

Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

на тему «База данных по странам мира»

Выполнил:

Студент группы 19ВВ4

Хлыстов А.Ю.

Принял:

Митрохин М.А.

Пенза 2020

Реализовать программу для работы с данными, хранящимися в файлах формата CSV. Программа должна реализовывать функции чтения файла, сортировки, добавления и удаления записей, поиска записей, вывода их в консоль и записи данных в файл. Программа должна быть многомодульной, написана на языке программирования Си и включать код на языке Assembler. Должны использоваться сложные типы данных – структуры, списки, файлы.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Блок-схема основного алгоритма программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

1. Разработка архитектуры программы — 05.03.2020

2. Реализация пользовательского интерфейса — 20.03.2020

3. Разработка основных механик игры — 20.04.2020

4. Добавление дополнительных механик игры — 15.05.2020

5. Оформление отчёта — 20.05.2020

Дата выдачи задания “ ”

Дата защиты проекта “ ”

Руководитель Митрохин М.А.

Задание получил “ 03 ” марта 2020 г.

Студент Хлыстов Артём Юрьевич

Оглавление

Введение.	5
Постановка задачи	6
Выбор решения	6
Описание алгоритма программы.....	7
Определение необходимых модулей программы.....	17
Руководство пользователя.	19
Отладка и тестирование	31
Заключение.	32
Список литературы.	32
Приложение А. Листинг Программы.	33

Введение.

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделённых запятыми.

Формат CSV стандартизирован не полностью. Идея использовать запятые для разделения полей очевидна, но при таком подходе возникают проблемы, если исходные табличные данные содержат запятые или переводы строк. Возможным решением проблемы запятых и переносов строк является заключение данных в кавычки, однако исходные данные могут содержать кавычки. Помимо этого термином "CSV" могут обозначаться похожие форматы, в которых разделителем является символ табуляции (TSV) или точка с запятой. Многие приложения, которые работают с форматом CSV, позволяют выбирать символ разделителя и символ кавычек.

Спецификация

- Каждая строка файла — это одна строка таблицы.
- Разделителем (англ. delimiter) значений колонок является символ запятой (.). Однако на практике часто используются другие разделители, то есть формат путают с DSVruen и TSV (см. ниже).
- Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("). Если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд.

Стандартизация

Название формата - CSV (значения, разделённые запятыми) - указывает что для разделения полей используется запятая. Тем не менее термин "CSV" может использоваться для обозначения схожих форматов. Много программ понимают под CSV более общий формат DSVruen (англ. delimiter-separated values — значения, разделённые разделителем), допускающий использование иных символов в качестве разделителя. В частности, в русской и других локалях запятая по умолчанию зарезервирована под десятичный разделитель. Поэтому как разделитель используется точка с запятой или табуляция (формат TSV). Другие отличия родственных форматов: используемые кавычки: двойные или одинарные; используемый набор символов; использование первой строки как заголовка. Это значительно затрудняет перенос данных из одних программ в другие, несмотря на всю простоту реализации поддержки CSV.

Стандарт RFC 4180

Существует RFC 4180, предназначенный для стандартизации и упрощения обмена данными в формате CSV. Он предъявляет, в частности, такие требования:

- окончание строк в стиле MS-DOS, символы (CR / LF).
- необязательная строка заголовка.
- каждая строка (запись) должна содержать одинаковое количество полей.
- поле может быть заключено в двойные кавычки.
- если поле содержит запятые, переносы строк, двойные кавычки, то это поле должно быть заключено в двойные кавычки. Если этого не сделать, то данные невозможно будет корректно обработать.
- символ двойной кавычки в поле должен быть удвоен.
- Вместе с тем стандарт рекомендует быть либеральным в отношении данных, получаемых от сторонних источников, при обработке CSV.

Постановка задачи

Требуется разработать программу, которая занесёт данные из файла (CSV) в структуру данных в программе и реализует функции сортировки, поиска и т.п.

Исходный файл должен находиться в кодировке ANSI и выполнять требования стандарта.

Устройство ввода – клавиатура.

Выбор решения

Для написания данной программы использован язык программирования Си и Assembler. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Описание алгоритма программы

Для хранения данных таблицы, дабы обеспечить возможность хранить неограниченное количество строк и столбцов, используется структура следующего типа:

За строку таблицы отвечает структура “line”

```
typedef struct line {  
    char* str;  
    cell* mesh;  
    line* ptr_line;  
}line;
```

“str” – строка целиком. Она выполняет функцию резервной копии данных, а также обеспечивает безопасность на случай прямого изменения значений ячеек.

“ptr_line” – указатель на следующую строку таблицы.

“mesh” – указатель на первый столбец этой строки.

“mesh” представляет собой указатель на начало следующей структуры:

```
typedef struct cell {  
    char* value;  
    struct cell* ptr_cell;  
}cell;
```

“value” – значение ячейки.

“ptr_cell” – указатель на следующий столбец.

Таки образом, структура представляет собой связный список, сложенный в элемент другого связного списка.

В начале программы необходимо указать путь к файлу, из которого будут считываться данные и разделитель, который используется в файле. Считывание происходит по следующему алгоритму:

Создаётся строка, под неё выделяется количество памяти, заданное заранее в переменной “size”. Далее запускается цикл *while*, условием выполнения которого является конструкция “!feof(data)”. Функция feof проверяет, достигнут ли конец файла, связанного с потоком. Возвращается значение, отличное от нуля, если конец файла был действительно достигнут.

В начале каждой итерации в переменную “str”, с помощью функции *fgets*, записывается строка из файла. В случае, если строка пустая, будет выведено соответствующее сообщение и программа перейдёт к следующей строке.

Для того, чтобы убедиться, что была считана вся строка, используется функция “check_memory”.

```
int check_memory(FILE* data) {
    int pos = ftell(data);
    int offset = pos - (size - 1);
    int len = strlen(str);
    if (size == len + 1) {
        fseek(data, offset, SEEK_SET);
        size *= 2;
        if ((str = (char*)realloc(str, size)) == NULL) {
            no_memory();
        }
        return 1;
    }
    return 0;
}
```

В переменную “pos” записывается количество байт (от начала) до текущего положения указателя в потоке данных. В переменную “offset” записывается количество байт до начала записываемой строки. Если длина строки равна переменной “size”, которая отвечает за память, выделяемую под строку, можно полагать, что строка была считана не полностью. В этом случае, с помощью функции *fseek*, указатель в потоке данных ставится на начало строки, переменная “size” увеличивается вдвое, и под строку “str” выделяется уже новое количество памяти и функция возвращает «1».

Если длина строки не равна переменной “size”, функция возвращает «0».

Данная функция используется в конструкции:

```
if (check_memory(data)) {
    continue;
}
```

Если функция возвращает «0», входа в конструкцию *if* не происходит и программа продолжает своё выполнение. Если «1», программа сдвинет положение указателя на начало строки, и перезапишет её. Конструкция выполняется до тех пор, пока не будет выделено достаточное количество памяти.

Далее из строки удаляется символ переноса «\n»:

```
str[strcspn(str, "\n")] = 0;
```

С помощью функции *strcspn* получаем позицию символа в строке, и заменяем его на нуль-терминатор.

Затем, с помощью функции “create_line” создаётся «строка» таблицы:

```
sample = create_line();
```

```

line* create_line() {
    line* sample = (line*)calloc(1, sizeof(line));
    sample->ptr_line = NULL;
    sample->mesh = create_cell();
    return sample;
}

```

Эта функция выделяет в памяти место под структуру типа “line”, присваивает указателю значение NULL. Дополнительно она использует функцию “create_cell”:

```

cell* create_cell() {
    cell* sample = (cell*)calloc(1, sizeof(cell));
    sample->ptr_cell = NULL;
    return sample;
}

```

Она создаёт «первый столбец» строки, выделяя под неё память и присваивает указателю значение NULL.

Далее, под резервную копию строки в структуре “line” выделяется необходимое количество памяти и в неё копируется содержимое “str”:

```

sample->str = (char*)malloc(strlen(str)+1);
sample->str = strcpy(sample->str, str);

```

Следующим пунктом в обработке строки идёт её разбиение по разделителю. На начало исходной строки “str” ставится указатель “ppstr”. Далее работа идёт с ним. Для разбиения строки используются функции “split” и “proc_quote”. Функция “split” используется для обычного деления строки, “proc_quote” – обрабатывает случай, когда в лексеме содержатся знаки разделителя, переносы строк, кавычки. В таком случае лексема должна быть закрыта с обеих сторон символом двойной кавычки.

Для корректного определения, какую функцию нужно использовать в конкретный момент используется функция “correct_split”:

```

char* correct_split(char* sep) {
    char* istr;
    if (ppstr[0] == '"') {
        istr = proc_quote(sep);
    }
    else {
        istr = split(sep);
    }
    return istr;
}

```

Если первым символом в строке идёт знак двойной кавычки – вызывается функция “proc_quote”, иначе – “split”.

Принцип работы этих функций:

```

char* split(char* sep) {
    char* bef = (char*)malloc(1);

```

```

int a = 1;
int comma = 0;
if (ppstr[0] == '\\0') {
    return NULL;
}
comma = strcspn(ppstr, sep);
if ((bef = (char*)realloc(bef, comma)) == NULL) {
    no_memory();
}
memset(bef, 0, comma);
if (comma >= strlen(ppstr)) {
    strncpy(bef, ppstr, comma);
    ppstr = ppstr + comma;
    bef[comma] = '\\0';
    return bef;
}
strncpy(bef, ppstr, comma);
bef[comma] = '\\0';
ppstr = ppstr + comma + 1;
return bef;
}

```

bef – лексема, строка от начала и до первого символа разделителя.

В переменную “comma” записывается позиция символа разделителя в строке. Под строку “bef” выделяется нужное количество памяти и в неё копируется *comma* символов из строки “ppstr”. Следующим символом “bef” записывается нуль-терминатор. Указатель “ppstr” смещается «за запятую», таким образом отбрасывая уже считанное значение.

```

char* proc_quote(char* sep) {
    char* ptr;
    char* bef = (char*)malloc(1);
    char* proc_sep = (char*)calloc(1, 2);
    int s = 1;
    int comma = 0;
    bef = (char*)malloc(1);
    ppstr++;
    if (ppstr[0] == '\"') {
        ppstr += 2;
        bef = (char*)realloc(bef, (strlen(none)));
        memset(bef, 0, strlen(none));
        strcpy(bef, none);
        return bef;
    }
    proc_sep[0] = '\"';
    strcat(proc_sep, sep);
    if (ppstr[0] == '\\0') {
        return NULL;
    }
    ptr = strstr(ppstr, proc_sep);
    if (ptr == NULL) {
        comma = strlen(ppstr);
        if ((bef = (char*)realloc(bef, comma)) == NULL) {
            no_memory();
        }
        strncpy(bef, ppstr, comma);
        bef[comma - 1] = '\\0';
        ppstr = ppstr + comma;
        return bef;
    }
}

```

```

    }
    comma = strlen(ppstr) - strlen(ptr);
    if ((bef = (char*)realloc(bef, comma)) == NULL) {
        no_memory();
    }
    memset(bef, 0, comma);
    if (comma >= strlen(ppstr)) {
        strncpy(bef, ppstr, comma);
        ppstr = ppstr + comma;
        return bef;
    }
    strncpy(bef, ppstr, comma);
    bef[comma] = '\\0';
    ppstr = ppstr + comma + 2;
    return bef;
}

```

`bef` – лексема, строка от начала и до первого символа разделителя.

`proc_sep` – специальная строка разделитель для данного случая, собирается на основе символа двойной кавычки и символа разделителя.

Определив символ двойной кавычки в начале строки, указатель “`ppstr`” сдвигается на следующий символ в строке. В случае если им опять же является символ двойной кавычки (это говорит о том, что строка является пустой, т.е. в ней отсутствует значение), в переменную `bef` записывается ‘None’.

С помощью функции *strstr* в переменную “`ptr`” записывается указатель на первое вхождение подстроки разделителя в строку “`ppstr`”. Положение запятой получается как разность длин строк “`ppstr`” и “`ptr`”.

Под строку “`bef`” выделяется нужное количество памяти и в неё копируется *comma* символов из строки “`ppstr`”. Следующим символом “`bef`” записывается нуль-терминатор. Указатель “`ppstr`” смещается за разделители, таким образом отбрасывая уже считанное значение.

Данные функции используются в конструкции:

```

while ((istr = correct_split(sep)) != NULL) {
    p_cell = sample_cell;
    sample_cell = create_cell();
    p_cell->ptr_cell = sample_cell;
    sample_cell->value = istr;
}

```

С помощью функции “`create_cell`” создаётся «ячейка», и в неё записывается значение, возвращаемое функциями. Также происходит переопределение указателей.

Она будет выполняться пока не будет достигнут конец строки (в этом случае функция “correct_split” вернёт NULL).

Всё вышеописанное повторяется до тех пор, пока не достигнут конец файла.

Код функции:

```
line* read_csv(FILE* data, char* sep)
{
    line* header = NULL;
    line* sample = NULL;
    line* p;
    cell* sample_cell = NULL;
    cell* p_cell;
    char* istr;
    int counter_line = 0;
    str = (char*)malloc(size);
    if ((str = (char*)malloc(size)) == NULL) {
        no_memory();
    }
    memset(str, 0, size);
    while (!feof(data)) {
        if ((str = fgets(str, size, data)) == NULL) {
            printf("Строка пустая.");
            continue;
        }
        if (check_memory(data)) {
            continue;
        }
        str[strcspn(str, "\n")] = 0;
        pptr = str;
        counter_line++;
        if (counter_line == 1) {
            header = create_line();
            sample = header;
        }
        else {
            p = sample;
            sample = create_line();
            p->ptr_line = sample;
        }
        sample->str = (char*)malloc(strlen(str)+1);
        sample->str = strcpy(sample->str, str);
        istr = correct_split(sep);
        sample->mesh->value = istr;
        sample_cell = sample->mesh;
        while ((istr = correct_split(sep)) != NULL) {
            p_cell = sample_cell;
            sample_cell = create_cell();
            p_cell->ptr_cell = sample_cell;
            sample_cell->value = istr;
        }
        size = BUFF;
    }
    free(str);
    if (header == NULL) {
        return NULL;
    }
    return header;
}
```

Функция сортировки:

Алгоритм сортировки, используемый в программе – сортировка выбором (selection sort). В общем виде алгоритм выглядит так:

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Сортировка в программе реализована с помощью функции *sort*.

Аргументами функции являются “header” – указатель на заголовок сортируемой таблицы, “by” – номер столбца, по которому будет произведена сортировка, “ascending” – булево значение, соответствующее возрастающему порядку списка.

В начале инициализируются две переменные:

```
line* end_s = header;  
line* begin_ns = end_s->ptr_line;
```

“end_s” – последний элемент отсортированного списка.

“begin_ns” – первый элемент неотсортированного списка.

Изначально отсортированный список пуст, началом неотсортированного списка является первый элемент исходной таблицы.

Функция разделяется на две ветви в соответствии с “ascending”:

```
if (ascending)
```

Далее запускается цикл:

```
while (begin_ns != NULL)
```

Условием его окончания является то, что больше не осталось неотсортированных элементов.

Внутри этого цикла запускается ещё один цикл:

```
while (p != NULL)
```

Он отвечает за очередной проход по неотсортированным элементам списка, с целью нахождения экстремума.

По окончании цикла, в переменной “ext” будет записан очередной минимум неотсортированного списка.

Далее необходимо поставить этот элемент в конец отсортированного списка.

```
end_s->ptr_line = ext;  
end_s = ext;
```

А затем изменить начало неотсортированного:

```
begin_ns = end_s->ptr_line;
```

В функции сортировки задействована дополнительная функция *type_column*.

```
int type_column(line* header, int n) {  
    line* p = header->ptr_line;  
    int b = n;  
    cell* pc = p->mesh;  
    for (b; b > 1; b--) {  
        pc = pc->ptr_cell;  
    }  
    while ((p->ptr_line != NULL) & (char_to_digit(pc->value) != NULL)) {  
        pc = p->mesh;  
        for (b; b > 1; b--) {  
            pc = pc->ptr_cell;  
        }  
        p = p->ptr_line;  
        b = n;  
    }  
    b = n;  
    if (p->ptr_line == NULL) {  
        pc = p->mesh;  
        for (b; b > 1; b--) {  
            pc = pc->ptr_cell;  
        }  
        if (char_to_digit(pc->value) != NULL) {  
            return 1;  
        }  
    }  
    return 0;  
}
```

Так как невозможно сортировать числовые значения как текстовые, используется данная функция: она определяет «тип столбца». Если все значения столбца являются числом, функция возвращает «1», что свидетельствует о том что столбец является числовым. В ином случае – возвращается «0».

Полный код функции *sort*:

```
line* sort(line* header, int by, bool ascending) {  
    // selection sort  
    line* p = header->ptr_line;  
    line* end_s = header;  
    line* begin_ns = end_s->ptr_line;  
    line* ext = p;
```

```

line* before_min = p;
line* before = p;
cell* pc1 = p->mesh;
cell* pc2 = ext->mesh;
int i = 1;
int numeric = type_column(header, by);
if (ascending) {
    while (begin_ns != NULL) {
        pc2 = ext->mesh;
        for (int j = by; j > 1; j--) {
            pc2 = pc2->ptr_cell;
        }
        while (p != NULL) {
            pc2 = ext->mesh;
            for (int j = by; j > 1; j--) {
                pc2 = pc2->ptr_cell;
            }
            pc1 = p->mesh;
            for (int j = by; j > 1; j--) {
                pc1 = pc1->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc1->value) <= (char_to_digit(pc2-
>value)))) {
                    ext = p;
                    before_min = before;
                }
            }
            else {
                if ((strcmp(pc1->value, pc2->value)) <= 0) {
                    ext = p;
                    before_min = before;
                }
            }
            if (i) {
                i = 0;
                p = p->ptr_line;
                continue;
            }
            before = before->ptr_line;
            p = p->ptr_line;
        }
        i = 1;
        if (ext == before_min) {
            begin_ns = ext->ptr_line;
            end_s = ext;
        }
        else {
            before_min->ptr_line = ext->ptr_line;
            ext->ptr_line = end_s->ptr_line;
            end_s->ptr_line = ext;
            end_s = ext;
            begin_ns = end_s->ptr_line;
        }
        p = begin_ns;
        before = p;
        ext = p;
    }
}
else {
    while (begin_ns != NULL) {
        pc2 = ext->mesh;
        for (int j = by; j > 1; j--) {
            pc2 = pc2->ptr_cell;

```

```

    }
    while (p != NULL) {
        pc2 = ext->mesh;
        for (int j = by; j > 1; j--) {
            pc2 = pc2->ptr_cell;
        }
        pc1 = p->mesh;
        for (int j = by; j > 1; j--) {
            pc1 = pc1->ptr_cell;
        }
        if (numeric) {
            if ((char_to_digit(pc1->value) >= (char_to_digit(pc2->value)))) {
                ext = p;
                before_min = before;
            }
        }
        else {
            if ((strcmp(pc1->value, pc2->value)) >= 0) {
                ext = p;
                before_min = before;
            }
        }
        if (i) {
            i = 0;
            p = p->ptr_line;
            continue;
        }
        before = before->ptr_line;
        p = p->ptr_line;
    }
    i = 1;
    if (ext == before_min) {
        begin_ns = ext->ptr_line;
        end_s = ext;
    }
    else {
        before_min->ptr_line = ext->ptr_line;
        ext->ptr_line = end_s->ptr_line;
        end_s->ptr_line = ext;
        end_s = ext;
        begin_ns = end_s->ptr_line;
    }
    p = begin_ns;
    before = p;
    ext = p;
}

}
return header;
}

```

Определение необходимых модулей программы.

Программа состоит из двух модулей:

“Kursach.cpp” – тестовая программа, реализующая возможности библиотеки.

“Play_csv.cpp” – библиотека, содержащая функции для работы с CSV-файлами.

Все функции “Play_csv.cpp” можно разбить на несколько блоков:

Блок "FILE"

read_csv – читает файл и записывает данные таблицы в структуру.

read_csv_without_header – читает файл и записывает данные в структуру (для таблиц, у которых отсутствует заголовок). *to_csv* – записывает таблицу, используемую в программе в файл.

Блок "INFO"

shape - возвращает в структуре типа 'shape_struct' количество строк и столбцов таблицы.

```
typedef struct shape_struct {  
    int i;  
    int j;  
}shape_struct;
```

Где I – количество строк, J – количество столбцов.

info – выводит данные о таблице (заголовочная строка, количество строк и столбцов).

Блок "FIND"

find_cell – возвращает указатель на ячейку, индексы которой указаны в аргументах, *find_line* – соответственно возвращает указатель на строку.

loc – возвращает значение ячейки, указатель на которую передан в аргументе.

index – возвращает массив номеров строк, удовлетворяющих логическому условию.

Блок "PRINT"

print - Выводит в консоль всю таблицу.

print_head - Выводит в консоль 'i' записей (строк) от начала.

print_tail - Выводит в консоль 'i' записей (строк) от конца.

print_slice - Выводит в консоль записи (строки) с 'i' по 'j' номер.

Print_index - Выводит в консоль записи, индексы которых содержатся в массиве индексов.

print_solo - Выводит единичную запись в консоль.

Блок "SORT"

sort – сортирует список по указанному столбцу, *sort_in* – записывает отсортированную версию исходного списка в новую таблицу, *print_sort* – выводит отсортированный список, не изменяя исходный.

Блок "EDIT"

remov – удаляет строку из списка, *append* – добавляет строку в список, *edit* – редактирует ячейку таблицы, *renam* - меняет заголовок таблицы.

Блок "MATH"

median – возвращает указатель на «средний элемент» таблицы, *mean* – возвращает среднее значение по столбцу, *maximit* – возвращает указатель на строку с максимальным значением по столбцу, *minimit* – возвращает указатель на строку с минимальным значением по столбцу.

Вспомогательные функции

char_to_digit – переводит строковое значение в числовое (если это возможно).

copy – возвращает копию строки.

На рисунке 1 представлена схема данных:

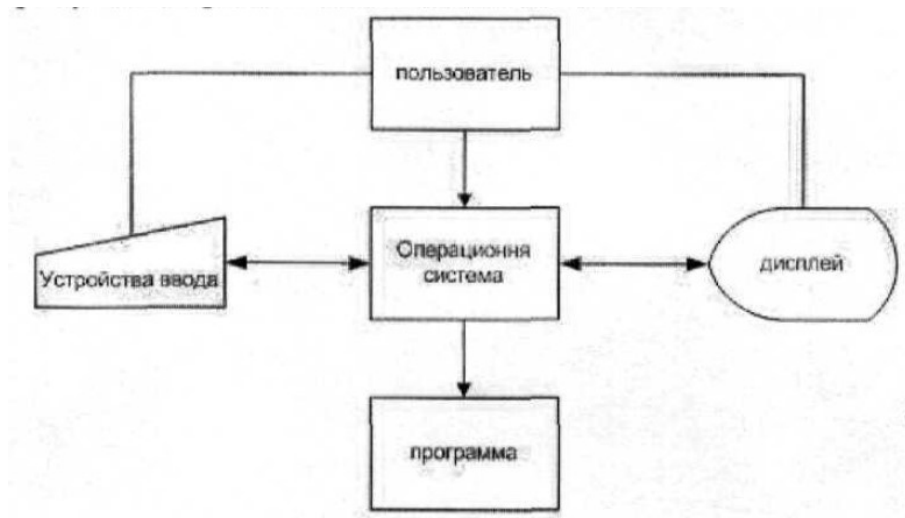


Рисунок 1 – схема данных

Руководство пользователя.

Kursach.cpp – тестовая программа, реализующая возможности библиотеки.

При запуске программы выводится приветствие и ожидается отклик пользователя – нажатие клавиши Enter. (рис. 2)

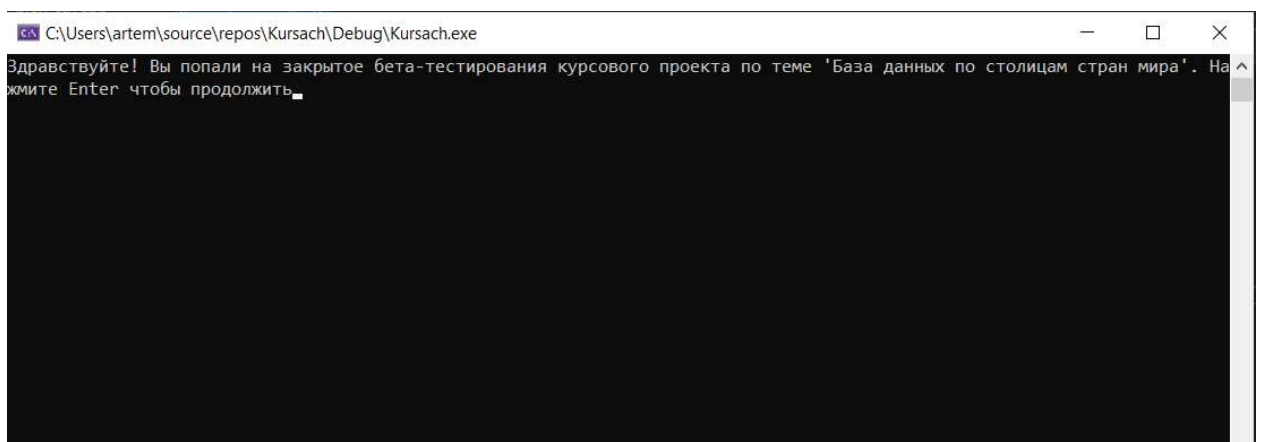


Рисунок 2 – запуск программы.

Далее необходимо указать путь к файлу.

В случае, если путь указан верно, файл будет открыт. Пользователю будет предложено указать знак разделителя, используемого в файле, а также наличие заголовка в файле. (рис. 3)

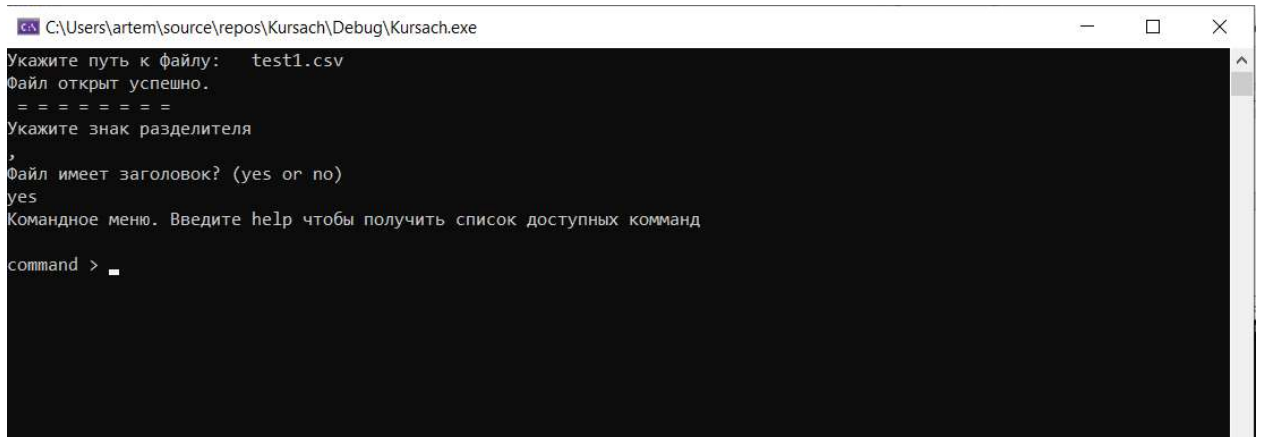


Рисунок 3 – заполнение.

В случае неправильного указанного пути будет выведено соответствующее сообщение, и программа попросит ввести путь ещё раз. (рис. 4)

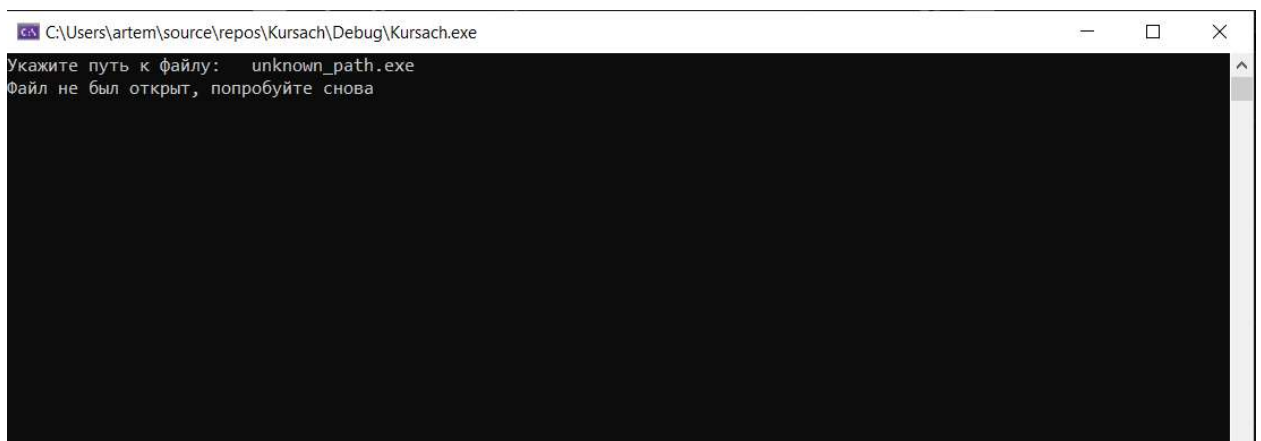


Рисунок 4 – неправильный ввод пути.

Если всё выполнено верно, будет открыто командное меню.

Меню представляет собой командный интерфейс, с последующим диалоговым выбором функций для исполнения. Команда, введённая пользователем, подвергается обработке:

- Все символы приводятся к нижнему регистру.
- Убираются все знаки пробела.

Список доступных команд:

command > print - вызывает блок "PRINT" и предоставляет пользователю выбор режима вывода таблицы.

Рисунок 5 – интерфейс меню 'PRINT'

```
command > print
Меню 'PRINT'
Выберите подходящий режим вывода таблицы:
1. Печать всей таблицы
2. Печать n значений с начала таблицы.
3. Печать n значений с конца таблицы.
4. Печать среза таблицы с n по m
0. Выйти из блока 'PRINT'
```

Рисунок 5 – интерфейс меню "PRINT".

1. Печать всей таблицы.

Не требует ввода дополнительных данных. Выводит всю таблицу. (рис. 6)

ID	Страна	Столица	Население(млн.ч.)	Регион
1	Китай	Пекин	1403	Азия
2	Индия	Нопе	1372	Азия
3	США	Вашингтон	331	Северная Америка
4	Бразилия	Бразилиа	211	Южная Америка
5	Россия	Москва	146	Европа
6	Япония	Токио	125	Азия
7	Германия	Берлин	83	Европа
8	Франция	Париж	68	Европа
9	Турция	Анкара	83	Европа
10	Египет	Каир	100	Африка
11	Мексика	Мехико	126	Северная Америка
12	Эфиопия	Аддис-Аббеба	112	Африка
13	Нидерланды	Амстердам	17	Европа

Рисунок 6 – печать всей таблицы.

2. Печать n значений с начала таблицы. Дополнительно необходимо указать n – количество строк с начала таблицы, вывод которых необходим. (рис. 7)

```

2
Введите n.
n = 5

```

h	ID	Страна	Столица	Население(млн.ч.)	Регион
1	1	Китай	Пекин	1403	Азия
2	2	Индия	Нопе	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	5	Россия	Москва	146	Европа
.	.	.	.	.	.

Рисунок 7 – печать с начала.

3. Печать n значений с конца таблицы. Дополнительно необходимо указать n – количество строк с конца таблицы, вывод которых необходим. (рис. 8)

```

3
Введите n.
n = 5

```

h	ID	Страна	Столица	Население(млн.ч.)	Регион
.	.	.	.	.	.
9	9	Турция	Анкара	83	Европа
10	10	Египет	Каир	100	Африка
11	11	Мексика	Мехико	126	Северная Америка
12	12	Эфиопия	Аддис-Аббеба	112	Африка
13	13	Нидерланды	Амстердам	17	Европа

Рисунок 8 – печать с конца.

4. Печать среза таблицы с n по m. Дополнительно необходимо указать два значения: n – строка таблицы, с которой необходимо вывести таблицу, m – строка, до которой необходимо вывести таблицу (рис. 9)

```

4
Введите n.
n = 2
Введите m.
m = 8

```

h	ID	Страна	Столица	Население(млн.ч.)	Регион
.	.	.	.	.	.
2	2	Индия	Нопе	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	5	Россия	Москва	146	Европа
6	6	Япония	Токио	125	Азия
7	7	Германия	Берлин	83	Европа
.	.	.	.	.	.

Рисунок 9 – печать среза.

command > math - вызывает блок "MATH" и предоставляет пользователю выбор математической функции.

Рисунок 10 – интерфейс меню 'MATH'

```
command > math
Меню 'MATH'
Выберите необходимую функцию:
1. Медиана. (Предупреждение: Функцию 'медиана' рационально использовать ТОЛЬКО на отсортированный список.
Внимание: Следующие функции используются только для столбцов, содержащих ЧИСЛОВЫЕ ЗНАЧЕНИЯ.
2. Среднее значение.
3. Максимум.
4. Минимум
0. Выйти из блока 'MATH'
```

Рисунок 10 – интерфейс меню 'MATH'

1. Медиана. Выводит строку, находящуюся с середине таблицы. (рис. 11)

```
1
Медиана:
7                Германия                Берлин                83                Европа
```

Рисунок 11 – медиана.

2. Среднее значение. Вывод среднее значение всех строк по столбцу, номер которого необходимо указать дополнительно. (рис. 12)

```
2
Укажите номер столбца:
i = 4
Среднее значение: 321,307692
```

Рисунок 12 – среднее значение.

3. Максимум. Выводит строку, значение которой по столбцу, указанному дополнительно, является максимальным. (рис. 13)

```
3
Укажите номер столбца:
i = 4
Запись с максимальным значением:
1                Китай                Пекин                1403                Азия
```

Рисунок 13 – максимум.

4. Минимум. Выводит строку, значение которой по столбцу, указанному дополнительно, является минимальным. (рис. 14)

```
4
Укажите номер столбца:
i = 4
Запись с минимальным значением:
13                Нидерланды                Амстердам                17                Европа
```

Рисунок 14 - минимум

command > sort - вызывает блок "SORT" и предоставляет пользователю выбор режима сортировки списка.(*)

Рисунок 15 – интерфейс меню 'SORT'

```
command > sort
Меню 'SORT'
Выберите необходимую режим сортировки:
1. Сортировать таблицу
2. Вывести отсортированный список
0. Выйти из блока 'EDIT'
```

Рисунок 15 – интерфейс меню 'SORT'

1. Сортировать таблицу. Сортирует таблицу по указанному столбцу. Дополнительно необходимо указать в каком порядке необходимо сортировать (yes – в порядке возрастания, no – убывания.) (рис 16-17)

```
1
Укажите номер столбца, по которому следует выполнить сортировку
i = 4
Сортировать по возрастанию? (yes or no)
yes
```

Рисунок 16 – сортировка.

h	ID	Страна	Столица	Население(млн.ч.)	Регион
1	13	Нидерланды	Амстердам	17	Европа
2	8	Франция	Париж	68	Европа
3	9	Турция	Анкара	83	Европа
4	7	Германия	Берлин	83	Европа
5	10	Египет	Каир	100	Африка
6	12	Эфиопия	Аддис-Аббеба	112	Африка
7	6	Япония	Токио	125	Азия
8	11	Мексика	Мехико	126	Северная Америка
9	5	Россия	Москва	146	Европа
10	4	Бразилия	Бразилиа	211	Южная Америка
11	3	США	Вашингтон	331	Северная Америка
12	2	Индия	None	1372	Азия
13	1	Китай	Пекин	1403	Азия

Рисунок 17 – вывод списка.

2. Вывести отсортированный список. Сразу выводит отсортированную версию списка по указанному столбцу, не меняя исходную таблицу. Дополнительно необходимо указать в каком порядке необходимо сортировать (yes – в порядке возрастания, no – убывания.) (рис.18)

```

2
Укажите номер столбца, по которому следует выполнить сортировку
i = 4
Сортировать по возрастанию? (yes or no)
n

```

h	ID	Страна	Столица	Население(млн.ч.)	Регион
1	1	Китай	Пекин	1403	Азия
2	2	Индия	None	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	5	Россия	Москва	146	Европа
6	11	Мексика	Мехико	126	Северная Америка
7	6	Япония	Токио	125	Азия
8	12	Эфиопия	Аддис-Аббеба	112	Африка
9	10	Египет	Каир	100	Африка
10	7	Германия	Берлин	83	Европа
11	9	Турция	Анкара	83	Европа
12	8	Франция	Париж	68	Европа
13	13	Нидерланды	Амстердам	17	Европа

Рисунок 18 – вывод отсортированного списка.

command> edit - вызывает блок "EDIT" и предоставляет пользователю выбор режима редактирования.

Рисунок 19 – интерфейс меню 'EDIT'

```

command > edit
Меню 'EDIT'
Выберите необходимую команду
1. Изменить заголовок таблицы
2. Добавить запись
3. Удалить запись.
4. Редактировать ячейку.
0. Выйти из блока 'EDIT'

```

Рисунок 19 – интерфейс меню 'EDIT'

1. Изменить заголовок таблицы. Изменяет заголовочную строку таблицы на строку, указанную пользователем. Разделитель в этой строке должен соответствовать разделителю, используемому в программе. Количество столбцов также должно совпадать. (рис. 20)

```

1
Введите новую заголовочную строку
:Id country,name country,capital,people,region
Новый заголовок:
Id country          name country          capital          people          region

```

Рисунок 20 – изменение заголовка.

2. Добавить запись. Добавляет запись в конец таблицы. Необходимо заполнить каждый столбец таблицы. (рис. 21-22).

```

2
Введите ID 24

Введите Страна Беларусь

Введите Столица Минск

Введите Население(млн.ч.) 9

Введите Регион Европа

```

Рисунок 21 – добавление новой записи.

h	ID	Страна	Столица	Население(млн.ч.)	Регион
1	1	Китай	Пекин	1403	Азия
2	2	Индия	None	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	5	Россия	Москва	146	Европа
6	6	Япония	Токио	125	Азия
7	7	Германия	Берлин	83	Европа
8	8	Франция	Париж	68	Европа
9	9	Турция	Анкара	83	Европа
10	10	Египет	Каир	100	Африка
11	11	Мексика	Мехико	126	Северная Америка
12	12	Эфиопия	Аддис - Аббеба	112	Африка
13	13	Нидерланды	Амстердам	17	Европа
14	24	Беларусь	Минск	9	Европа

Рисунок 22 – вывод списка с новой записью.

3. Удалить запись. Удаляет запись, номер строки которой указан дополнительно. (рис. 23-24).

```
3
Укажите номер строки:
i = 5
```

Рисунок 23 – удаление записи.

h	ID	Страна	Столица	Население(млн.ч.)	Регион
1	1	Китай	Пекин	1403	Азия
2	2	Индия	None	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	6	Япония	Токио	125	Азия
6	7	Германия	Берлин	83	Европа
7	8	Франция	Париж	68	Европа
8	9	Турция	Анкара	83	Европа
9	10	Египет	Каир	100	Африка
10	11	Мексика	Мехико	126	Северная Америка
11	12	Эфиопия	Аддис-Аббеба	112	Африка
12	13	Нидерланды	Амстердам	17	Европа
13	24	Беларусь	Минск	9	Европа

Рисунок 24 – вывод списка.

4. Редактировать ячейку. Редактирует ячейку, индексы которой указаны дополнительно. Необходимо ввести новое значение ячейки. (рис. 25-26)

```
4
Укажите номер строки:
i = 2
Укажите номер столбца:
j = 3
Редактируемая ячейка: None
Введите новое значение:
Дели
```

Рисунок 25 – редактирование ячейки.

h	ID	Страна	Столица	Население (млн.ч.)	Регион
1	1	Китай	Пекин	1403	Азия
2	2	Индия	Дели	1372	Азия
3	3	США	Вашингтон	331	Северная Америка
4	4	Бразилия	Бразилиа	211	Южная Америка
5	6	Япония	Токио	125	Азия
6	7	Германия	Берлин	83	Европа
7	8	Франция	Париж	68	Европа
8	9	Турция	Анкара	83	Европа
9	10	Египет	Каир	100	Африка
10	11	Мексика	Мехико	126	Северная Америка
11	12	Эфиопия	Аддис-Аббеба	112	Африка
12	13	Нидерланды	Амстердам	17	Европа
13	24	Беларусь	Минск	9	Европа

Рисунок 26 – вывод списка с редактированной ячейкой.

`command > find` - вызывает блок "FIND" и предоставляет пользователю выбор режима поиска значений в списке.

Рисунок 27 – интерфейс меню 'FIND'

```
command > find
Меню 'FIND'
Выберите необходимую функцию
1. Найти строку
2. Найти ячейку
3. Найти по условию
0. Выйти из блока 'FIND'
```

Рисунок 27 – интерфейс меню 'FIND'

1. Найти строку. Выводит строку, номер которой указан дополнительно. (рис. 28)

```
Укажите номер строки
i = 5
6                Япония                Токио                125                Азия
```

Рисунок 28 – поиск строки.

2. Найти ячейку. Выводит ячейку, индексы которой указаны дополнительно. (рис. 29)

```
2
Укажите номер строки
i = 7
Укажите номер столбца
j = 2
Франция
```

Рисунок 29 – поиск ячейки.

3. Найти по условию. Выводит все строки, соответствующие условию, заданным пользователем. (рис. 30)

```
3
Укажите номер столбца
j = 4
Введите значение.
value = 150
Введите режим:
Режимы:
1. ==
2. >
3. <
4. !=
i = 2
1          Китай          Пекин          1403          Азия
2          Индия          Дели          1372          Азия
3          США            Вашингтон      331          Северная Америка
4          Бразилия        Бразилиа      211          Южная Америка
```

Рисунок 30 – поиск по условию

command > info - выводит в консоль заголовочную строку таблицы и её размеры.

Рисунок 31 – интерфейс команды info.

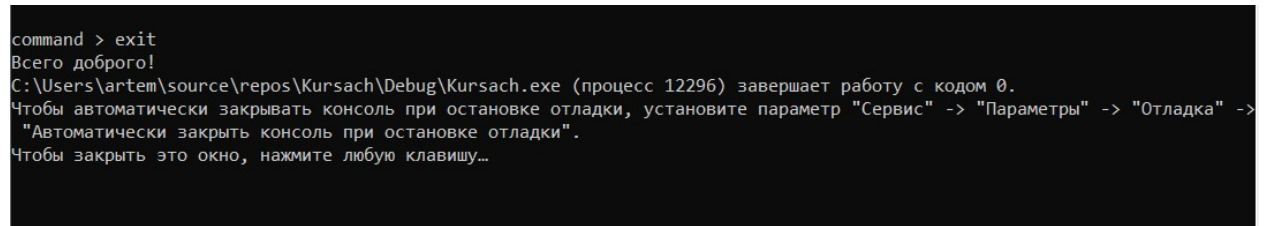
```
command > info
Информация о таблице
Заголовочная строка:
ID,Страна,Столица,Население(млн.ч.),Регион
Размер таблицы:
[5 x 13]
command > _
```

Рисунок 31 – интерфейс команды info.

command > cls - очищает консоль.

command > exit - завершает работу программы.

Рисунок 32 – интерфейс команды exit.



```
command > exit
Всего доброго!
C:\Users\artem\source\repos\Kursach\Debug\Kursach.exe (процесс 12296) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рисунок 32 – интерфейс команды exit.

command > write - записывает таблицу в файл.

Отладка и тестирование

В качестве среды разработки была выбрана программа Visual Studio.

Программа предоставляет все средства необходимые при разработке и отладке разрабатываемых модулей и программ. Для отладки использовались такие инструменты как точка останова, выполнение кода по шагам, анализ содержимого локальных и глобальных переменных, анализ содержимого памяти. Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.

В ходе тестирования было выявлено и исправлено множество проблем связанных с неправильной обработкой строк, исключительными случаями, неправильным выделением памяти и т.п.

Описание теста	Ожидаемый результат.	Полученный результат
Запуск программы.	Программа запустится, откроет и прочтёт нужный файл.	Совпал с ожидаемым.
Командное меню.	Верная интерпретация команд, дальнейшее разветвление.	Совпал с ожидаемым.
Функции сортировки.	Сортировка следуя числовой или текстовой сортировке, правильная перезапись структуры, правильный вывод.	Совпал с ожидаемым.
Функции поиска.	Правильное нахождение строк по индексу или значению.	Совпал с ожидаемым.
Математические функции.	Правильное выполнение математических функций.	Совпал с ожидаемым.
Функции вывода.	Правильный форматированный	Совпал с ожидаемым.

	вывод элементов таблицы.	
Функции редактирования.	Правильное определение нужной строки для редактирования, её изменение.	Совпал с ожидаемым.

Заключение.

Была создана библиотека для работы с файлами формата CSV. Библиотека предоставляет возможности редактирования таблиц для дальнейшего их анализа. Также был создан тестовый пример программы, реализующий пользовательский интерфейс для работы с библиотекой в текстовом режиме. Программа работает корректно, при условии соблюдения пользователем требований к стандарту CSV.

Список литературы.

1. Прата, Стивен .

П70 Язык программирования C . Лекции и упражнения , 5 - е издание . : Пер . с англ . -

М . : Издательский дом "Вильямс" , 2013 . - 960 с . : ил . - Парал . тит. англ .

2. Страуструп, Бьярне.

С83 Язык программирования C++. Краткий курс, 2-е изд. : Пер. с англ. -

СПб.: ООО "Диалектика", 2019. - 320 с.: ил. - Парал. тит. англ.

Приложение А. Листинг Программы.

Kursach.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include "play_csv.h"
#define cls system("cls")
#define Yans (strcmp(StAns, "yes") == 0) || (strcmp(StAns, "ye") == 0) || (strcmp(StAns, "y") == 0)
#define Nans (strcmp(StAns, "no") == 0) || (strcmp(StAns, "n") == 0)
#define clear while (getchar() != '\n');

int START1 = 1;
int START2 = 1;
int START3 = 1;
line* header;

int main() {

    char command[16];
    char* correct_command;
    char path[256];
    char ans[8];
    char value[128];
    int ians;
    int n;
    int m;
    double mean_v;
    char* StAns;
    char head[1024];
    FILE* file;
    int num_case;
    int signal = 0;
    line* ptr;
    cell* cptr;
    int* ind;
    char sep[2] = { 0 };
    char* ptr_sep;
    setlocale(LC_ALL, "Rus");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    //Start
    printf("Здравствуйте! Вы попали на закрытое бета-тестирования курсового проекта по теме 'База данных по столицам стран мира'. Нажмите любую клавишу чтобы продолжить");
    getchar();
    while (START1) {
        cls;
        printf("Укажите путь к файлу: ");
        scanf("%s", path);
        file = fopen(path, "r");
        if (file == NULL) {
            printf("Файл не был открыт, попробуйте снова");
            Sleep(2000);
            continue;
        }
        printf("Файл открыт успешно.\n = = = = = \n");
        clear
        printf("Укажите знак разделителя\n");
        sep[0] = getchar();
        sep[1] = '\0';
        ptr_sep = &sep[0];
        while (START2) {
            clear
```

```

printf("Файл имеет заголовок? (yes or no)\n");
fgets(ans, 7, stdin);
StAns = standart_command(ans);
if (Yans) {
    if ((header = read_csv(file, ptr_sep)) == NULL) {
        continue;
    }
}
else {
    if (Nans) {
        printf("Введите заголовочную строку для вашей
таблицы:\n");

        fgets(head, 1023, stdin);
        if ((header = read_csv_without_header(file, head,
ptr_sep)) == NULL) {
            printf("Произошла ошибка, попробуйте ещё
раз.\n");

            Sleep(1000);
            continue;
        }
    }
    else {
        printf("Ответ не распознан, попробуйте ещё раз.\n");
        continue;
    }
}
fseek(file, 0, SEEK_SET);
fclose(file);
free(StAns);
break;
}
// Menu
printf("Командное меню. Введите help чтобы получить список доступных
команд\n");
while (START3) {
    printf("\ncommand > ");
    fgets(command, 15, stdin);
    correct_command = standart_command(command);
    if ((num_case = get_num(correct_command)) == NULL){
        printf("Команда не распознана, попробуйте ещё раз. Введите
help чтобы получить список доступных команд\n");
        continue;
    }
    switch (num_case) {
        case -2:
            printf("Всего доброго!");
            return 0;
        case -1:
            printf("Список доступных
команд:\nprint\nmath\nedit\nsort\nfind\ninfo\nexit\ncls\nwrite\nПодробнее о командах
можно узнать в документации.\n");
            break;
        case 1:
            printf("Меню 'PRINT'\nВыберите подходящий режим вывода
таблицы:\n");

            printf("1. Печать всей таблицы\n2. Печать n значений с начала
таблицы.\n3. Печать n значений с конца таблицы.\n4. Печать среза таблицы с n по m\n0.
Выйти из блока 'PRINT'\n");
            scanf("%d", &ians);
            clear
            switch (ians) {
                case 1:
                    print(header);
                    break;
                case 2:

```

```

        printf("Введите n.\n n = ");
        scanf("%d", &n);
        print_head(header, n);
        break;
case 3:
    printf("Введите n.\n n = ");
    scanf("%d", &n);
    print_tail(header, n);
    break;
case 4:
    printf("Введите n.\n n = ");
    scanf("%d", &n);
    printf("Введите m.\n m = ");
    scanf("%d", &m);
    print_slice(header, n, m);
    break;
case 0:
    break;
default:
    printf("Ответ не распознан. Попробуйте ещё раз.");
    continue;
}
clear
break;
case 2:
    printf("Меню 'МАТН'\nВыберите необходимую функцию:\n");
    printf("1. Медиана. (Предупреждение: Функцию 'медиана' рационально использовать ТОЛЬКО на отсортированный список.\nВнимание: Следующие функции используются только для столбцов, содержащих ЧИСЛОВЫЕ ЗНАЧЕНИЯ.\n2. Среднее значение.\n3. Максимум.\n4. Минимум\n0. Выйти из блока 'МАТН'\n");
    scanf("%d", &ians);
    switch (ians) {
case 1:
        printf("Медиана:\n\n");
        print_solo(median(header));
        break;
case 2:
        printf("Укажите номер столбца:\ni = ");
        scanf("%d", &n);
        if ((mean_v = mean(header, n)) == NULL) {
            printf("В столбце обнаружено нечисловое значение!\n");
            break;
        }
        printf("Среднее значение: %Lf\n", mean_v);
        break;
case 3:
        printf("Укажите номер столбца:\ni = ");
        scanf("%d", &n);
        if ((ptr = maximum(header, n)) == NULL) {
            printf("В столбце обнаружено нечисловое значение!\n");
            break;
        }
        printf("Запись с максимальным значением:\n\n");
        print_solo(ptr);
        break;
case 4:
        printf("Укажите номер столбца:\ni = ");
        scanf("%d", &n);
        if ((ptr = minimum(header, n)) == NULL) {
            printf("В столбце обнаружено нечисловое значение!\n");
            break;
        }
    }
}

```

```

        printf("Запись с минимальным значением:\n\n");
        print_solo(ptr);
        break;
    case 0:
        break;
    default:
        printf("Ответ не распознан. Попробуйте ещё раз.\n");
        continue;
    }
    clear;
    break;
case 3:
    printf("Меню 'EDIT'\nВыберите необходимую команду\n");
    printf("1. Изменить заголовок таблицы\n2. Добавить запись\n3.
Удалить запись.\n4. Редактировать ячейку.\n0. Выйти из блока 'EDIT'\n");
    scanf("%d", &ians);
    switch (ians) {
    case 1:
        header = renam(header, ptr_sep);
        printf("Новый заголовок:\n");
        print_solo(header);
        break;
    case 2:
        append(header, ptr_sep);
        break;

    case 3:
        printf("Укажите номер строки:\ni = ");
        scanf("%d", &n);
        remov(header, n);
        break;
    case 4:
        printf("Укажите номер строки:\ni = ");
        scanf("%d", &n);
        printf("Укажите номер столбца:\nj = ");
        scanf("%d", &m);
        edit(header, n, m, ptr_sep);
        break;
    case 0:
        break;
    default:
        printf("Ответ не распознан. Попробуйте ещё раз.\n");
        continue;
    }
    clear;
    break;
case 4:
    printf("Меню 'SORT'\nВыберите необходимый режим
сортировки:\n");
    printf("1. Сортировать таблицу\n2. Вывести отсортированный
список\n0. Выйти из блока 'EDIT'\n");
    scanf("%d", &ians);
    switch (ians) {
    case 1:
        printf("Укажите номер столбца, по которому следует
выполнить сортировку\ni = ");
        scanf("%d", &n);
        clear;
        printf("Сортировать по возрастанию? (yes or no)\n");
        fgets(ans, 7, stdin);
        StAns = standart_command(ans);
        if (Yans) {
            header = sort(header, n, 1);
        }
        else {

```

```

        if (Nans) {
            header = sort(header, n, 0);
        }
        else {
            printf("Ответ не распознан, попробуйте ещё
раз.\n");
            continue;
        }
    }
    break;
case 2:
    printf("Укажите номер столбца, по которому следует
выполнить сортировку\ni = ");
    scanf("%d", &n);
    clear
    printf("Сортировать по возрастанию? (yes or no)\n");
    fgets(ans, 7, stdin);
    StAns = standart_command(ans);
    if (Yans) {
        print_sort(header, n, 1);
    }
    else {
        if (Nans) {
            print_sort(header, n, 0);
        }
        else {
            printf("Ответ не распознан, попробуйте ещё
раз.\n");
            continue;
        }
    }
    break;
case 0:
    break;
}
clear
break;
case 5:
    printf("Меню 'FIND'\nВыберите необходимую функцию\n");
    printf("1. Найти строку \n2. Найти ячейку\n3. Найти по
условию\n0. Выйти из блока 'FIND'\n");
    scanf("%d", &ians);
    switch (ians) {
case 1:
        printf("Укажите номер строки\ni = ");
        scanf("%d", &n);
        clear
        ptr = find_line(header, n);
        if (ptr == NULL) {
            printf("Строка не найдена");
            break;
        }
        print_solo(ptr);
        break;
case 2:
        printf("Укажите номер строки\ni = ");
        scanf("%d", &n);
        clear
        printf("Укажите номер столбца\nj = ");
        scanf("%d", &m);
        clear
        cptr = find_cell(header, n, m);
        if (cptr == NULL) {
            printf("Строка не найдена");
            break;

```

```

    }
    printf("%s", loc(cpтр));
    break;
case 3:
    printf("Укажите номер столбца\nj = ");
    scanf("%d", &m);
    clear
    printf("Введите значение.\nvalue = ");
    fgets(value, 127, stdin);
    value[strcspn(value, "\n")] = 0;
    printf("Введите режим:\n");
    printf("Режимы:\n1. == \n2. >\n3. <\n4. != \n");
    printf("i = ");
    scanf("%d", &n);
    clear
    ind = (index(header, m, value, n));
    print_index(header, ind);
    break;
case 0:
    break;
}
clear
break;

case 6:
    printf("Информация о таблице\n");
    info(header);
    break;
case 7:
    cls;
    break;
case 8:
    printf("Укажите путь к файлу (если его не существует, он будет
создан. В случае существования его содержимое перезапишется): ");
    scanf("%s", path);
    file = fopen(path, "w");
    if (file == NULL) {
        printf("Файл не был открыт, попробуйте снова");
        getchar();
        continue;
    }
    to_csv(header, file);
    clear
    break;
default:
    //-
    break;
}

}
}
return 0;
}

```

Play_csv_cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include "play_csv.h"
#define BUFF 64
char* str = NULL;
char none[] = "None";
char* ppstr;
size_t size = BUFF;
char h = 'h';

```

```
//=====
// Play_CSV
//=====
```

```
//Блок "FILE"
```

```
line* read_csv(FILE* data, char* sep)
{
    line* header = NULL;
    line* sample = NULL;
    line* p;
    cell* sample_cell = NULL;
    cell* p_cell;
    char* istr;
    int counter_line = 0;
    str = (char*)malloc(size);
    if ((str = (char*)malloc(size)) == NULL) {
        no_memory();
    }
    memset(str, 0, size);
    while (!feof(data)) {
        if ((str = fgets(str, size, data)) == NULL) {
            printf("строка пустая, вы не читали требования, но я удалю её просто.");
            continue;
        }
        if (check_memory(data)) {
            continue;
        }
        str[strcspn(str, "\n")] = 0;
        ppstr = str;
        counter_line++;
        if (counter_line == 1) {
            header = create_line();
            sample = header;
        }
        else {
            p = sample;
            sample = create_line();
            p->ptr_line = sample;
        }
        sample->str = (char*)malloc(strlen(str)+1);
        sample->str = strcpy(sample->str, str);
        istr = correct_split(sep);
        sample->mesh->value = istr;
        sample_cell = sample->mesh;
        while ((istr = correct_split(sep)) != NULL) {
            p_cell = sample_cell;
            sample_cell = create_cell();
            p_cell->ptr_cell = sample_cell;
            sample_cell->value = istr;
        }
    }
}
```

```

        }
        size = BUFF;
    }
    free(str);
    if (header == NULL) {
        return NULL;
    }
    return header;
}

```

```

line* read_csv_without_header(FILE* data, char* head, char* sep)
{

```

```

    line* header = NULL;
    line* sample = NULL;
    line* p;
    cell* sample_cell = NULL;
    cell* p_cell;
    char* istr;
    int counter_line = 0;
    if ((str = (char*)malloc(size)) == NULL) {
        no_memory();
    }
    memset(str, 0, size);

```

```

    header = create_line();
    sample = header;
    str = head;
    counter_line++;

```

```

    while (!feof(data)) {
        if ((str = fgets(str, size, data)) == NULL) {
            printf("строка пустая, вы не читали требования, но я удалю её просто.");
            continue;
        }
        if (check_memory(data)) {
            continue;
        }
        str[strcspn(str, "\n")] = 0;
        counter_line++;
        p = sample;
        sample = create_line();
        p->ptr_line = sample;
        sample->str = strcpy(sample->str, str);
        istr = correct_split(sep);
        sample->mesh->value = istr;
        sample_cell = sample->mesh;
        while ((istr = correct_split(sep)) != NULL) {
            p_cell = sample_cell;
            sample_cell = create_cell();
            p_cell->ptr_cell = sample_cell;
            sample_cell->value = istr;
        }
        size = BUFF;
    }

```

```

    }
    free(str);
    if (header == NULL) {
        return NULL;
    }
    return header;
}

void to_csv(line* header, FILE* in) {
    line* p = header;
    while (p != NULL) {
        fputs(p->str, in);
        if (p->ptr_line != NULL) {
            fputs("\n", in);
        }
        p = p->ptr_line;
    }
}

//Блок "INFO"

shape_struct shape(line* header) {
    shape_struct sh;
    sh.i = 0;
    sh.j = 0;
    line* p = header;
    cell* pc = p->mesh;
    while (pc->ptr_cell != NULL) {
        pc = pc->ptr_cell;
        sh.i++;
    }
    while (p->ptr_line != NULL) {
        p = p->ptr_line;
        sh.j++;
    }
    return sh;
}

void info(line* header) {
    int i = 1; int j = 1;
    line* p = header->ptr_line;
    cell* pc = p->mesh;
    while (pc->ptr_cell != NULL) {
        pc = pc->ptr_cell;
        i++;
    }
    while (p->ptr_line != NULL) {
        p = p->ptr_line;
        j++;
    }
    printf("Заголовочная строка:\n");
    puts(header->str);
    printf("Размер таблицы:\n");

```

```
        printf("[%d x %d]\n", i, j);  
    }
```

```
//Блок "FIND"
```

```
cell* find_cell(line* header, int i, int j) {  
    line* p = header->ptr_line;  
    cell* pc;  
    while ((p != NULL) & (i > 1)) {  
        p = p->ptr_line;  
        i--;  
    }  
    if (p == NULL) {  
        return NULL;  
    }  
    pc = p->mesh;  
    while ((pc != NULL) & (j > 1)) {  
        pc = pc->ptr_cell;  
        j--;  
    }  
    if (pc == NULL) {  
        return NULL;  
    }  
    return pc;  
}
```

```
line* find_line(line* header, int i) {  
    line* p = header->ptr_line;  
    while ((p != NULL) & (i > 1)) {  
        p = p->ptr_line;  
        i--;  
    }  
    if (p == NULL) {  
        return NULL;  
    }  
    return p;  
}
```

```
char* loc(cell* sample) {  
    char* stroka = (char*)malloc(strlen(sample->value));  
    strcpy(stroka, sample->value);  
    return stroka;  
}
```

```
int* index(line* header, int by, char* value, int mode) {  
    line* p = header->ptr_line;  
    cell* pc = p->mesh;  
    int num = by;  
    int i = 1;  
    int j = 0;  
    int numeric = type_column(header, by);  
    int* index_list = (int*)malloc(sizeof(int));  
    index_list[0] = -1;
```

```

    if (!(char_to_digit(value))) {
        numeric = 0;
    }
    switch (mode) {
    case (1):
        while (p != NULL) {
            pc = p->mesh;
            for (num; num > 1; num--) {
                pc = pc->ptr_cell;
            }
            if ((strcmp(pc->value, value)) == 0) {
                index_list[j] = i;
                j++;
                if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) == NULL) {
                    no_memory();
                }
                index_list[j] = -1;
            }
            p = p->ptr_line;
            i++;
            num = by;
        }
        break;
    case (2):
        while (p != NULL) {
            pc = p->mesh;
            for (num; num > 1; num--) {
                pc = pc->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(value) < char_to_digit(pc->value))) {
                    index_list[j] = i;
                    j++;
                    if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) ==
NULL) {
                        no_memory();
                    }
                    index_list[j] = -1;
                }
            }
            else {
                if ((strcmp(pc->value, value)) > 0) {
                    index_list[j] = i;
                    j++;
                    if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) ==
NULL) {
                        no_memory();
                    }
                    index_list[j] = -1;
                }
            }
            p = p->ptr_line;
            i++;
            num = by;
        }
    }
}

```

```

    }
    break;
case (3):
    while (p != NULL) {
        pc = p->mesh;
        for (num; num > 1; num--) {
            pc = pc->ptr_cell;
        }
        if (numeric) {
            if ((char_to_digit(value) > char_to_digit(pc->value))) {
                index_list[j] = i;
                j++;
                if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) ==
NULL) {
                    no_memory();
                }
                index_list[j] = -1;
            }
        }
        else {
            if ((strcmp(pc->value, value)) < 0) {
                index_list[j] = i;
                j++;
                if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) ==
NULL) {
                    no_memory();
                }
                index_list[j] = -1;
            }
        }
        p = p->ptr_line;
        i++;
        num = by;
    }
    break;
case (4):
    while (p != NULL) {
        pc = p->mesh;
        for (num; num > 1; num--) {
            pc = pc->ptr_cell;
        }
        if ((strcmp(pc->value, value)) != 0) {
            index_list[j] = i;
            j++;
            if ((index_list = (int*)realloc(index_list, (sizeof(int) * (j + 1)))) == NULL) {
                no_memory();
            }
            index_list[j] = -1;
        }
        p = p->ptr_line;
        i++;
        num = by;
    }
    break;

```

```

        default:
            printf("Неверно указан mode");
            break;
    }
    if (index_list[0] == -1) {
        return NULL;
    }
    return index_list;
}

//Блок "PRINT"

void print(line* header) {
    int n = 1;
    line* p = header;
    cell* pc = p->mesh;
    printf("%-2c\t", h);
    while (pc != NULL) {
        printf("%-16s\t", pc->value);
        pc = pc->ptr_cell;
    }
    printf("\n\n");
    p = p->ptr_line;
    while (p != NULL) {
        pc = p->mesh;
        printf("%-2i\t", n);
        while (pc != NULL) {
            printf("%-16s\t", pc->value);
            pc = pc->ptr_cell;
        }
        printf("\n");
        p = p->ptr_line;
        n++;
    }
}

void print_head(line* header, int i) {
    line* p = header;
    cell* pc = p->mesh;
    int n = 1;
    if (i < 1) {
        printf("no");
    }
    printf("%-2c\t", h);
    while (pc != NULL) {
        printf("%-16s\t", pc->value);
        pc = pc->ptr_cell;
    }
    printf("\n\n");
    p = p->ptr_line;
    while ((p != NULL) & (i > 0)) {
        pc = p->mesh;
        printf("%-2i\t", n);

```

```

        while (pc != NULL) {
            printf("%-16s\t", pc->value);
            pc = pc->ptr_cell;
        }
        printf("\n");
        p = p->ptr_line;
        n++;
        i--;
    }
    printf(" . . \n");
}

```

```

void print_tail(line* header, int i) {
    line* beg = header;
    line* end = header;
    cell* pc = beg->mesh;
    int n = 1;
    if (i < 1) {
        printf("no");
    }
    printf("%-2c\t", h);
    while (pc != NULL) {
        printf("%-16s\t", pc->value);
        pc = pc->ptr_cell;
    }
    printf("\n\n");
    printf(" . . \n");
    beg = beg->ptr_line;

    end = beg;
    while ((end->ptr_line != NULL) & (i > 0)) {
        end = end->ptr_line;
        i--;
    }
    while (end != NULL) {
        end = end->ptr_line;
        beg = beg->ptr_line;
        n++;
    }
    while (beg != NULL) {
        pc = beg->mesh;
        printf("%-2i\t", n);
        while (pc != NULL) {
            printf("%-16s\t", pc->value);
            pc = pc->ptr_cell;
        }
        printf("\n");
        beg = beg->ptr_line;
        n++;
    }
}

```

```

void print_slice(line* header, int i, int j) {
    line* beg = header;

```

```

line* end = header;
cell* pc = beg->mesh;
int n = i;
if ((i < 1) || (j < 1) || (j < i)) {
    printf("no");
}
printf("%-2c\t", h);
while (pc != NULL) {
    printf("%-16s\t", pc->value);
    pc = pc->ptr_cell;
}
printf("\n\n");
printf(". . \n");
beg = beg->ptr_line;
end = beg;

while ((end->ptr_line != NULL) & (j > 0)) {
    end = end->ptr_line;
    j--;
}
while ((beg->ptr_line != NULL) & (i > 1)) {
    beg = beg->ptr_line;
    i--;
}
while (beg->ptr_line != end) {
    pc = beg->mesh;
    printf("%-2i\t", n);
    while (pc != NULL) {
        printf("%-16s\t", pc->value);
        pc = pc->ptr_cell;
    }
    printf("\n");
    beg = beg->ptr_line;
    n++;
}
printf(". . \n");
}

```

```

void print_index(line* header, int* ind) {
    line* p = header->ptr_line;
    int i = 1;
    while (p != NULL) {
        if (is_sequence(ind, i)) {
            print_solo(p);
        }
        i++;
        p = p->ptr_line;
    }
}

```

```

void print_solo(line* sample) {
    cell* pc = sample->mesh;
    while (pc != NULL) {
        printf("%-16s\t", pc->value);
    }
}

```

```

        pc = pc->ptr_cell;
    }
    printf("\n");
}

```

//Блок "SORT"

```

line* sort(line* header, int by, bool ascending) {
    // selection sort
    line* p = header->ptr_line;
    line* end_s = header;
    line* begin_ns = end_s->ptr_line;
    line* ext = p;
    line* before_min = p;
    line* before = p;
    cell* pc1 = p->mesh;
    cell* pc2 = ext->mesh;
    int i = 1;
    int numeric = type_column(header, by);
    if (ascending) {
        while (begin_ns != NULL) {
            pc2 = ext->mesh;
            for (int j = by; j > 1; j--) {
                pc2 = pc2->ptr_cell;
            }
            while (p != NULL) {
                pc2 = ext->mesh;
                for (int j = by; j > 1; j--) {
                    pc2 = pc2->ptr_cell;
                }
                pc1 = p->mesh;
                for (int j = by; j > 1; j--) {
                    pc1 = pc1->ptr_cell;
                }
                if (numeric) {
                    if ((char_to_digit(pc1->value) <= (char_to_digit(pc2->value)))) {
                        ext = p;
                        before_min = before;
                    }
                }
                else {
                    if ((strcmp(pc1->value, pc2->value)) <= 0) {
                        ext = p;
                        before_min = before;
                    }
                }
                if (i) {
                    i = 0;
                    p = p->ptr_line;
                    continue;
                }
                before = before->ptr_line;
                p = p->ptr_line;
            }
        }
    }
}

```

```

    }
    i = 1;
    if (ext == before_min) {
        begin_ns = ext->ptr_line;
        end_s = ext;
    }
    else {
        before_min->ptr_line = ext->ptr_line;
        ext->ptr_line = end_s->ptr_line;
        end_s->ptr_line = ext;
        end_s = ext;
        begin_ns = end_s->ptr_line;
    }
    p = begin_ns;
    before = p;
    ext = p;
}
}
else {
    while (begin_ns != NULL) {
        pc2 = ext->mesh;
        for (int j = by; j > 1; j--) {
            pc2 = pc2->ptr_cell;
        }
        while (p != NULL) {
            pc2 = ext->mesh;
            for (int j = by; j > 1; j--) {
                pc2 = pc2->ptr_cell;
            }
            pc1 = p->mesh;
            for (int j = by; j > 1; j--) {
                pc1 = pc1->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc1->value) >= (char_to_digit(pc2->value)))) {
                    ext = p;
                    before_min = before;
                }
            }
            else {
                if ((strcmp(pc1->value, pc2->value)) >= 0) {
                    ext = p;
                    before_min = before;
                }
            }
        }
        if (i) {
            i = 0;
            p = p->ptr_line;
            continue;
        }
        before = before->ptr_line;
        p = p->ptr_line;
    }
}

```

```

        i = 1;
        if (ext == before_min) {
            begin_ns = ext->ptr_line;
            end_s = ext;
        }
        else {
            before_min->ptr_line = ext->ptr_line;
            ext->ptr_line = end_s->ptr_line;
            end_s->ptr_line = ext;
            end_s = ext;
            begin_ns = end_s->ptr_line;
        }
        p = begin_ns;
        before = p;
        ext = p;
    }
}
return header;
}

```

```

line* sort_in(line* header, int by, bool ascending) {
    line* header_new = copy(header);
    line* p_new = header_new;
    line* pp_new = header_new;
    line* p = header->ptr_line;
    line* end_s = header->ptr_line;
    line* ext = create_line();
    line* ext_reserv = ext;
    cell* pc1 = p->mesh;
    cell* pc2;
    int numeric = type_column(header, by);
    int pr = 0;
    int j = 0;
    int* sequence = (int*)malloc(sizeof(int) * 2);
    sequence[0] = -2;
    sequence[1] = -1;

    char minmax[2];
    char* pstr;
    __asm {
        lea eax, minmax;
        mov pstr, eax;
        mov eax, pstr;
        mov byte ptr[eax], 0xFF;
        inc pstr;
        mov eax, pstr;
        mov byte ptr[eax], 0x01;
    }

    if (ascending) {
        ext->mesh->value = &minmax[0];
        ext_reserv = ext;
        while (end_s != NULL) {

```

```

        pc2 = ext->mesh;
        while (p != NULL) {
            pc1 = p->mesh;
            for (int j = by; j > 1; j--) {
                pc1 = pc1->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc1->value) <= (char_to_digit(pc2->value))) &
(is_sequence(sequence, j) == 0)) {
                    ext = p;
                    pc2 = ext->mesh;
                    sequence[pr] = j;
                    sequence[pr + 1] = -1;
                }
            }
            else {
                if ((strcmp(pc1->value, pc2->value) <= 0) &
(is_sequence(sequence, j) == 0)) {
                    ext = p;
                    pc2 = ext->mesh;
                    sequence[pr] = j;
                    sequence[pr + 1] = -1;
                }
            }
            p = p->ptr_line;
            j++;
        }
        j = 0;
        p_new = copy(ext);
        ext = ext_reserv;
        pp_new->ptr_line = p_new;
        pp_new = pp_new->ptr_line;
        pr++;
        if ((sequence = (int*)realloc(sequence, (sizeof(int) * (2 + pr)))) == NULL) {
            no_memory();
        }
        sequence[pr + 1] = -1;
        end_s = end_s->ptr_line;
        p = header->ptr_line;
    }
}
else {
    ext->mesh->value = &minmax[1];
    ext_reserv = ext;
    while (end_s != NULL) {
        pc2 = ext->mesh;
        while (p != NULL) {
            pc1 = p->mesh;
            for (int j = by; j > 1; j--) {
                pc1 = pc1->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc1->value) >= (char_to_digit(pc2->value))) &
(is_sequence(sequence, j) == 0)) {

```

```

        ext = p;
        pc2 = ext->mesh;
        sequence[pr] = j;
        sequence[pr + 1] = -1;
    }
}
else {
    if ((strcmp(pc1->value, pc2->value) >= 0) &
(is_sequence(sequence, j) == 0)) {
        ext = p;
        pc2 = ext->mesh;
        sequence[pr] = j;
        sequence[pr + 1] = -1;
    }
}
p = p->ptr_line;
j++;
}
j = 0;
p_new = copy(ext);
ext = ext_reserv;
pp_new->ptr_line = p_new;
pp_new = pp_new->ptr_line;
pr++;
if ((sequence = (int*)realloc(sequence, (sizeof(int) * (2 + pr)))) == NULL) {
    no_memory();
}
sequence[pr + 1] = -1;
end_s = end_s->ptr_line;
}
}
free(sequence);
free(ext_reserv);
return header_new;
}

```

```

void print_sort(line* header, int by, bool ascending) {
    int* sequence = (int*)malloc(sizeof(int)*2);
    int i = 0;
    int pr = 0;
    line* p = header->ptr_line;
    line* pp = p;
    cell* pc;
    char mins[] = "2147483647";
    char maxs[] = "-2147483647";
    char* max;
    char* min;
    char minmax[2];
    char* pstr;
    int numeric = type_column(header, by);
    if (numeric) {
        min = &mins[0];
        max = &maxs[0];
    }
}

```

```

else {
    __asm {
        lea eax, minmax;
        mov pstr, eax;
        mov eax, pstr;
        mov byte ptr[eax], 0xFF;
        inc pstr;
        mov eax, pstr;
        mov byte ptr[eax], 0x01;
    }
    min = &minmax[0];
    max = &minmax[1];
}
sequence[0] = -2;
sequence[1] = -1;
if (ascending) {
    while (pp != NULL) {
        while (p != NULL) {
            pc = p->mesh;
            for (int j = by; j > 1; j--) {
                pc = pc->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc->value) <= (char_to_digit(min))) &
(is_sequence(sequence, i) == 0)) {
                    min = pc->value;
                    sequence[pr] = i;
                    sequence[pr + 1] = -1;
                }
            }
            else {
                if ((strcmp(pc->value, min) <= 0) & (is_sequence(sequence, i) ==
0)) {
                    min = pc->value;
                    sequence[pr] = i;
                    sequence[pr + 1] = -1;
                }
            }
            p = p->ptr_line;
            i++;
        }
        pp = pp->ptr_line;
        p = header->ptr_line;
        pr++;
        i = 0;
        if (numeric) {
            min = &mins[0];
        }
        else {
            min = &minmax[0];
        }
        if ((sequence = (int*)realloc(sequence, (sizeof(int) * (2 + pr)))) == NULL) {
            no_memory();
        }
    }
}

```

```

        sequence[pr + 1] = -1;
    }

}
else {
    while (pp != NULL) {
        while (p != NULL) {
            pc = p->mesh;
            for (int j = by; j > 1; j--) {
                pc = pc->ptr_cell;
            }
            if (numeric) {
                if ((char_to_digit(pc->value) >= (char_to_digit(max))) &
(is_sequence(sequence, i) == 0)) {
                    max = pc->value;
                    sequence[pr] = i;
                    sequence[pr + 1] = -1;
                }
            }
            else {
                if ((strcmp(pc->value, max) >= 0) & (is_sequence(sequence, i) ==
0)) {
                    max = pc->value;
                    sequence[pr] = i;
                    sequence[pr + 1] = -1;
                }
            }
            p = p->ptr_line;
            i++;
        }
        pp = pp->ptr_line;
        p = header->ptr_line;
        pr++;
        i = 0;
        if (numeric) {
            max = &maxs[0];
        }
        else {
            max = &minmax[1];
        }
        if ((sequence = (int*)realloc(sequence, (sizeof(int) * (2 + pr)))) == NULL) {
            no_memory();
        }
        sequence[pr + 1] = -1;
    }

}

i = 0;
pr = 0;
p = header;
pc = p->mesh;
printf("%-2c\t", h);
while (pc != NULL) {
    printf("%-16s\t", pc->value);

```

```

        pc = pc->ptr_cell;
    }
    printf("\n\n");
    p = p->ptr_line;
    while (sequence[i] != -1) {
        pr = sequence[i];
        for (pr; pr > 0; pr--) {
            p = p->ptr_line;
        }
        printf("%-2i\t", i+1);
        pc = p->mesh;
        while (pc != NULL) {
            printf("%-16s\t", pc->value);
            pc = pc->ptr_cell;
        }
        printf("\n");
        p = header->ptr_line;
        i++;
    }
    free(sequence);
}

```

//Блок "EDIT"

```

void remov(line* header, int by) {
    line* p = header->ptr_line;
    line* bef = header;
    while ((by > 1) & (p->ptr_line != NULL)) {
        p = p->ptr_line;
        bef = bef->ptr_line;
        by--;
    }
    bef->ptr_line = p->ptr_line;

    free(p);
    p = NULL;
}

```

```

void append(line* header, char* sep) {
    cell* hp = header->mesh;
    line* end = header;
    line* fresh = create_line();
    cell* f_cell = fresh->mesh;
    cell* f_cell2;
    char* ptr;
    char* stroka = (char*)calloc(1, 1024);
    int sumsize = 0;
    str = (char*)malloc(size);
    while (end->ptr_line != NULL) {
        end = end->ptr_line;
    }
    end->ptr_line = fresh;
    while (hp != NULL) {

```

```

scanf_s("%*[^\\n]");
scanf_s("%*c");
printf("Введите %s ", hp->value);
fgets(stroka, 1024, stdin);
ppstr = stroka;
stroka[strcspn(stroka, "\\n")] = 0;
while ((strlen(stroka) > size) & ((int)size > sumsize)) {
    size *= 2;
    str = (char*)realloc(str, size);
}
memset(str, 0, size);
sumsize += strlen(stroka);
strcat(str, stroka);
if (hp->ptr_cell != NULL) {
    strcat(str, sep);
}
if (ppstr[0] == '"') {
    ppstr++;
    ppstr[strlen(stroka) - 2] = '\\0';
}
if ((f_cell->value = (char*)realloc(f_cell->value, strlen(ppstr))) == NULL) {
    no_memory();
}
f_cell->value = strcpy(f_cell->value, ppstr);
if (hp->ptr_cell == NULL) {
    break;
}
f_cell2 = create_cell();
f_cell->ptr_cell = f_cell2;
f_cell = f_cell2;
hp = hp->ptr_cell;
}
if ((fresh->str = (char*)realloc(fresh->str, strlen(str))) == NULL) {
    no_memory();
}
fresh->str = strcpy(fresh->str, str);
size = BUFF;
free(stroka);
free(str);
}

void edit(line* header, int i, int j, char* sep) {
    line* p = header->ptr_line;
    line* zag = header;
    cell* pc1 = header->mesh;
    cell* pc2 = p->mesh;
    char* stroka;
    int difference = 0;
    char* ptr;
    while ((p != NULL) & (i > 1)) {
        p = p->ptr_line;
        i--;
    }

```

```

    if (p == NULL) {
        printf("Выход за пределы массива");
        exit(0);
    }
    pc2 = p->mesh;
    while ((pc2 != NULL) & (j > 1)) {
        pc2 = pc2->ptr_cell;
        j--;
    }
    if (pc2 == NULL) {
        printf("Выход за пределы массива");
        exit(0);
    }
    stroka = (char*)calloc(1, 64);
    scanf_s("%*[^\\n]");
    scanf_s("%*c");
    printf("Редактируемая ячейка: %s\\nВведите новое значение: \\n", pc2->value);
    fgets(stroka, 64, stdin);
    stroka[strcspn(stroka, "\\n")] = 0;
    difference = strlen(stroka) - strlen(pc2->value);
    if ((p->str = (char*)realloc(p->str, (strlen(p->str)+difference))) == NULL) {
        no_memory();
    }
    ptr = stroka;
    if (stroka[0] == "") {
        ptr++;
        ptr[strlen(stroka) - 2] = '\\0';
        difference -= 2;
    }
    if ((pc2->value = (char*)realloc(pc2->value, (strlen(pc2->value) + difference))) == NULL) {
        no_memory();
    }
    strcpy(pc2->value, stroka);
    pc2 = p->mesh;
    memset(p->str, 0, strlen(p->str));
    while (pc2 != NULL) {
        strcat(p->str, pc2->value);
        if (pc2->ptr_cell != NULL) {
            strcat(p->str, sep);
        }
        pc2 = pc2->ptr_cell;
    }
    free(stroka);
}

```

```

line* renam(line* header, char* sep) {
    cell* pc = header->mesh;
    int difference = 0;
    char* istr;
    char* stroka;
    stroka = (char*)calloc(1, 1024);
    scanf_s("%*[^\\n]");
    scanf_s("%*c");
    printf("Введите новую заголовочную строку\\n:");

```

```

fgets(stroka, 1024, stdin);
ppstr = stroka;
ppstr[strcspn(str, "\n")] = 0;
difference = strlen(str) - strlen(header->str);
if (!(difference)) {
    if ((header->str = (char*)realloc(header->str, (strlen(header->str) + difference))) ==
NULL) {
        no_memory();
    }
}
memset(header->str, 0, strlen(header->str));
strcpy(header->str, str);
while (pc != NULL) {
    istr = correct_split(sep);
    strcpy(pc->value, istr);
    pc = pc->ptr_cell;
}
return header;
}

```

//Блок "МАТН"

```

line* median(line* header) {
    line* med = header->ptr_line;
    line* p = header->ptr_line;
    int i = 1;
    while (p != NULL) {
        if (i % 2 == 0) {
            med = med->ptr_line;
        }
        p = p->ptr_line;
        i++;
    }
    return med;
}

double mean(line* header, int by) {
    double sum = 0;
    int count = 0;
    double num;
    line* p = header->ptr_line;
    cell* pc;
    while (p != NULL) {
        pc = p->mesh;
        for (int j = by; j > 1; j--) {
            pc = pc->ptr_cell;
        }
        if ((num = char_to_digit(pc->value)) != NULL) {
            sum = sum + num;
            p = p->ptr_line;
            count++;
            continue;
        }
    }
}

```

```

        }
        return NULL;
    }

    return (sum / count);
}

line* maximum(line* header, int by) {
    line* p = header->ptr_line;
    line* max = p;
    cell* maxc = p->mesh;
    double maxc_v;
    double value;
    cell* pc = p->mesh;
    for (int j = by; j > 1; j--) {
        maxc = maxc->ptr_cell;
    }

    if ((maxc_v = char_to_digit(maxc->value)) == NULL) {
        return NULL;
    }
    while (p != NULL) {
        pc = p->mesh;
        for (int j = by; j > 1; j--) {
            pc = pc->ptr_cell;
        }
        if ((value = char_to_digit(pc->value)) != NULL) {
            p = p->ptr_line;
            if (value > maxc_v) {
                maxc_v = value;
                max = p;
            }
            continue;
        }
    }
    return NULL;
}

return max;
}

line* minimum(line* header, int by) {
    line* p = header->ptr_line;
    line* min = p;
    cell* minc = p->mesh;
    double minc_v;
    double value;
    cell* pc = p->mesh;
    for (int j = by; j > 1; j--) {
        minc = minc->ptr_cell;
    }

    if ((minc_v = char_to_digit(minc->value)) == NULL) {
        return NULL;
    }

```

```

    }
    while (p != NULL) {
        pc = p->mesh;
        for (int j = by; j > 1; j--) {
            pc = pc->ptr_cell;
        }
        if ((value = char_to_digit(pc->value)) != NULL) {
            if (value < minc_v) {
                minc_v = value;
                min = p;
            }
            p = p->ptr_line;
            continue;
        }
        return NULL;
    }

    return min;
}

```

// Вспомогательные функции
double char_to_digit(char* src) {

```

    int int_ = 0;
    int fract_ = -1;
    double full;
    int sign = 0;
    int no_digit_ = 0;
    char digit;
    char* pstr;
    __asm {
        mov eax, src;
        mov pstr, eax;
        mov eax, pstr;
        mov cl, byte ptr[eax];
        cmp cl, 0x2D;
        jne LabelStart1;
        mov sign, 1;
        inc pstr;
LabelStart1:
        mov eax, pstr;
        xor ecx, ecx;
        mov cl, byte ptr[eax];
        test cl, cl;
        je LabelEnd1;
        cmp cl, 0x2E;
        je fract;
        cmp cl, 0x30;
        jl no_digit;
        cmp cl, 0x39;
        jg no_digit;
        sub ecx, 0x30;

```

```

        mov digit, cl;
        mov eax, int_;
        xor ebx, ebx;
        mov bx, 10;
        mul bx;
        xor ebx, ebx;
        mov bl, digit;
        add eax, ebx;
        mov int_, eax;
        inc pstr;
        jmp LabelStart1;
no_digit:
        mov no_digit_, 1;
        jmp LabelEnd1;
fract:
        mov fract_, 0;
        inc pstr;
        jmp LabelStart2;
LabelStart2:
        mov eax, pstr;
        xor ecx, ecx;
        mov cl, byte ptr[eax];
        test cl, cl;
        je LabelEnd1;
        cmp cl, 0x30;
        jl no_digit;
        cmp cl, 0x39;
        jg no_digit;
        sub ecx, 0x30;
        mov digit, cl;
        mov eax, fract_;
        xor ebx, ebx;
        mov bx, 10;
        mul bx;
        xor ebx, ebx;
        mov bl, digit;
        add eax, ebx;
        mov fract_, eax;
        inc pstr;
        jmp LabelStart2;
LabelEnd1:
}
if (no_digit_ == 1) {
    return NULL;
}
if (fract_ != -1) {
    full = fract_;
    while (full > 1) {
        full /= 10;
    }
    full = full + int_;
}
else {
    full = int_;

```

```

    }
    if (sign == 1) {
        full = full * (-1);
    }
    return full;
}

char* split(char* sep) {
    char* bef = (char*)malloc(1);
    int a = 1;
    int comma = 0;
    if (ppstr[0] == '\0') {
        return NULL;
    }
    comma = strcspn(ppstr, sep);
    if ((bef = (char*)realloc(bef, comma)) == NULL) {
        no_memory();
    }
    memset(bef, 0, comma);
    if (comma >= strlen(ppstr)) {
        strncpy(bef, ppstr, comma);
        ppstr = ppstr + comma;
        bef[comma] = '\0';
        return bef;
    }
    strncpy(bef, ppstr, comma);
    bef[comma] = '\0';
    ppstr = ppstr + comma + 1;
    return bef;
}

char* proc_quote(char* sep) {
    char* ptr;
    char* bef = (char*)malloc(1);
    char* proc_sep = (char*)calloc(1, 2);
    int s = 1;
    int comma = 0;
    bef = (char*)malloc(1);
    ppstr++;
    if (ppstr[0] == '"') {
        ppstr += 2;
        bef = (char*)realloc(bef, (strlen(none)));
        memset(bef, 0, strlen(none));
        strcpy(bef, none);
        return bef;
    }
    proc_sep[0] = '"';
    strcat(proc_sep, sep);
    if (ppstr[0] == '\0') {
        return NULL;
    }
    ptr = strstr(ppstr, proc_sep);
    if (ptr == NULL) {
        comma = strlen(ppstr);
    }

```

```

        if ((bef = (char*)realloc(bef, comma)) == NULL) {
            no_memory();
        }
        strncpy(bef, ppstr, comma);
        bef[comma - 1] = '\0';
        ppstr = ppstr + comma;
        return bef;
    }
    comma = strlen(ppstr) - strlen(ptr);
    if ((bef = (char*)realloc(bef, comma)) == NULL) {
        no_memory();
    }
    memset(bef, 0, comma);
    if (comma >= strlen(ppstr)) {
        strncpy(bef, ppstr, comma);
        ppstr = ppstr + comma;
        return bef;
    }
    strncpy(bef, ppstr, comma);
    bef[comma] = '\0';
    ppstr = ppstr + comma + 2;
    return bef;
}

```

```

char* correct_split(char* sep) {
    char* istr;
    if (ppstr[0] == '"') {
        istr = proc_quote(sep);
    }
    else {
        istr = split(sep);
    }
    return istr;
}

```

```

bool is_sequence(int* seq, int n) {
    bool ans = 0;
    int i = 0;
    while (seq[i] != -1) {
        if (n == seq[i]) {
            ans = 1;
            return ans;
        }
        i++;
    }
    return ans;
}

```

```

line* copy(line* sample)
{
    line* cop = create_line();
    cell* ptr_cop = cop->mesh;
    cell* ptr_cop2;

```

```

cell* ptr_sam = sample->mesh;
if ((cop->str = (char*)realloc(cop->str, strlen(sample->str))) == NULL) {
    no_memory();
}
cop->str = strcpy(cop->str, sample->str);
while (ptr_sam != NULL) {
    if ((ptr_cop->value = (char*)realloc(ptr_cop->value, strlen(ptr_sam->value))) == NULL) {
        no_memory();
    }
    ptr_cop->value = strcpy(ptr_cop->value, ptr_sam->value);
    if (ptr_sam->ptr_cell == NULL) {
        return cop;
    }
    ptr_cop2 = create_cell();
    ptr_cop->ptr_cell = ptr_cop2;
    ptr_cop = ptr_cop2;
    ptr_sam = ptr_sam->ptr_cell;
}
return cop;
}

void* no_memory() {
    printf("Ошибка. Попробуйте позже.");
    exit(0);
}

cell* create_cell() {
    cell* sample = (cell*)calloc(1, sizeof(cell));
    sample->ptr_cell = NULL;
    return sample;
}

line* create_line() {
    line* sample = (line*)calloc(1, sizeof(line));
    sample->ptr_line = NULL;
    sample->mesh = create_cell();
    return sample;
}

int check_memory(FILE* data) {
    int pos = ftell(data);
    int offset = pos - (size - 1);
    int len = strlen(str);
    if (size == len + 1) {
        fseek(data, offset, SEEK_SET);
        size *= 2;
        if ((str = (char*)realloc(str, size)) == NULL) {
            no_memory();
        }
        return 1;
    }
    return 0;
}

```

```

int type_column(line* header, int n) {
    line* p = header->ptr_line;
    int b = n;
    cell* pc = p->mesh;
    for (b; b > 1; b--) {
        pc = pc->ptr_cell;
    }
    while ((p->ptr_line != NULL) & (char_to_digit(pc->value) != NULL)) {
        pc = p->mesh;
        for (b; b > 1; b--) {
            pc = pc->ptr_cell;
        }
        p = p->ptr_line;
        b = n;
    }
    b = n;
    if (p->ptr_line == NULL) {
        pc = p->mesh;
        for (b; b > 1; b--) {
            pc = pc->ptr_cell;
        }
        if (char_to_digit(pc->value) != NULL) {
            return 1;
        }
    }
    return 0;
}

```

// Интерфейс

```

char* standart_command(char* command) {
    char* standart = (char*)malloc(strlen(command)+1);
    int i;
    int j = 0;
    while (command[j]) {
        command[j] = tolower(command[j]);
        j++;
    }
    j = 0;
    for (i = 0; i < strlen(command); i++) {
        if (command[i] == ' ')
            continue;
        standart[j] = command[i];
        j++;
    }
    standart[strlen(standart, "\n")] = 0;
    return standart;
}

```

```

int get_num(char* command) {
    if ((strcmp(command, "exit") == 0)) {
        return -2;
    }
}

```

```
    if ((strcmp(command, "help") == 0)) {  
        return -1;  
    }  
    if ((strcmp(command, "print") == 0)) {  
        return 1;  
    }  
    if ((strcmp(command, "math") == 0)) {  
        return 2;  
    }  
    if ((strcmp(command, "edit") == 0)) {  
        return 3;  
    }  
    if ((strcmp(command, "sort") == 0)) {  
        return 4;  
    }  
    if ((strcmp(command, "find") == 0)) {  
        return 5;  
    }  
    if ((strcmp(command, "info") == 0)) {  
        return 6;  
    }  
    if ((strcmp(command, "cls") == 0)) {  
        return 7;  
    }  
    if ((strcmp(command, "write") == 0)) {  
        return 8;  
    }  
    return NULL;  
}
```