программы собирается:

```
gcc -Wall -pedantic -ansi -std=c99 -O2 -g cocktail_sort.c -o prog -lm
```

## 4. КАК ЭТО РАБОТАЕТ

Программа принимает на вход текстовый файл, считывает число  $m$  — количество строк таблицы. Создает два массива: массив ключей и массив значений, наполняет массивы данными из входного файла, распечатывает таблицу.

Далее программа считывает из входного файла ключи, которые необходимо найти в таблице и выводит результат.

После чего выводится меню действий состоящее из следующих пунктов:

Program menu:

- 1) — Print
- 2) — Binary search
- 3) — Cocktail\_sort
- 4) — Mix table
- 5) — Reverse
- 6) — Exit

**void printTable**

печать таблицы

**void binarysearch**

бинарный поиск по ключу

**void cocktail\_sort**

шейкерная сортировка

**void reverse**

переставляет все элементы в массиве в обратном порядке

**void exchange**

меняет местами  $k[a]$   $k[b]$  и  $v[a]$  и  $v[b]$

**void mix**

перемешивание строк таблицы в псевдослучайном порядке

**int wild**

возвращает "случайное" число

**int is\_sorted**

проверяет отсортирован ли массив ключей по возрастанию

**int match**

сравнивает значения ключей для `is_sorted`

```
$ ./prog test4.txt  
File test4.txt open for reading...
```

24 rows:

#	key	Data
0	0.04	_____♥_Я
1	0.39	_____♥_Л
2	0.94	_____♥_Ю
3	1.08	_____♥_Б
4	1.87	_____♥_Л
5	2.07	_____♥_Ю
6	2.30	_____♥
7	3.44	_____♥_Т
8	3.98	_____♥_Е
9	4.40	_____♥_Б
10	4.62	_____♥_Я
11	4.71	_____♥
12	5.17	_____♥_____♥_____♥
13	5.93	_____♥_____♥_____♥_____♥
14	6.34	_____♥_____♥_____♥_____♥_____♥
15	6.79	_____♥_____♥_____♥_____♥_____♥_____♥

```

16 | 6.85 |
17 | 7.30 | Л_Е_Т_О
18 | 7.39 |
19 | 7.57 |
20 | 7.89 |
21 | 8.16 |
22 | 8.82 |
23 | 9.86 |

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
binary search result:

4.40 row: 9      key: 4.40      data:
6.34 row: 14     key: 6.34      data:
2.08 Not found
1.87 row: 4      key: 1.87      data:
7.39 row: 18     key: 7.39      data:
9.86 row: 23     key: 9.86      data:
5.91 Not found

Program menu:
1) - Print
2) - Binary search
3) - Cocktail_sort
4) - Mix table
5) - Reverse
6) - Exit

Select action:

Mix table result:

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| # | key | Data |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

0 | 0.94 |
1 | 4.71 |
2 | 4.62 |
3 | 1.08 |
4 | 1.87 |
5 | 9.86 |
6 | 7.39 |
7 | 2.30 |
8 | 3.98 |
9 | 4.40 |
10 | 0.04 |
11 | 5.93 |
12 | 7.30 | Л_Е_Т_О
13 | 6.85 |
14 | 2.07 |
15 | 5.17 |
16 | 6.34 |
17 | 7.89 |
18 | 3.44 |
19 | 8.82 |
20 | 0.39 |
21 | 7.57 |
22 | 8.16 |
23 | 6.79 |

```

## ЗАКЛЮЧЕНИЕ

Данная программа, составлена в соответствии с постановкой задачи на курсовое проектирование по теме "Сортировка и поиск" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". При написании программы использованы методические указания по курсовому проектированию по дисциплине. Для проверки работоспособности программы и правильности обработки входных данных разработан тестовый пример. Тестирование программы подтвердило, что программа правильно выполняет обработку данных и выдаёт верные результаты. Всё это свидетельствует о работоспособности программы и позволяет сделать вывод о пригодности программы к решению практических задач по обработке таблиц.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAXLEN 100

typedef struct {
    float key;
} key_t;

typedef struct {
    char data[MAXLEN];
} data_t;

int menu();

void printTable(key_t *k, data_t *v, int size);
void binarysearch(key_t*, data_t*, int, float);
void cocktail_sort(key_t*, data_t*, int);
void reverse(key_t*, data_t*, int);
void exchange(key_t*, data_t*, int, int);
void mix(key_t*, data_t*, int);
int wild(int, int);
int is_sorted(key_t*, int);
int match(float, float);

int main(int argc, char *argv[]) {
    FILE *pfile = NULL;
    char *fname;
    size_t cnt = 0;
    float a;
    int action;

    if (argc == 2) {
        fname = argv[1];
    } else {
        printf("Invalid number of parameters\n");
        exit(1);
    }
    //Open the file for reading
    if(!(pfile = fopen(fname, "r"))) {
        printf("Error opening %s for reading. Program terminated.\n", fname);
        exit(1);
    } else printf("File %s open for reading...\n", fname);

    if (fscanf(pfile, "%zu", &cnt)!=1) {
        printf("Error! Invalid input format! No number of rows specified. Program terminated.\n");
        exit(1);
    }
    printf("\n%3lu rows:\n\n",cnt);

    key_t keys[cnt];
    data_t str[cnt];

    int i = 0;
    while (i < cnt) {
        if (fscanf(pfile, "%f", &keys[i].key)!=1) {
            printf("Error! Invalid input format key. Program terminated.\n");
            exit(1);
        }
        if (fscanf(pfile, "%100s", str[i].data)!=1) {
            printf("Error! Invalid input format str! Program terminated.\n");
            exit(1);
        }
        i++;
    }
    // print то что считали из файла
    printTable(keys, str, cnt);
    printf("\n");

    //    binarysearch
    printf("binary search result:\n\n");
    while (fscanf(pfile, "%f", &a)==1) {
        printf("%.2f\t",a);
        binarysearch(keys, str, cnt, a);
    }
}
```

```

    }
    printf("\n");
    fclose(pfile); // Close the file

do {
    action = menu();
    switch (action) {
        case 1:
            printTable(keys, str, cnt);
            break;
        case 2:
            if (!is_sorted(keys, cnt)) {
                cocktail_sort(keys, str, cnt);
            }
            printTable(keys, str, cnt);
            float s_key;
            printf("Enter key: ");
            scanf("%f", &s_key);
            printf("\nBinary search result:\n\n");
            printf("%.2f\t", s_key);
            binarysearch(keys, str, cnt, s_key);
            printf("\n");
            break;
        case 3:
            printTable(keys, str, cnt);
            printf("\nCocktail sort result:\n\n");
            cocktail_sort(keys, str, cnt);
            printTable(keys, str, cnt);
            break;
        case 4:
            printTable(keys, str, cnt);
            printf("\nMix table result:\n\n");
            mix(keys, str, cnt);
            printTable(keys, str, cnt);
            break;
        case 5:
            printTable(keys, str, cnt);
            printf("\nReverse table result:\n\n");
            reverse(keys, str, cnt);
            printTable(keys, str, cnt);
            break;
        case 6: break;
        default:
            printf("Invalide menu action!\n");
            break;
    }
} while (action != 6);
return 0;
}

// ВЫВОДИТ МЕНЮ
int menu() {
    int action;
    printf("Program menu:\n");
    printf("1) - Print\n");
    printf("2) - Binary search\n");
    printf("3) - Cocktail_sort\n");
    printf("4) - Mix table\n");
    printf("5) - Reverse\n");
    printf("6) - Exit\n");
    printf("\nSelect action: ");
    scanf("%d", &action);
    return action;
}

// print table
void printTable(key_t *k, data_t *v, int size){
    printf("+-----+-----+-----+-----+-----+\n");
    printf("| # | key | Data | \n");
    printf("+-----+-----+-----+-----+-----+\n\n");
    for (int i = 0; i < size; i++) {
        printf("| %2d | %.2f | %-*s \n", i, k[i].key, (int)((MAXLEN-
strlen(v[i].data))/2+strlen(v[i].data)), v[i].data);
    }
    printf("\n+-----+-----+-----+-----+-----+\n");
}

// бинарный поиск по ключу таблицы
void binarysearch(key_t *k, data_t *v, int size, float key) {
    if (!size) printf("Error size!");
    int test = 0;

```



```

int low, high, middle;
low = 0;
high = size - 1;
while (low <= high) {
    middle = (low + high)/2;
    if (key < k[middle].key) {
        high = middle - 1;
    } else if (key > k[middle].key) {
        low = middle + 1;
    } else {
        test = 1;
        break;
    }
}
if (test) {
    printf("row: %-3d\t", middle);
    printf("key: %-.2f\t", k[middle].key);
    printf("data: %-*s\n", (int)((MAXLEN-
strlen(v[middle].data))/2+strlen(v[middle].data)), v[middle].data);
} else printf("Not found\n");
}

// Шейкерная сортировка
void cocktail_sort(key_t *k, data_t *v, int size) {
    int flag = 1; // флаг наличия перемещений
    int left = 0;
    int right = size - 1; // левая и правая границы сортируемой области массива
    // Выполнение цикла пока левая граница не сомкнется с правой или пока в массиве имеются перемещения
    while ((left < right) && flag > 0) {
        flag = 0;
        for (int i = left; i < right; i++) {
            //двигаемся слева направо
            if (k[i].key > k[i+1].key) {
                // если следующий элемент меньше текущего, меняем их местами
                exchange(k, v, i+1, i);
                flag = 1; // перемещения в этом цикле были
            }
        }
        right--; // сдвигаем правую границу на предыдущий элемент
        for (int i = right; i > left; i--) {
            //двигаемся справа налево
            if (k[i-1].key > k[i].key) {
                // если предыдущий элемент больше текущего, меняем их местами
                exchange(k, v, i, i-1);
                flag = 1; // перемещения в этом цикле были
            }
        }
        left++; // сдвигаем левую границу на следующий элемент
    }
}

// меняет местами k[a] k[b] и v[a] b v[b]
void exchange(key_t *k, data_t *v, int a, int b) {
    key_t tmpkey_t;
    data_t tmpdata_t;

    tmpkey_t = k[a];
    k[a] = k[b];
    k[b] = tmpkey_t;

    tmpdata_t = v[a];
    v[a] = v[b];
    v[b] = tmpdata_t;
}

// Перемешивание строк таблицы в псевдослучайном порядке
void mix(key_t *k, data_t *v, int size) {
    int i, j;
    srand((size_t)time(0));
    for (int z = 0; z < size; z++) {
        i = wild(0, size-1);
        j = wild(0, size-1);
        exchange(k, v, i, j);
    }
}

// возвращает "очень" случайное число от 0 до size для mix
int wild(int a, int b) {
    return a + rand() % (b - a + 1);
}

```

```

// расставляет элементы таблицы в обратном порядке
void reverse(key_t *k, data_t *v, int size) {
    int i, j;
    for (i = 0, j = size-1; i < j; i++, j--) {
        exchange(k, v, i, j);
    }
}

// проверяет отсортирована ли таблица по возрастанию
int is_sorted(key_t *k, int size){
    for (int i = 0; i < size; i++) {
        if (!match(k[i].key, k[i + 1].key)) {
            return 0;
        }
    }
    return 1;
}

// сравнивает значения ключей для is_sorted
int match(float k1, float k2) {
    return k2 >= k1;
}

```