АННОТАЦИЯ

Данная пояснительная записка содержит описание программы для работы с табличными данными, а именно, записями о студентах, разработанной в рамках курсового проектирования, целью которого закрепление и углубление знаний в области основ программирования.

СОДЕРЖАНИЕ

АННОТАЦИЯ.....2
ВВЕДЕНИЕ.....4
1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ.....5
2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ.....6
2.1 Постановка задачи на разработку программы (Вариант 23).....6
2.2 Применяемые математические методы.....6
2.3 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных.....6
2.4 Разработка модульной структуры программы.....7
2.5 Описание алгоритмов функционирования программы.....17
2.6 Обоснование состава технических и программных средств.....25
3 ВЫПОЛНЕНИЕ ПРОГРАММЫ.....25
3.1 Условия выполнения программы.....25
3.2 Загрузка и запуск программы.....26
3.3 Проверка работоспособности программы.....26
ВЫВОДЫ.....38
Перечень ссылок.....39
Приложение А ИСХОДНЫЙ КОД ПРОГРАММЫ.....40

ВВЕДЕНИЕ

В рамках настоящего курсового проектирования ведется разработка программы по теме «Программа обработки данных о студентах» на основании документа – техническое задание и в рамках организации – Севастопольский государственный университет. Дата выдачи задания: 09.09.2020.

С появлением и широким распространением табличных процессоров, одним из самых известных представителей которых является Microsoft Exel, и иных программ, позволяющих обрабатывать большие объёмы табличных данных без необходимости знания пользователем языков программирования, написание узкоспециализированных программ, примером которых является разрабатываемая программа, утратило свою актуальность, что, тем не менее, никак не сказалось на возможности использования разработки программы, преимуществом которой является простота, для закрепления знаний.

Целью курсового проектирования является систематизация, закрепление и углубление знаний в области основ программирования и совершенствование практических навыков разработки программ на языке Си на примере разработки «программы обработки данных о студентах», представляющей собой упрощённое подобие базы данных и позволяющей выполнять различные операции над записями, в том числе осуществление выборки 5 юношей и девушек с наиболее высокими баллами ЕГЭ в каждой группе.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

- выбор варианта задания и детализация поставки задачи;
- определение требований к функциям, выполняемых разрабатываемой программой;
- выбор типов и проектирование структур данных, определяющих способы представления, хранения и преобразования входных, выходных и промежуточных данных;
- разработка модульной структуры программы, определение функций модулей и способов их взаимодействия;
- написание текста программных модулей на алгоритмическом языке;
- разработка тестовых примеров;
- тестирование и отладка программы;
- разработка программных документов в соответствии с действующими стандартами.

1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

Программа предназначена для организации, хранения, модификации записей о студентах и предоставление удобного доступа к ним через консольный пользовательский интерфейс.

Областью применения программы может быть приёмная комиссия Севастопольского государственного университета в случае отказа оной от использования значительно более продвинутого продукта 1С, поскольку позволяет сортировать студентов по баллу ЕГЭ и дате поступления, запоминает пол, ФИО, дату рождения, группу и форму обучения.

2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

2.1 Постановка задачи на разработку программы (Вариант 23)

Структура записей входного файла имеет следующий вид:
- шифр группы (6 символов);
- номер зачетной книжки студента;
- ФИО студента;
- пол;
- форма обучения;
- дата рождения;
- дата поступления;
- балл ЕГЭ.

Разрабатываемая программа должна использовать меню-ориентированный интерфейс, обеспечивающий выполнение следующего минимального состава действий:
- начальное создание таблицы;
- просмотр таблицы с возможностью скроллинга;
- добавление новой записи в таблицу;
- удаление записи (по ключевому полю);
- корректировка записи в таблице (по ключевому полю);
- сортировка таблицы;
- поиск записи в таблице (по ключевому полю);
- сохранение таблицы в указанном пользователем текстовом файле;
- обработка таблицы и просмотр результатов обработки;
- завершение работы программы;

В процессе работы с одной и той же таблицей она должна автоматически сохраняться в типизированном файле и её загрузка из типизированного файла должна происходить автоматически при новом запуске программы.

При обработке таблицы необходимо выбрать в каждой группе по 5 девушек и юношей, имеющих высший балл.

2.2 Применяемые математические методы

В ходе разработки активно применялся макрос для нахождения длинны статического массива.

```
#define len(x) (sizeof(x) / sizeof((x)[0]))
```

Рисунок 2.1 – Макрос len

2.3 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных

Для хранения данных в памяти было решено использовать двухнаправленный список, поскольку он позволил упростить и успорить реализацию скролла вверх. Во избежание дополнительных выделений памяти во время поиска и лишнего перехода по указателю, которые бы неизбежно возникли в случае использования двух двунаправленных списков, ссылающихся на одни и те же данные, было решено использовать одни и те же элементы и в основном списке и вспомогательном, ипользуемом для поиска. Для этого в изначальный вариант элемента списка было добавлено ещё два указателя на предыдущий и следующий элемент, а так же введено понятие уровня связки («`link_layer`» в коде), изменение которого заставляет программу использовать начало, конец, длинну и пару указателей в элементе либо для основного списка, либо для вспомогательного. Использование же макросов позволяет скрыть особенности реализации, связанные с уровнем связывания, и работать со списком не задумываясь, в каком режиме он сейчас находится, если это не необходимо.

Для хранения строк было решено применять формат, напоминающий таковые же в языке Паскаль, когда вместо нулевого символа записывается длинна строки. Это решение продиктовано тем, что функция для считывания строки активно использует эту информацию. Потенциально это так же могло позволить сократить размер бинарного файла автосохранения, если бы в него записывалась только минимально необходимая часть массива, отведённого под строку, и её длинна, но было решено не делать размер записи в файле варьируемым, чтобы не усложнять процесс чтения, и чтобы можно было легко сказать, сколько записей хранится в файле по его размеру и выделить необходимое количество памяти при запуске программы, и ограничиться той оптимизацией размера, что даёт запись чисел в двоичном формате.

2.4 Разработка модульной структуры программы

В основу организации программы был положен принцип событийного управления. В процессе разработки для упрощения навигации в коде программы было принято решение разделить программу на 4 части: пользовательский ввод, список, пользовательский интерфейс и основную программу, выполняющую так же связующую функцию между перечисленными частями.

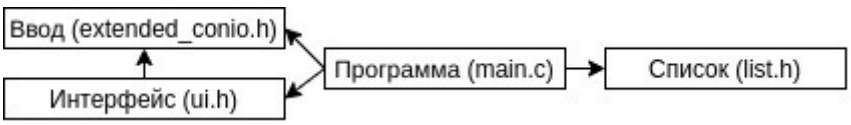


Рисунок 2.1 – Схема взаимодействия частей программы

Внутренние структуры данных:

```
typedef struct {
	char d, m;
	short y;
} Date;

#define FILEDATA_NAME_LEN 32

typedef struct {
	char group_name[6 + 2];
	unsigned int gradebook_number;
	char full_name[FILEDATA_NAME_LEN];
	char gender, education_form;
	Date birth_date, admission_date;
	unsigned short USE_score;
} FileData; // Запись

typedef struct list_element {
	FileData data;
	struct list_element * link[4];
} ListElement; // Элемент

/* Временная структура, применяемая рекурсивной функцией алгорита
сортировки для возврата нескольких занчений */
typedef struct {
	ListElement * first, * last;
	ListElement * next;
} Cut; // Отрезок

#define READ_FUNC_SIGNATURE(name) char (* name)(char enter_dir, short
posx, void * dest, struct field_struct field)
typedef int (* CompareFunc) (void * a, void * b);

/* Структура, применяемая для описания параметров полей элемента
списка. Используется редактором элементов и функциями ввода */
typedef struct field_struct {
	unsigned short posx;   // Позиция на экране
	unsigned char len;     // Длинна на экране
	unsigned char size;    // Размер в памяти
	READ_FUNC_SIGNATURE(read_func); // Указатель на функцию чтения
	CompareFunc comp_func; // Указатель на функцию для сравнения
	char * name;           // Название поля
	size_t offset;         // Смещение относительно начала элемента
	union {
		char values[8];  // Допустимые значения для char'ов
		char allow;      // Допустимые значения для строк
		#define ALLOW_NOTHING 0 // Ничего не разрешать кроме букв
		#define ALLOW_DIGITS 1   // Разрешить цифры
		#define ALLOW_SPECIAL 2  // Разрешить специальные символы
	} prop; // Дополнительные параметры для функций чтения
} Field; // Поле записи
```

```
typedef struct {
	char * name;
	char name_len;
	void (* func)();
} MenuItem; // Пункт

struct {
	int exit, remove, mistake, start, edit;
} quotes_state;
```

Константы:

```
#define MENU_POSY 0
#define EDITOR_POSY 2 // +-1
#define HEADER_POSY 4
#define SCROLL_POSY 5
#define E404_POSY 6

#define AUTOSAVE_FILE_NAME "test.bin"

#define ARROW_UP 72
#define ARROW_DOWN 80
#define ARROW_LEFT 75
#define ARROW_RIGHT 77
#define KEY_ENTER 13
#define KEY_ESC 27
#define KEY_BACKSPACE 8
#define KEY_TAB 9

#define BACKGROUND_WHITE (BACKGROUND_BLUE|BACKGROUND_GREEN|BACKGROUND_RED)
#define BACKGROUND_GRAY BACKGROUND_INTENSITY

#define field(x, l, s, r, c, n, o, p) { x, l, s, r, c, n, offsetof(FileData, o), p }
Field list_element_fields[] = {
	field( 1, 6, 6, read_string, strcmp, "Группа", group_name, { allow: ALLOW_DIGITS }),
	field( 9, 6, sizeof(int), read_fixed_int, intcmp, "Зачетка", gradebook_number, {0}),
	field(19, 1, 1, read_char, chrcmp, "Пол", gender,
{ values: "\2mf" }),
	field(25, 1, 1, read_char , chrcmp, "Форма", education_form, { values: "\3ozd" }),
	field(30, 10, sizeof(Date), read_fixed_date, cmpdte, "Рождение ", birth_date, {0}),
	field(42, 10, sizeof(Date), read_fixed_date, cmpdte, "Поступление", admission_date, {0}),
	field(55, 3, sizeof(short), read_fixed_short, shrcmp, "ЕГЭ", USE_score, {0}),
```

```
            field(60, 32, 32, read_string, strcmp, "ФИО", full_name, { allow:
ALLOW_NOTHING })
      };

      #define item(name, func) { name, len(name) - 1, func }
      MenuItem menu_items[] = {
            item("+", menu_add ),
            item("-", menu_remove),
            item("Edit", menu_edit),
            item("Search", menu_search),
            item("Sort", menu_sort),
            item("Export", menu_export),
            item("Import", menu_import),
            item("Save", empty_func),
            item("Process", menu_process),
            item(" X ", menu_close_search)
      };
```

Функция, печатающая переданную через параметр дату.

```
void print_date(Date date)
```

Вспомогательная функция, перемещающая курсор в указанную позицию (использует WinAPI).

```
void setCursorPosition(SHORT x, SHORT y)
```

Вспомогательная функция, скрывающая курсор при необходимости (использует WinAPI).

```
void setCursorVisibility(char state)
```

Вспомогательная функция, отчищающая строки консоли от и до указанных (использует WinAPI).

```
void clear_lines(short from, short to)
```

Функция, устанавливающая размер буфера консоли таким образом, чтобы он не выходил за рамки окна (использует WinAPI).

```
void adjust_buffer()
```

Функция, восстанавливающая исходный размер буфера (использует WinAPI).

```
void restore_buffer()
```

Вспомогательная функция, устанавливающая цвет. В качестве параметров принимает начальную точку, количество закрашиваемых символов консоли и цвет (использует WinAPI).

```
void setColor(short from_x, short from_y, DWORD _len, WORD attr)
```

Функция, считывающая строку. В качестве параметров принимает начальное положение курсора (`ARROW_LEFT` если в конце поля), позицию поля ввода по оси X (по оси Y позиция задаётся константой `EDITOR_POSY`), указатель на буфер, который будет меняться и описание поля (из которого используются члены `.len` и `.prop.allow`; в соответствии с последним осуществляется фильтрация пользовательского ввода по принципу разрешительной политики). Возвращает код клавиши, из-за которой завершился ввод, который потом может быть использован в качестве `enter_dir` для следующего вызываемого поля ввода.

```
char read_string(char enter_dir, short posx, void * dest, Field field)
```

Функция, считывающая число фиксированного размера, хранимое в `int`. Параметры и возвращаемое значение идентично таковым у функции `read_string`. Из описания поля `field` используется только `.len`.

```
char read_fixed_int(char enter_dir, short posx, void * dest, Field field)
```

Функция, считывающая число фиксированного размера, хранимое в `short`. Параметры и возвращаемое значение идентично таковым у функции `read_fixed_int`, которая и используется для считывания числа, которое потом конвертируется из `int` в `short`.

```
char read_fixed_short(char enter_dir, short posx, void * dest, Field field)
```

Функция, считывающая дату. Параметры и возвращаемое значение идентично таковым у функции `read_string`. Описание поля `field` не используется.

```
char read_fixed_short(char enter_dir, short posx, void * dest, Field field)
```

Функция, считывающая единичный символ. Параметры и возвращаемое значение идентично таковым у функции `read_string`. Из описания поля `field` используется только `.prop.values`, в соответствии с которым осуществляется фльтрация пользовательского ввода на основе разрешительной политики.

```
char read_fixed_short(char enter_dir, short posx, void * dest, Field field)
```

Функция для сравнения переменных типа `int`, передаваемых через указатель типа `void *` для унификации. Указатель на эту функцию передаётся алгоритму сортировки в случае, если поле, по которому требуется произвести сортировку имеет тип `int`.

```
int intcmp(void * a, void * b)
```

Функция для сравнения переменных типа `char`. Получаемые аргументы, возвращаемое значение, и использование аналогично функции `intcmp`.

```
int chrcmp(void * a, void * b)
```

Функция для сравнения переменных типа `short`. Получаемые аргументы, возвращаемое значение, и использование аналогично функции `intcmp`.

```
int shrcmp(void * a, void * b)
```

Функция для сравнения переменных типа `Date`. Получаемые аргументы, возвращаемое значение, и использование аналогично функции `intcmp`.

```
int cmpdte(Date * a, Date * b)
```

Функция, присоединяющая переданный указателем элемент к стеку `freeded_elements`, реализованному в виде односвязного списка. Таким образом, если программе вновь понадобится новый элемент, она не будет вновь вызывать `malloc`.

```
void list_element_free(ListElement * el)
```

Функция, возвращающая указатель на элемент, взятый из стека `freeded_elements` или размещённый в памяти с помощью `malloc`, если тот пуст.

```
ListElement * list_element_new()
```

Функция, выводящая элемент `cur` на экран.

```
void element_print(ListElement * cur)
```

Функция, выводящая запись `cur` в текстовый файл `f`. Возвращает отрицательное значение в случае неудачи.

```
int element_print_to_txt(FILE * f, ListElement * cur)
```

Функция, считывающая запись из текстового файла `f` в элемент `cur`. Возвращает отрицательное значение в случае неудачи.

```
int element_read_from_txt(FILE * f, ListElement * cur)
```

Функция, заполняющая элемент `elem` начальными значениями.

```
void element_zerofill(ListElement * elem)
```

Функция, добавляющая элемент `el` в список (основной или вспомогательный, в зависимости от `link_layer`).

```
void list_add(ListElement * el)
```

Функция, возвращающая размер файла `f` в байтах.

```
size_t get_file_size(FILE * f)
```

Функция, выполняющая первое выделение памяти под записи из файла автосохранения (константа `AUTOSAVE_FILE_NAME`) и элемента `last_readed`, в который производится запись вводимых пользователем данных при добавлении нового элемента и загружающая эти записи в основной список. Память выделяется единой частью, начало и конец которой записываются в глобальные переменные `first_alloc_begin` и `first_alloc_end`.

```
void list_autoload()
```

Функция, выполняющая сохранение списка (основного или вспомогательного, в зависимости от `link_layer`) в файл автосохранения (константа `AUTOSAVE_FILE_NAME`).

```
void list_autosave()
```

Функция, удаляющая элемент `el` из списка (основного или вспомогательного, в зависимости от `link_layer`).

```
void list_remove(ListElement * el)
```

Часть алгоритма сортировки. Функция, выполняющая сравнение полей, расположенным в памяти со смещением относительно начала элемента, указанным глобальной переменной `field_to_compare_by_offset`, элементов `a` и `b` с использованием функции для сравнения, заданной глобальной переменной `list_element_compare_func`, которая может принимать значение указателя типа `CompareFunc` на одну из функций: `strcmp`, `intcmp`, `chrcmp`, `shrcmp`, `dtecmp` или `list_process_cmp`.

```
int list_element_compare(ListElement * a, ListElement * b)
```

Часть алгоритма сортировки. Функция, выполняющая объединение двух отсортированных отрезков `l1` и `l2` в один с использованием функции `list_element_compare` для определения порядка следования.

```
Cut merge(Cut l1, Cut l2)
```

Часть алгоритма сортировки. Функция, выполняющая сортировку `len` (>=2) элементов двухнаправленного списка начиная с элемента `first`. Использует `merge` и возвращает осортированный отрезок.

```
Cut recursion(ListElement * first, int len)
```

Часть алгоритма сортировки. Функция, выполняющая установку глобальных переменных `field_to_compare_by_offset` и `list_element_compare_func`, в зависимости от порядкового номера `field_id` в глобальном массиве `list_element_fields`, используемых функцией

`list_element_compare`, а так же начала и конца списка (основного или вспомогательного, в зависимости от `link_layer`) на первый и последний элемент отрезка, возвращаемого функцией `recursion`.

```
void merge_sort(unsigned char field_id)
```

Функция, выполняющая передачу всех элементов основного списка стеку `freeded_elements` и обнуляющая занчения первого и последнего элементов списка, а так же его длинну.

```
void list_free()
```

Функция, выполняющая фактическое освобождение памяти, занимаемой главным (и воспомогательным, содержащим часть элементов главного) списком, а так же элементами стека `freeded_elements` и элементом `last_readed`. Использует глобальные переменные `first_alloc_begin` и `first_alloc_end` для определения, следует ли вызвать на элементе функцию `free` или память, выделенная при запуске программы уже была освобождена.

```
void list_release_memory()
```

Часть алгоритма обработки списка. Функция, выполняющая копирование основного списка в вспомогательный.

```
void list_copy_to_search_layer()
```

Часть алгоритма обработки списка. Функция, выполняющая сравнение двух переменных типа `FileData` по полям `group_name`, `gender`, `USE_score`, `admission_date` в порядке убывания приоритета. Получаемые аргументы, возвращаемое значение, и использование аналогично функции `intcmp`.

```
int list_process_cmp(FileData * a, FileData * b)
```

Часть алгоритма обработки списка. Функция, выполняющая обработку. Выбирает по 5 юношей и девушек с наивысшим баллом ЕГЭ в каждой группе. Вызывает функцию `list_copy_to_search_layer`, выполняет переключение `link_layer` в режим вспомогательного списка `SEARCH`, выполняет действия, аналогичные функции `merge_sort`, но использует смещение структуры `FileData` относительно начала элемента и `list_process_cmp` в качестве функции сортировки, отсекает лишние элементы списка после каждой пятой записи о юноше или девушке в группе.

```
void list_process()
```

Функция, печатающая во второй сверху строке буфера консоли фразу одной из песен группы «Ария», по смыслу связанную с запуском программы. Использует счётчик `start` из глобальной структуры `quotes_state`, состояние которой предполагалось загружать из файла автосохранения при запуске программы. Определена макросом.

```
void start_quote()
```

Функция, аналогичная функции `start_quote`. Печатает фразу, по смыслу связанную с завершением программы, использует счетчик `start`.

```
void exit_quote()
```

Функция, аналогичная функции `start_quote`. Печатает фразу, по смыслу связанную с редактированием элемента, использует счетчик `edit`.

```
void edit_quote()
```

Функция, аналогичная функции `start_quote`. Печатает фразу, по смыслу связанную с допущенной пользователем ошибкой, использует счетчик `mistake`.

```
void mistake_quote()
```

Функция, аналогичная функции `start_quote`. Печатает фразу, по смыслу связанную с удалением элемента, использует счетчик `remove`.

```
void remove_quote()
```

Функция, задающая переменные, необходимые для работы скролла, присваивающая `el` глобальным переменным `scroll_first_element_on_screen` и `scroll_selected_element`, а так же устанавливающая значение глобальной переменной `scroll_selected_element_pos` в `0`.

```
void scroll_set_head(ListElement * el)
```

Функция, перерисовывающая скролл. Использует переменные, установленные функцией `scroll_set_head`. Подсвечивает выбранный элемент с помощью `setColor` или выводит сообщение о том, что список пуст в строке буфера, заданной константой `E404_POSY`.

```
void redraw_scroll()
```

Функция, инкрементирующая или декриментирующая глобальную переменную `scroll_selected_element_pos` и заменяющая `scroll_selected_element` на предыдущий или следующий в зависимости от направления прокрутки `dir`, которое может принимать значение `UP` или `DOWN`. В случае, если выбранный элемент находится у того края буфера, в сторону которого производится прокрутка, происходит переключение `scroll_first_element_on_screen` на предыдущий или следующий элемент списка (основного или вспомогательного, в зависимости от `link_layer`)

```
void scroll_scroll(Vertical dir)
```

Функция, производящая циклическое считывание полей, указанных в константном глобальном массиве `list_element_fields` в элемент `elem`. В процессе работы вызывает `element_print` и функции, описание которых соответствует `READ_FUNC_SIGNATURE`, указанные в качестве функций для считывания поля в `list_element_fields`. Возвращаемое значение идентично таковому у функции `read_string`.

```
char draw_editor(ListElement * elem)
```

Вспомогательная пустая функция. Используется в качестве заглушки.

```
void empty_func()
```

Функция, перерисовывающая меню. Подсвечивает текущий выбранный пункт, индекс которого хранится в глобальной переменной `selected_menu_item`. Если `link_layer` установлен в режим вспомогательного списка `SEARCH`, то отрисовывается последний пункт меню, иначе – скрывается.

```
void redraw_menu()
```

Функция, вызывающая `setColor` для заполнения фона, затем `redraw_menu`.

```
void draw_menu()
```

Функция, меняющая значение глобальной переменной `selected_menu_item` в зависимости от аргумента `dir`, который может принимать занчение `LEFT` или `RIGHT`, и вызывающая функцию `redraw_menu`. Если `link_layer` не установлен в режим вспомогательного списка `SEARCH`, то выбрать последний элемент меню – не возможно.

```
void scroll_menu(Horizontal dir)
```

Функция, перерисовывающая заголовок таблицы. Использует глобальный константный массив `list_element_fields` для получения имён полей и

константу `HEADER_POSY` для позиционирования по вертикали. Подсвечивает элемент, индекс которого задан аргументом `selected_header_item`.

```
void redraw_header(unsigned char selected_header_item)
```

Функция, вызывающая `setColor` для заполнения фона, затем `redraw_header`.

```
void draw_header(unsigned char selected_header_item)
```

Функция, позволяющая выбрать поле записи с помощью стрелочек влево-вправо на клавиатуре. Возвращает код клавиши, прервавшей выбор (`KEY_ENTER` или `KEY_ESC`), результат записывает по адресу `in_out`, переданному как аргумент.

```
char header_select_column(unsigned char * in_out)
```

Функция, печатающая `str` в строку статуса – последнюю строку консольного буфера.

```
void print_to_status(char * str)
```

Функция, печатающая `str` в строку статуса и вызывающая `mistake_quote`.

```
void print_error_or_mistake(char * str)
```

Функция меню, добавляющая элемент в основной список. Заполняет глобальный элемент `last_readed` путём вызова функции `draw_editor` и если та вернула код `KEY_ENTER`, то добавляет `last_readed` в основной список, после чего присваивает переменной `last_readed` значение, возвращённое функцией `list_element_new`. Если элемент с таким же значением ключевого поля `gradebook_number` уже существует, выводит ошибку через `print_error_or_mistake`. Если список до этого был пуст, вызывает `scroll_set_head`. Если завершилась без ошибок, вызывает `start_quote`.

```
void menu_add()
```

Функция меню, удаляющая выбранный элемент. Если список пуст, выводит ошибку через `print_error_or_mistake`, иначе удаляет элемент из основного и вспомогательного списков и устанавливает те же глобальные переменные, что и `scroll_set_head` на предыдущий элемент, если это возможно. Если завершилась без ошибок, вызывает функции `redraw_scroll` и `remove_quote`.

```
void menu_remove()
```

Функция меню, редактирующая выбранный элемент. Если список пуст, выводит ошибку через `print_error_or_mistake`, иначе последовательно вызывает функции `draw_editor`, `redraw_scroll` и `edit_quote`.

```
void menu_edit()
```

Функция меню, сортирующая список. Если список содержит менее двух элементов, выводит ошибку, иначе выбирает поле для сортировки вызовом функции `header_select_column`, сортирует вызовом `merge_sort`, обновляет те же глобальные переменные, что и `scroll_set_head`, а потом вызывает `redraw_scroll` и `start_quote`.

```
void menu_sort()
```

Функция меню, осуществляющая поиск в списке, для чего вызывает `header_select_column` для выбора поля, по которому производить поиск, считывает искомое значение поля вызовом указанной для этого `read_func` в описании поля из `list_element_fields`, и режим поиска, который может принимать значения `'o'` – перезаписать (по умолчанию), `'+'` – добавить к существующему вспомогательному списку, `'-'` – удалить совпавшие элементы

из вспомогательного списка. Если ввод данных не был прерван пользователем нажатием KEY_ESC, производит поиск и последовательно вызывает redraw_menu, scroll_set_head, redraw_scroll и start_quote.

```
void menu_search()
```

Функция меню для выхода из режима поиска. Переключает link_layer в режим основного списка SHOW, устанавливает переменные, необходимые для корректной работы скролла вызовом функции scroll_set_head с первым элементом основного списка в качестве аргумента, обновляет меню и скролл вызовом redraw_menu и redraw_scroll, вызывает start_quote.

```
void menu_close_search()
```

Вспомогательная функция, производящая считывание имени файла с помощью вызова функции read_string с разрешением на ввод специальных символов (константы ALLOW_DIGITS и ALLOW_SPECIAL). Возвращает код клавиши, прервавшей ввод.

```
char read_filename(char * filename)
```

Функция меню, экспортирующая список (основной или вспомогательный) в текстовый файл, имя которого считывается вызовом read_filename вызовом element_print_to_txt. В случае возникновения ошибки, она выводится через print_error_or_mistake, иначе по завершении сообщение об успешном окончании операции выводится через print_to_status, после чего вызывается start_quote.

```
void menu_export()
```

Функция меню, импортирующая основной список из текстового файла (предварительно сбросив и основной и вспомогательный вызовом list_free и переключив link_layer в SHOW). Имя файла считывается вызовом read_filename. После считывания списка, которое производится вызовами element_read_from_txt и list_add (получающих last_readed в качестве аргумента, который потом заменяется на значение возвращаемое функцией list_element_new). Далее первый элемент списка устанавливается в качестве первой записи, отображаемой на экране с помощью скролла, через вызов scroll_set_head, после чего скролл обновляется через redraw_scroll. Завершается так же как и menu_export.

```
void menu_import()
```

Функция меню, осуществляющая обработку списка вызовом функции list_process, после чего обновляет элементы интерфейса через redraw_menu, start_quote, scroll_set_head и redraw_scroll.

```
void menu_process()
```

Главная функция. Выполняет инициализацию, получая значения глобальных переменных stdout_handle и cursor_info, используемых для отрисовки, вызовами GetStdHandle и GetConsoleCursorInfo (WinAPI), вызывая adjust_buffer для настройка буфера, устанавливая scroll_last_line как предпоследнюю строку буфера, вызывая list_autoload для загрузки списка из файла автосохранения, а так же draw_menu, draw_header, draw_scroll и start_quote для начальной отрисовки интерфейса.

После в цикле считывается пользовательский ввод. В случае нажатия клавиш-стрелочек вызываются функции scroll_scroll и menu_scroll для

вертикальных и горизонтальных клавиш соответственно. В случае нажатия клавиши `KEY_ENTER`, управление передаётся выбранному из `menu_items` пункту меню. Если же нажата клавиша `KEY_ESC`, программа выходит из цикла, отчищает экран (`clear_lines`), восстанавливает исходный размер буфера (`restore_buffer`), параметры курсора (`setCursorVisibility`), сохраняет основной список в файл автосохранения (`list_autosave`), освобождает память (`list_release_memory`), печатает цитату (`exit_quote`) и завершается.

```
int main()
```

2.5 Описание алгоритмов функционирования программы

Главная функция (рисунок 2.1)

Блок 1 – Выделение памяти и загрузка списка из файла автосохранения, обнуление вместилища последнего считанного элемента `last_readed`.

Блок 2 – Настройка буфера консоли и отрисовка интерфейса.

Блок 3 – Установка необходимых для этого значений и отрисовка скролла.

Блок 4 – Считывание кода клавиши.

Блок 5 – Если получен дополнительный код.

Блок 6 – Считывание кода клавиши.

Блок 7 – Если была нажата стрелочка вверх.

Блок 8 – Прокрутка скролла вверх.

Блок 9 – Иначе, если была нажата стрелочка вниз.

Блок 10 – Прокрутка скролла вниз.

Блок 11 – Иначе, если была нажата стрелочка влево.

Блок 12 – Выбор предыдущего пункта меню.

Блок 13 – Иначе, если была нажата стрелочка вправо.

Блок 14 – Выбор следующего пункта меню.

Блок 15 – Иначе, если был получен не дополнительный код, а код Enter.

Блок 16 – Вызов функции, соответствующей выбранному пункту меню.

Блок 17 – Если был получен код клавиши Escape.

Блок 18 – Вывод прощальной цитаты, восстановление исходных настроек буфера, автосохранение списка, высвобождение всей одолженной памяти.

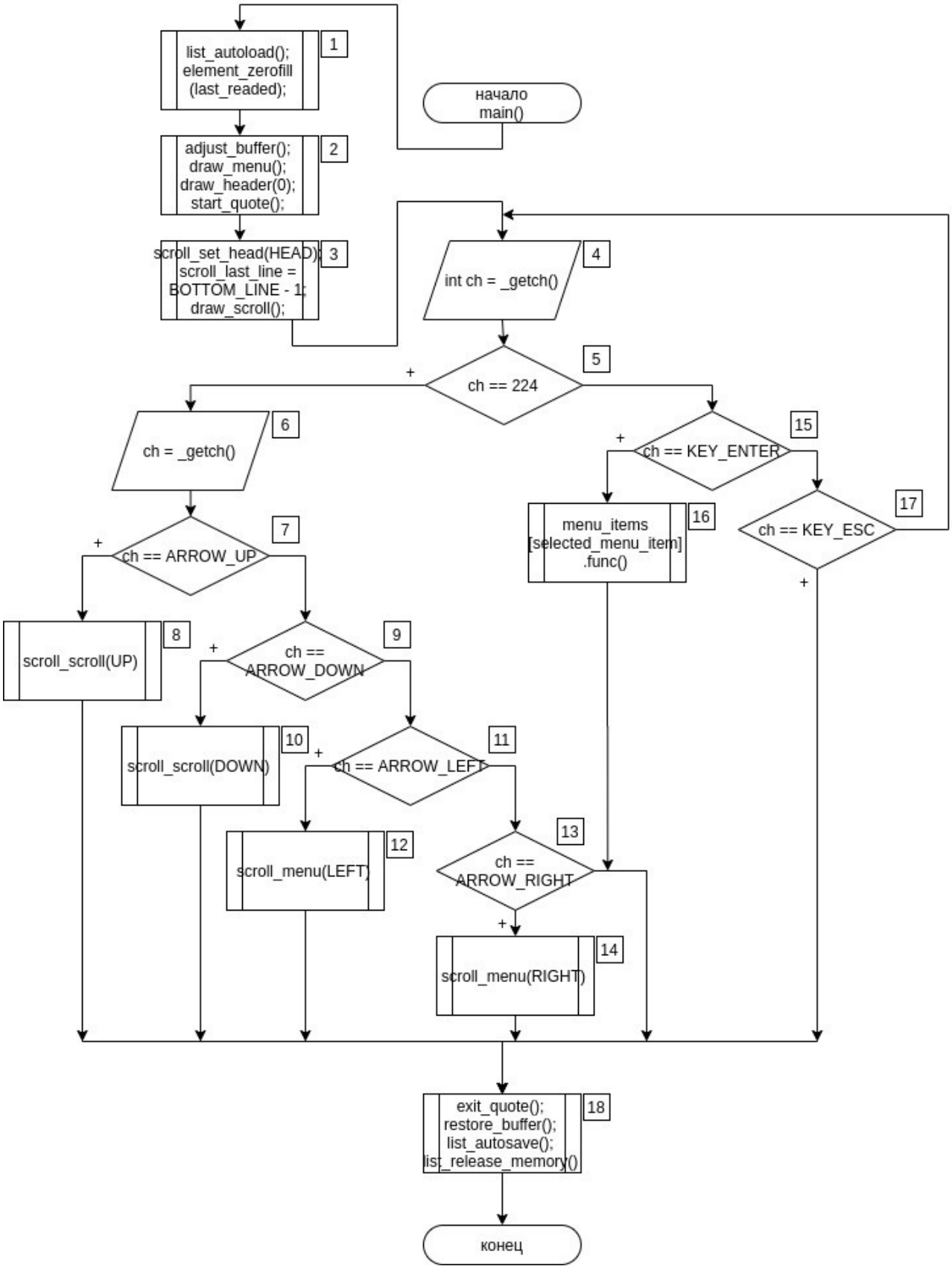


Рисунок 2.1 – Главная функция

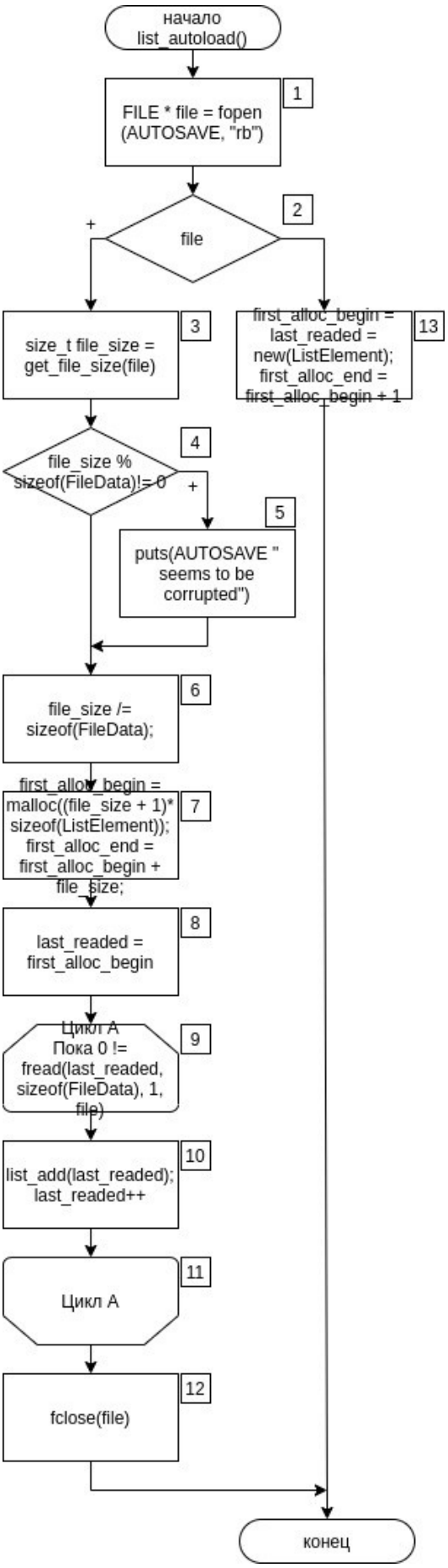


Рисунок 2.2 – Функция автозагрузки

Функция автозагрузки списка и первого выделения памяти.
Блок 1 – Попытка открытия файла автосохранения.
Блок 2 – Если попытка была удачной.
Блок 3 – Вычисление размера файла
Блок 4 – Если размер файла не делится нацело на размер записи.
Блок 5 – Вывод ошибки.
Блок 6 – Вычисление количества записей в файле
Блок 7 – Выделение памяти с запасом на 1 элемент под `last_readed` и вычисление адреса конца выделенного участка памяти.
Блок 8 – Задание начала выделенной памяти как места, куда будет считываться первый элемент.
Блок 9 – Считывание элементов, пока это не возможно.
Блок 10 – Добавление считанного элемента в список, смещение указателя на свободную для использования память.
Блок 11 – Окончание цикла.
Блок 12 – Закрытие файла.

Функция перерисовки скролла (рисунок 2.3).
Блок 1 – Установка курсора на первую строку отведённую под скролл.
Блок 2 – Если список не пуст.
Блок 3 – Установка текущего элемента на первый элемент на экране и сброс количества выведенных записей в 0.
Блок 4 – Пока есть, что выводить и не достигнута нижняя строка.
Блок 5 – Если отрисовываемый элемент выбран.
Блок 6 – Печать записи на белом фоне.
Блок 7 – Иначе, просто печать.
Блок 8 – Следующий элемент становится текущим
Блок 9 – Окончание цикла.
Блок 10 – Очистка незадействованных строк.

Функция меню для сортировки списка (рисунок 2.4).
Блок 1 – Если в списке более одного эллемента.
Блок 2 – Если пользователь прервал выбор поля для сортировки.
Блок 3 – Иначе, сортировка массива слиянием.
Блок 4 – Установка выбранного элемента в качестве тукущего.
Блок 5 – Для i от 0 до позиции выбранного элемента на экране.
Блок 6 – Если это не первый элемент.
Блок 7 – Предыдущий становится текущим.
Блок 8 – Иначе, позиция выбранного элемента на экране равна i.
Блок 9 – Конец цикла.
Блок 10 – Текущий элемент становится первым элементом на экране.
Блок 11 – Вывод приветственной цитаты и перерисовка скролла.
Блок 12 – Вывод уведомления об успешном завершении операции в статус.

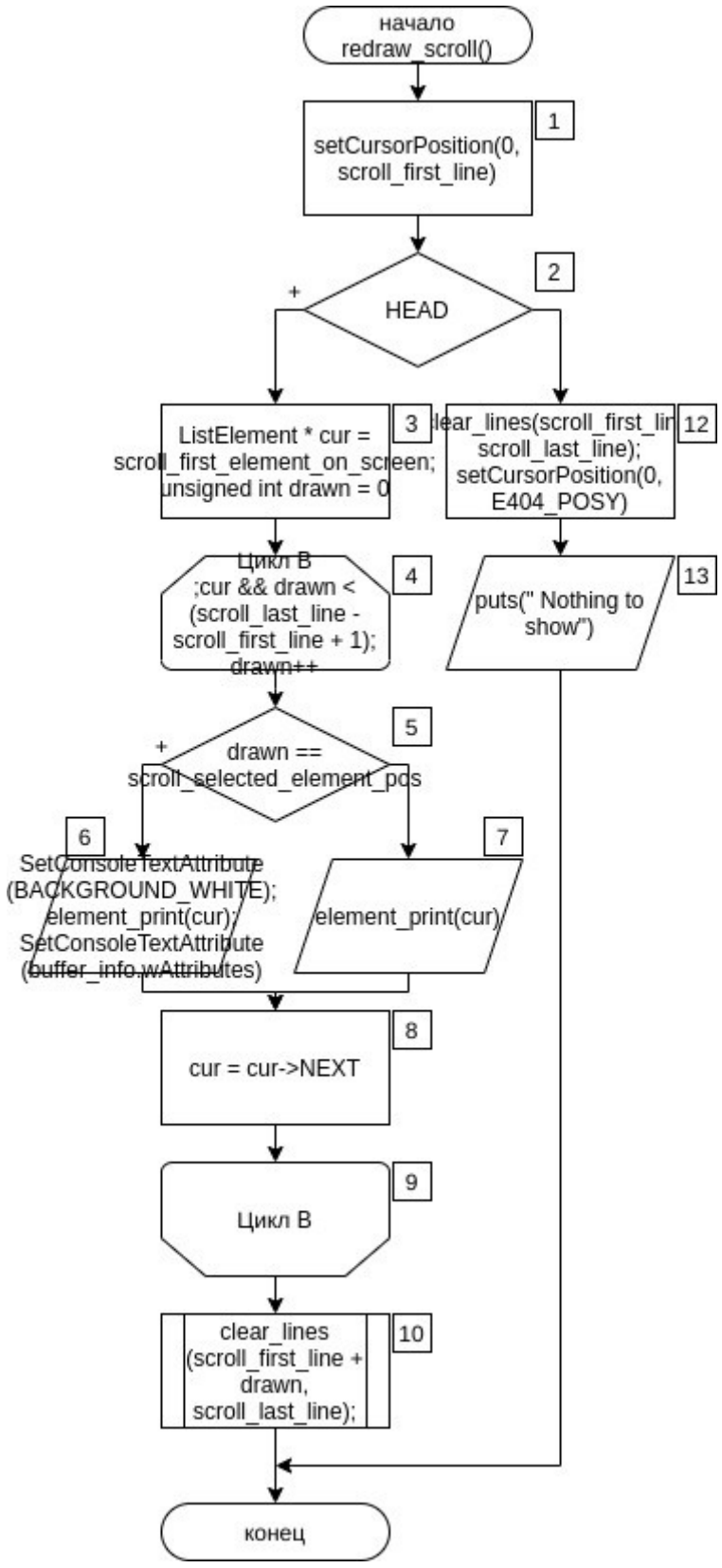


Рисунок 2.3 – Функция перерисовки скролла

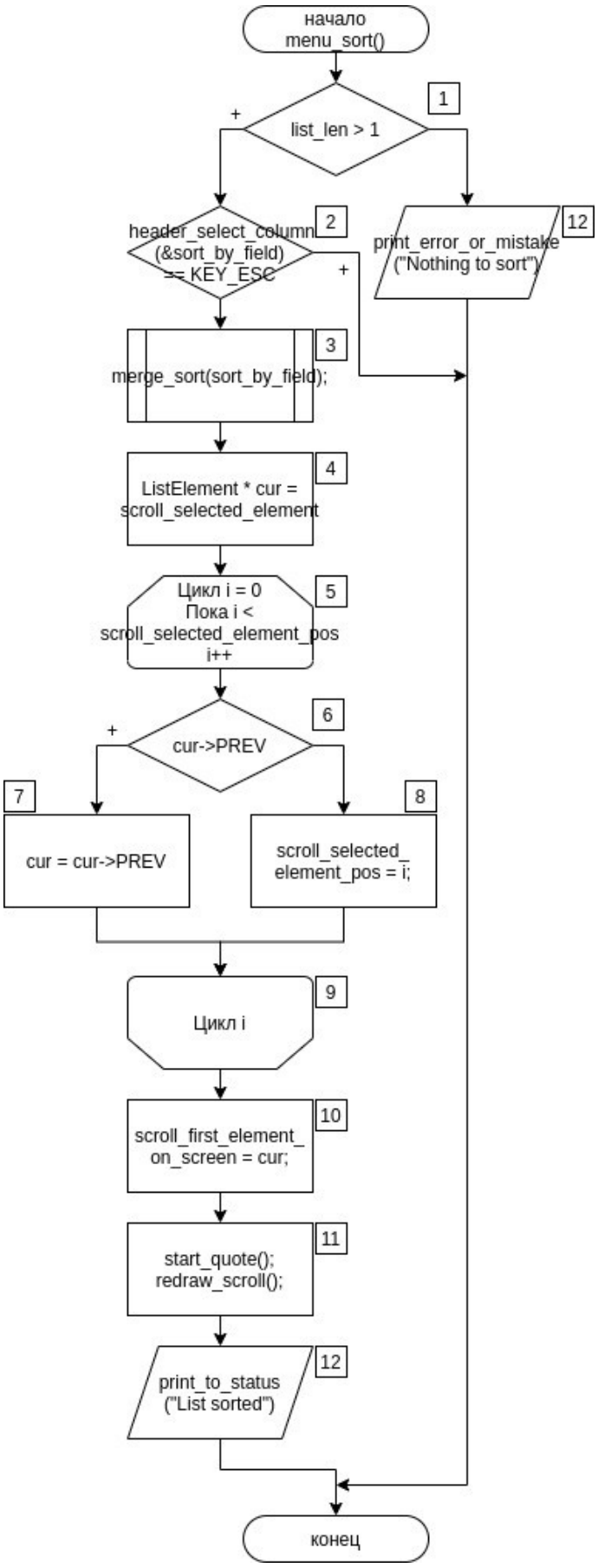


Рисунок 2.4 – Функция меню для сортировки списка

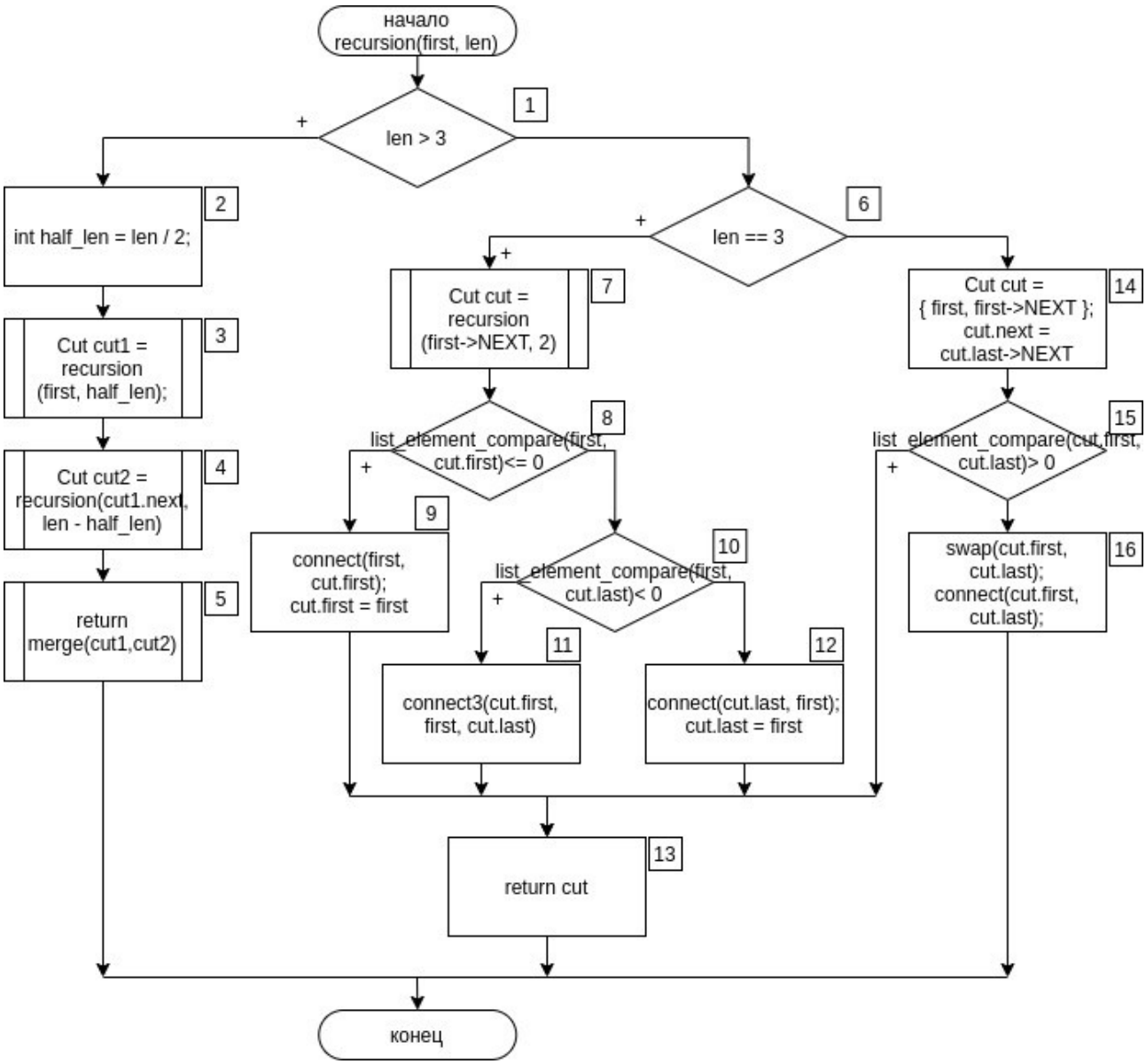


Рисунок 2.5 – Часть алгоритма сортировки

Часть алгоритма сортировки. Рекурсивная функция.

Блок 1 – Если длинна сортируемого отрезка больше трёх.

Блок 2 – Вычисление половины длинны.

Блок 3 – Вызов самой себя для первой половины отрезка.

Блок 4 – Вызов самой себя для второй половины отрезка.

Блок 5 – Возврат объединённого из результатов вызовов отрезка.

Блок 6 – Если длинна сортируемого отрезка равна трём.

Блок 7 – Вызов самой себя для двух последних элементов отрезка.

Блок 8 – Если первый элемент отрезка меньше или равен первому из двух последних элементов после их сортировки.

Блок 9 – Тогда первый элемент остаётся таковым.

Блок 10 – Иначе, если первый элемент меньше последнего из двух последних элементов после их сортировки.

Блок 11 – Первый элемент становится вторым.

Блок 12 – Иначе, первый элемент становится третьим.

Блок 13 – Возврат отсортированного отрезка.

Блок 14 – Если в сортируемом отрезке всего 2 элемента, из них формируется возвращаемый отрезок.

Блок 15 – Если первый элемент получившегося отрезка больше второго.

Блок 16 – Первый и второй элемент меняются местами.

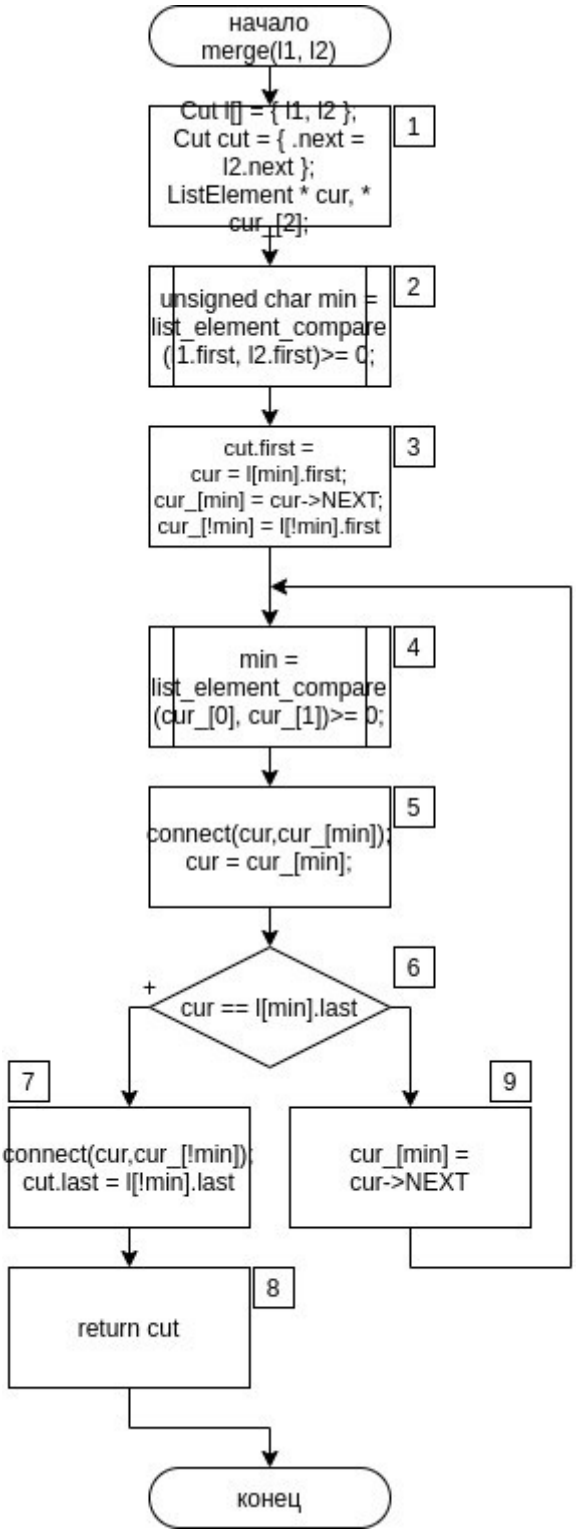


Рисунок 2.6 – Часть алгоритма сортировки

Часть алгоритма сортировки. Функция, выполняющая объединение.

Блок 1 – Создание результирующего отрезка.

Блок 2 – Определение массива с наименьшим первым элементом.

Блок 3 – Первым элементом результирующего массива, а так же его текущим становится первый элемент определённого в блоке 2 массива, текущим элементом для этого массива становится его второй элемент, а для другого – первый элемент другого массива.

Блок 4 – Определение массива с наименьшим первым элементом.

Блок 5 – Присоединение текущего элемента определённого блоком 4 отрезка к текущему элементу результирующего и становление им.

Блок 6 – Если последний присоединённый элемент был последним в своём отрезке.

Блок 7 – К нему присоединяются остатки второго отрезка и последний элемент второго отрезка остаётся таковым и в результирующем.

Блок 8 – Возвращение результирующего отрезка.

Блок 9 – Текущим для определённого блоком 4 отрезка становится следующий.

2.6 Обоснование состава технических и программных средств

Для работы программы рекомендуется использовать 64-разрядный компьютер архитектуры IBM PC с подключённым экраном, клавиатурой, под управлением ОС Windows 7 или старше или любой другой ОС с установленным Wine версии не менее 5.19. Компьютер должен соответствовать минимальным требованиям, необходимым для запуска упомянутых операционных систем, иметь 93КБ дискового или иного пространства для хранения исполняемого файла и 5.6МБ свободной оперативной памяти для запуска. Каждая новая запись будет занимать 72Б (размер струкруры `ListElement`). Учитывая, что ключевое поле является шестизначным десятичным числом, максимальный объём памяти, который могут занять записи приблизительно равен 69МБ.

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Условия выполнения программы

Т.к. программа изначально разрабатывалась для исполнения под ОС Windows и требует сравнительно ничтожных ресурсов, системные требования к компьютеру, на котором может осуществляться её запуск соответствуют системным требованиям ОС, приведённым в таблице 3.1.

Таблица 3.1 – Требования ОС Windows 10 x64

Процессор	Не менее 1 ГГц или SoC
ОЗУ	2 ГБ
Место на жестком диске	20 ГБ
Видеоадаптер	DirectX 9 или более поздняя версия с драйвером WDDM 1.0
Экран	800 x 600

3.2 Загрузка и запуск программы

Для запуска программы достаточно запустить в консоли исполняемый файл main.exe в папке, в которой она сможет создать файл автосохранения.

3.3 Проверка работоспособности программы

Рисунки с 3.1 по 3.24 описывают последовательность действий, демонстрирующих работу программы, и их результаты, они были снабжены подписями и сочтены достаточно экспрессивными.

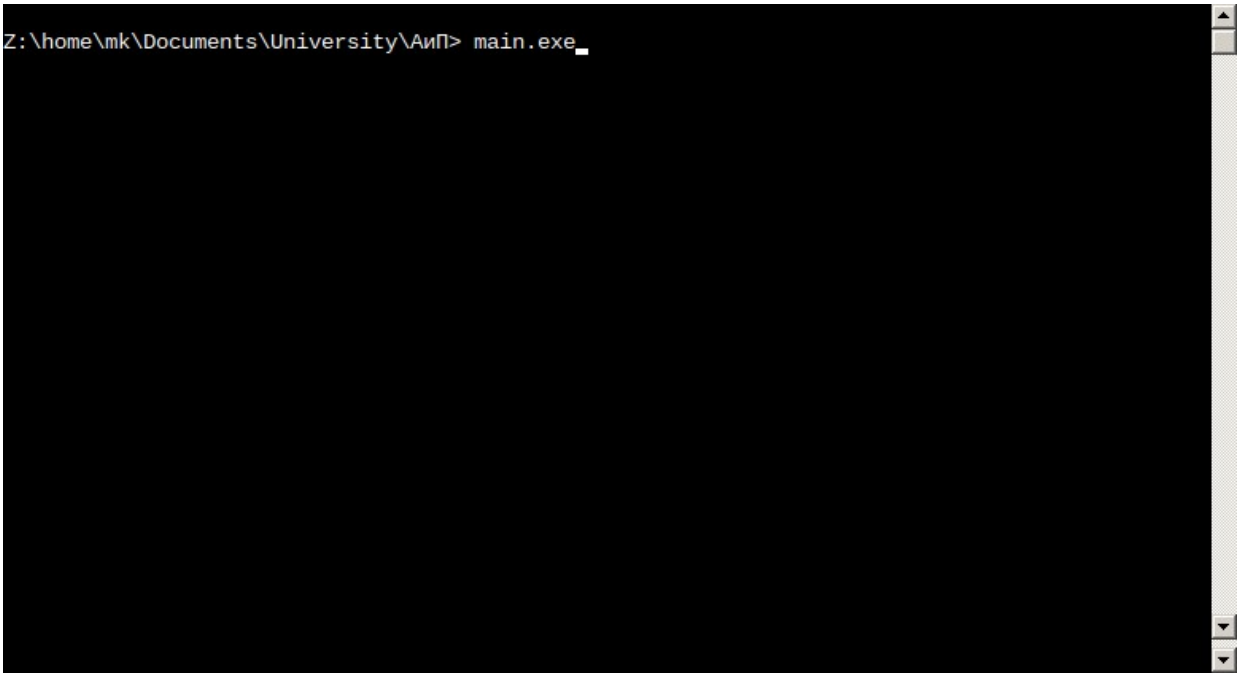


Рисунок 3.1 – Запуск программы из консоли

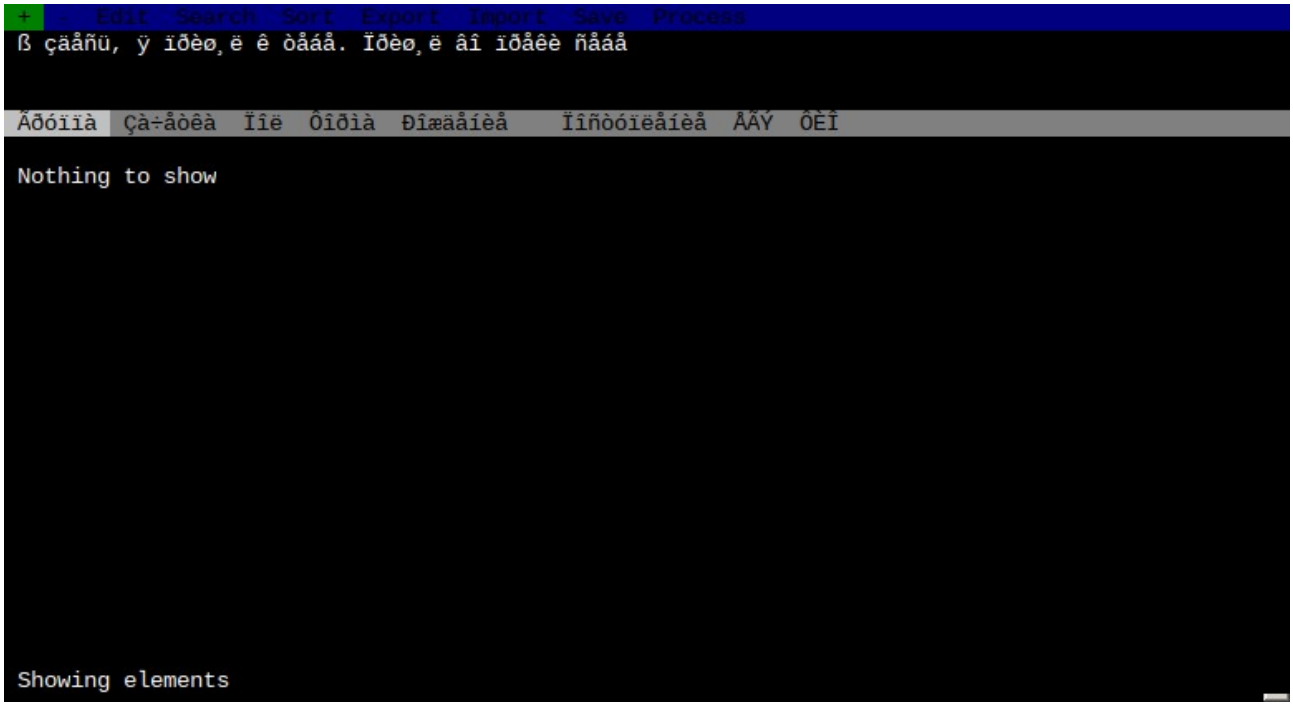


Рисунок 3.2 – Главный экран

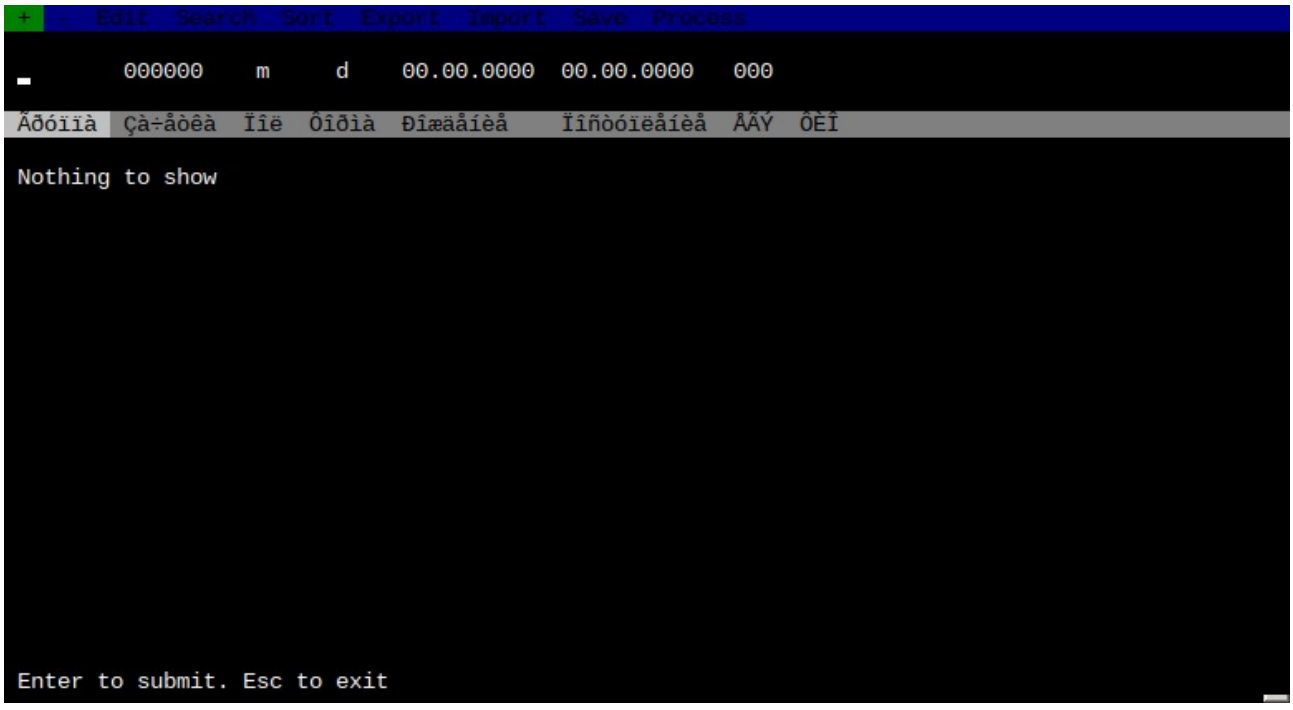


Рисунок 3.3 – Добавление элемента

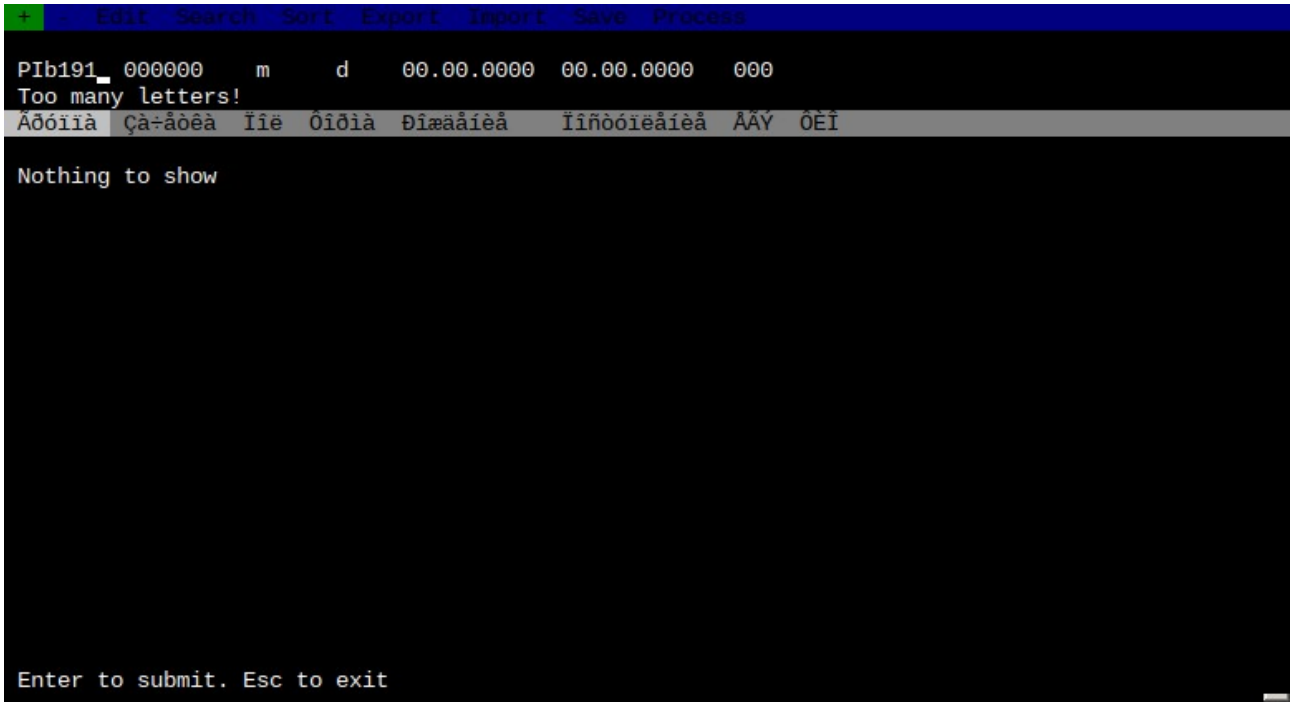


Рисунок 3.4 – Ошибка ввода

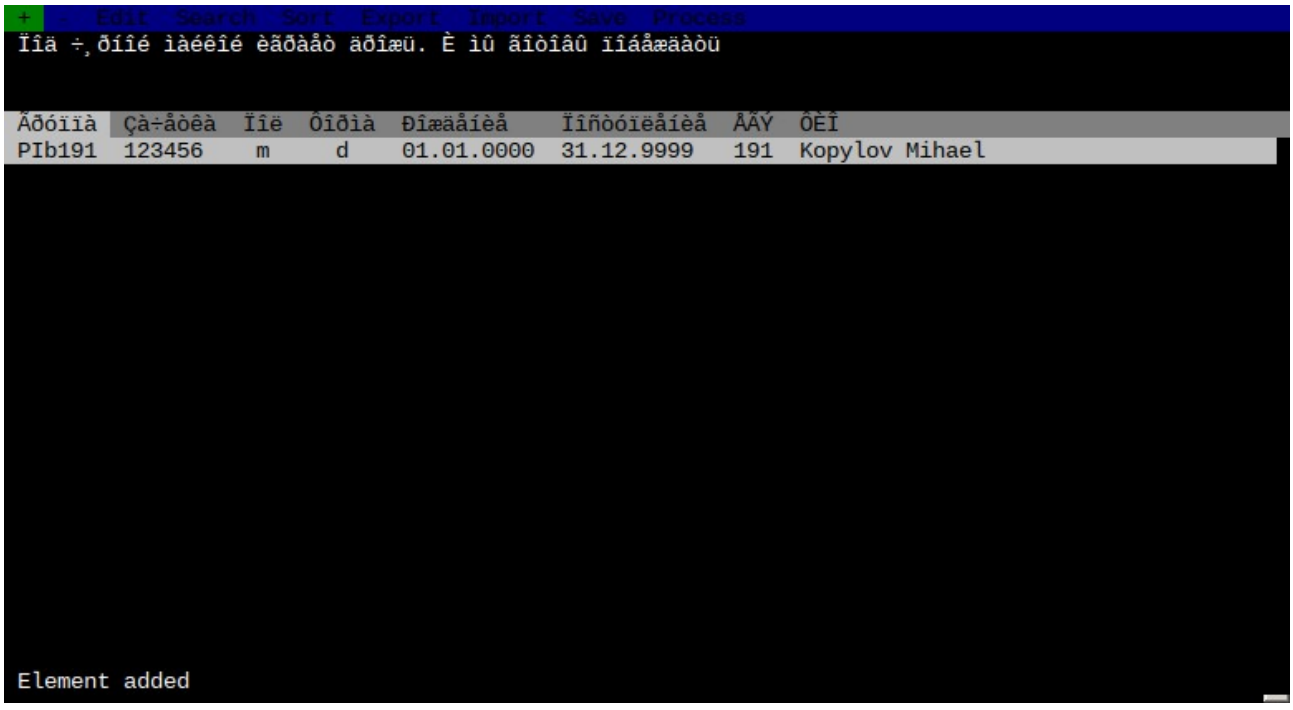


Рисунок 3.5 – Результат добавления

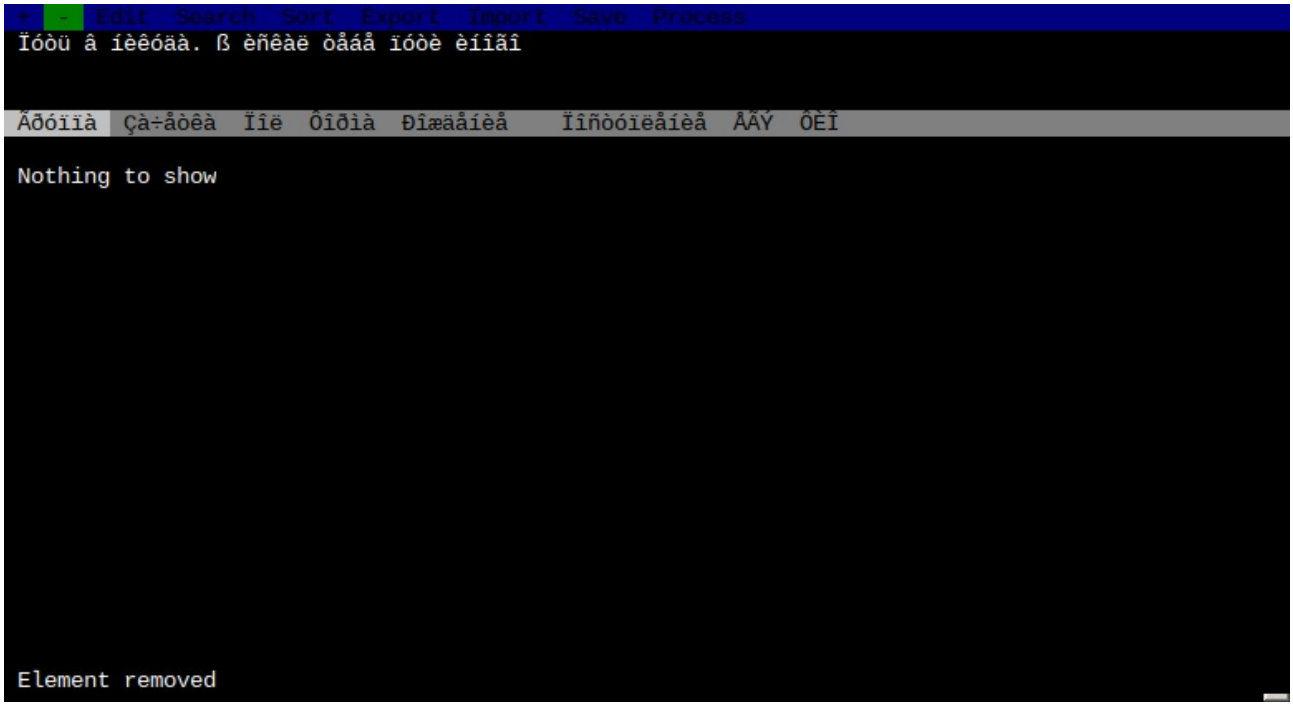


Рисунок 3.6 – Результат удаления

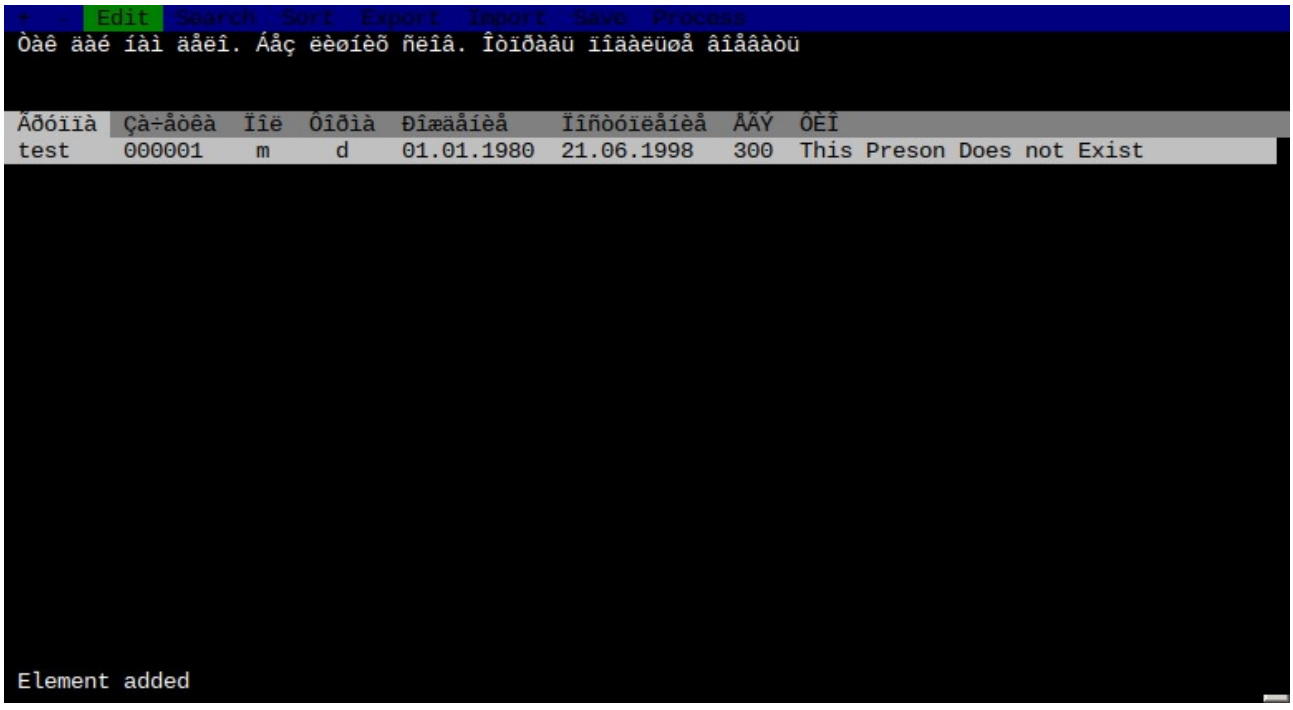


Рисунок 3.7 – Новый элемент

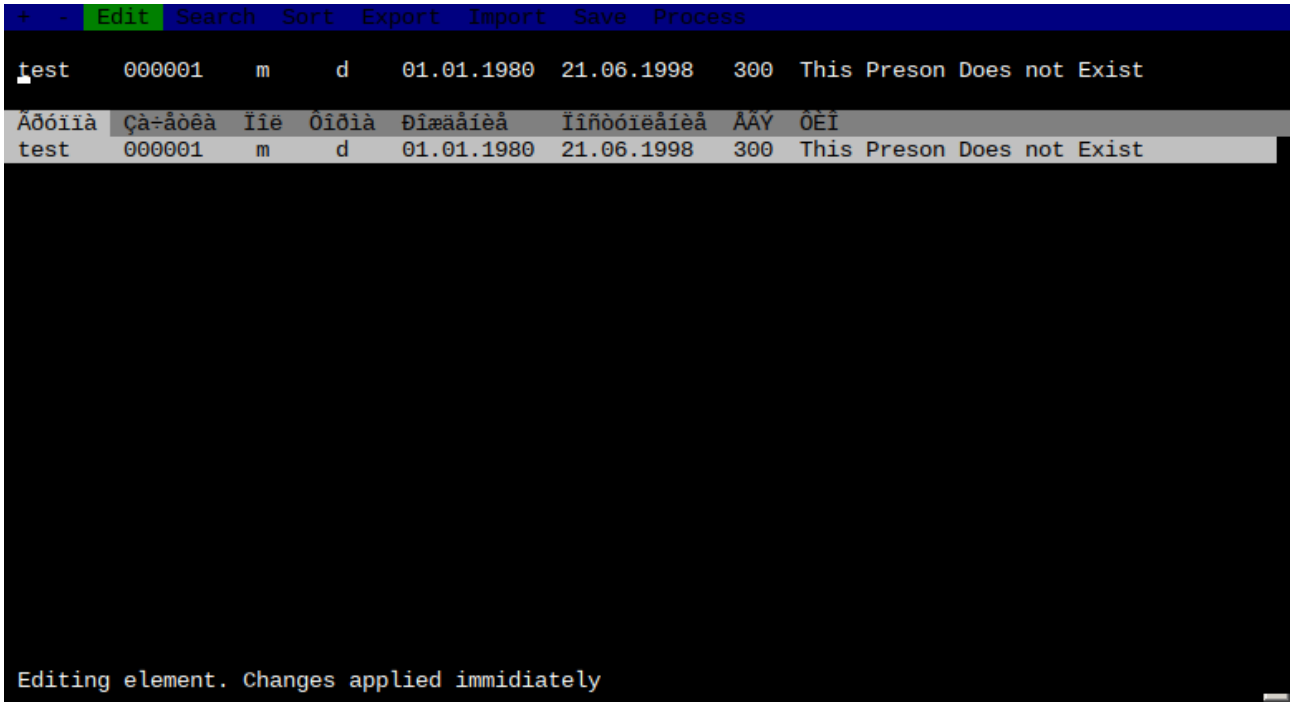


Рисунок 3.8 – Редактирование элемента

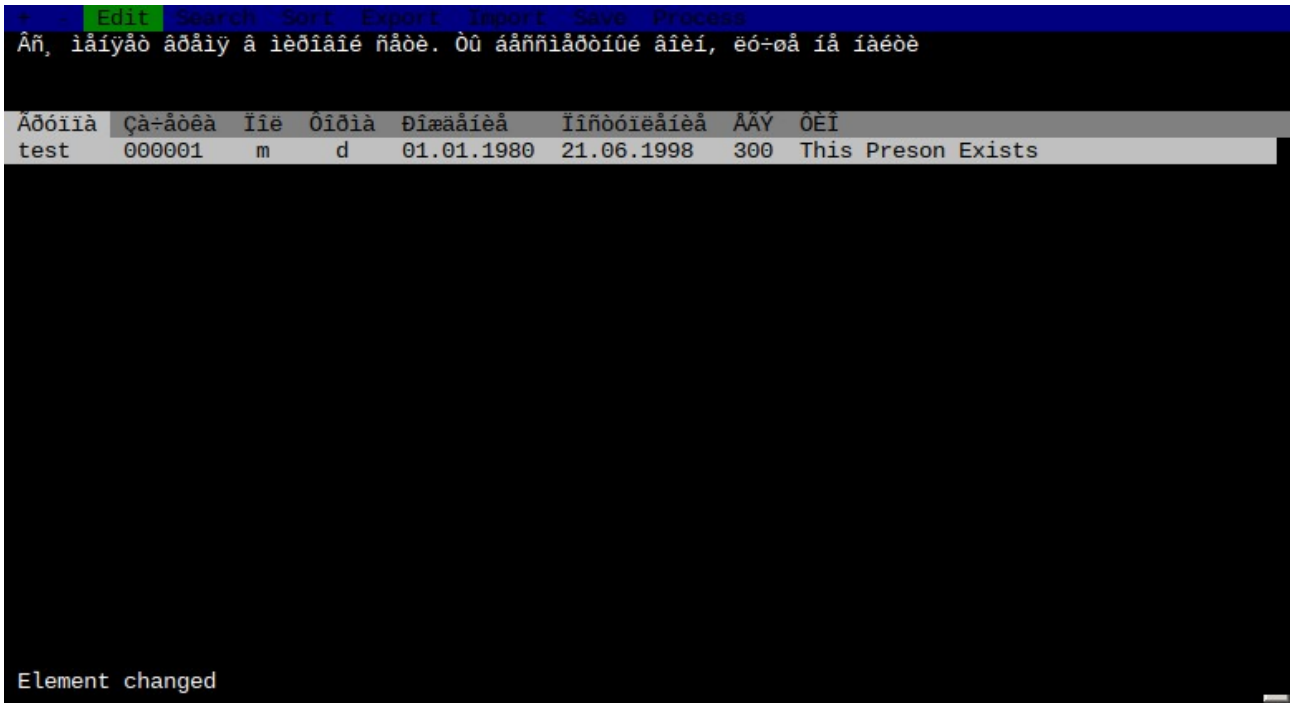


Рисунок 3.9 – Результат редактирования

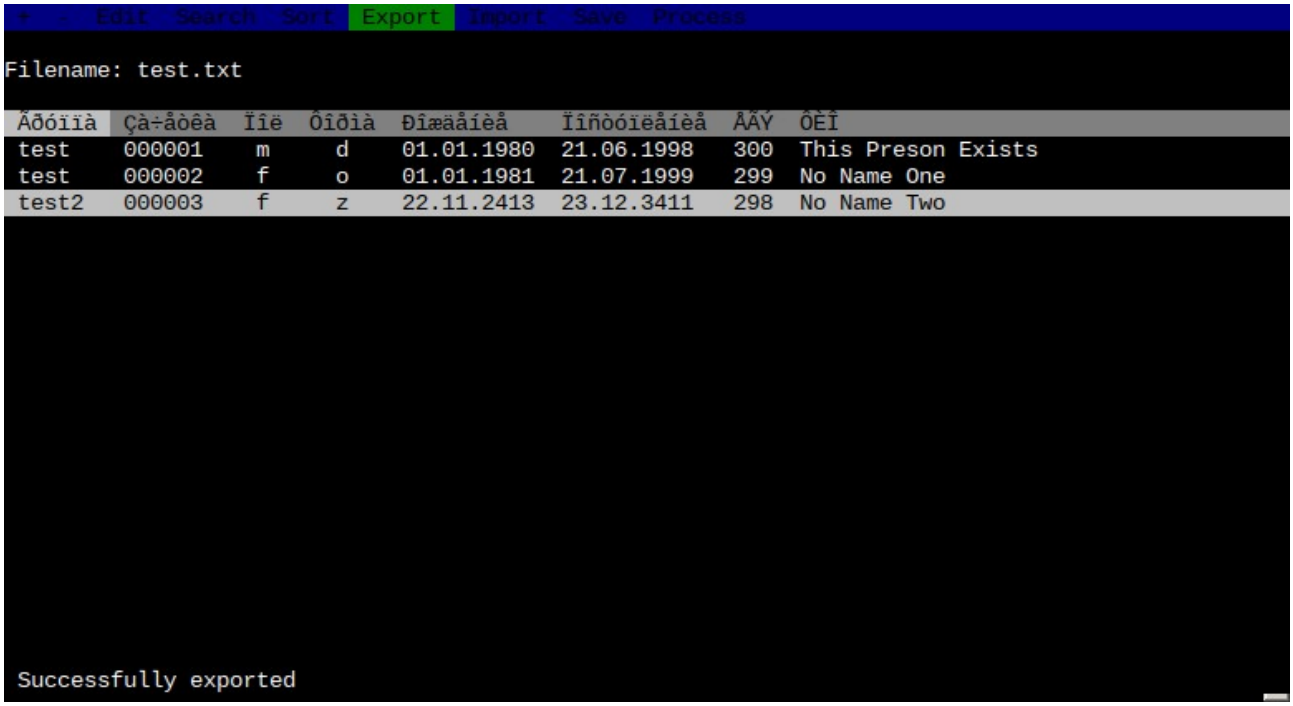


Рисунок 3.10 – Экспорт элементов в текстовый файл

```
test   000001 m d 01.01.1980 21.06.1998 300 This Preson Exists
test   000002 f o 01.01.1981 21.07.1999 299 No Name One
test2  000003 f z 22.11.2413 23.12.3411 298 No Name Two
```

Рисунок 3.11 – Содержимое файла test.txt

После файл был отредактирован таким образом, чтобы он содержал больше уникальных записей.

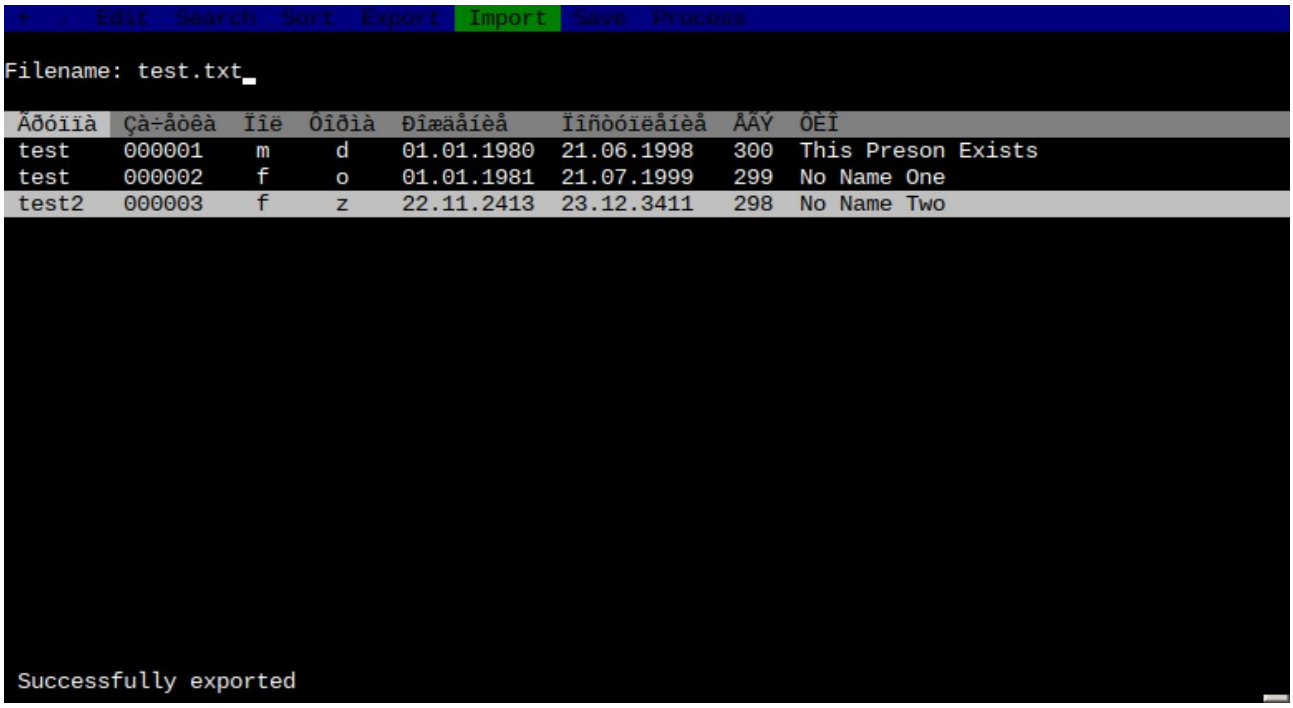


Рисунок 3.12 – Импорт элемента из текстового файла

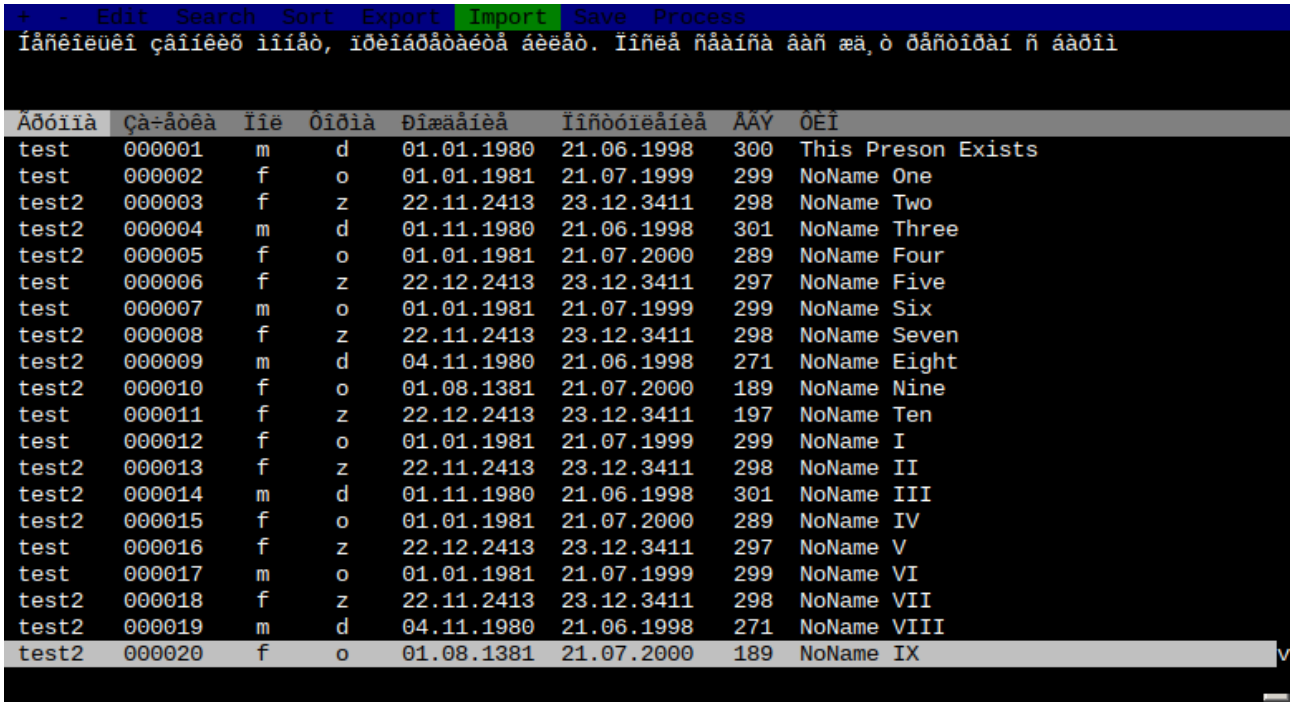


Рисунок 3.13 – Результат импорта и скролл до прокрутки

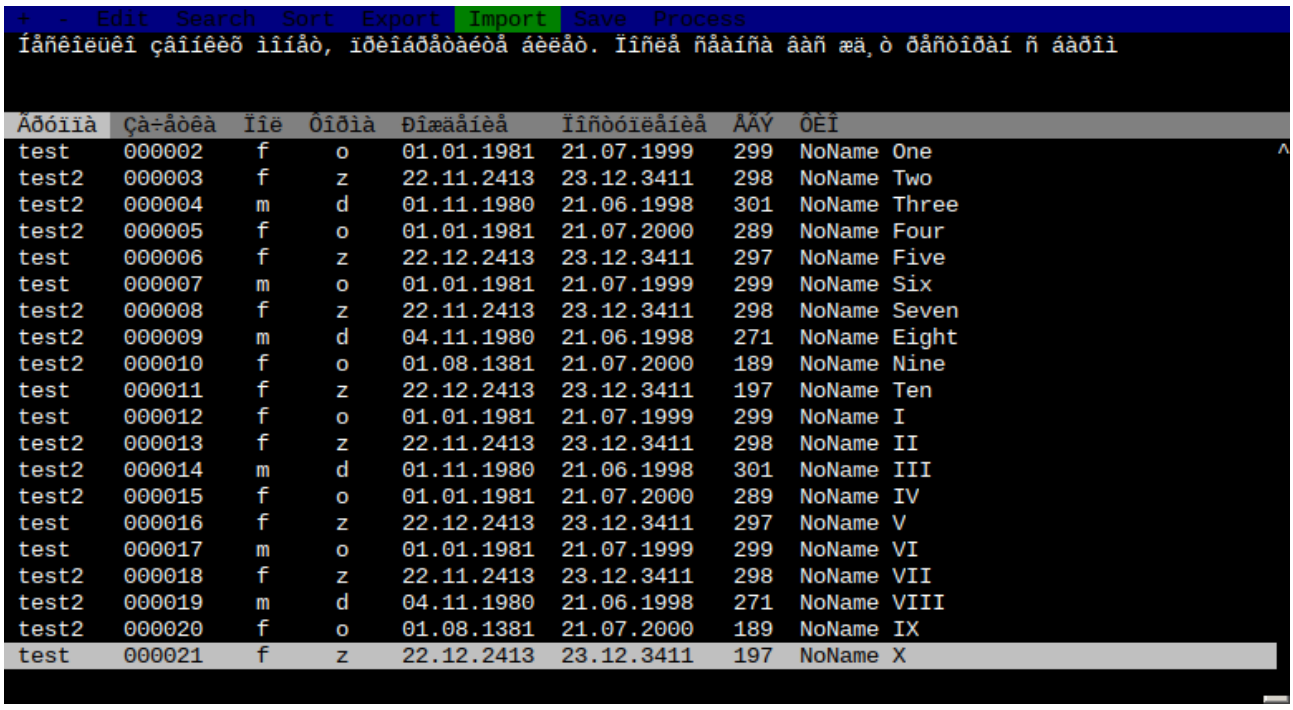


Рисунок 3.14 – Скролл после прокрутки

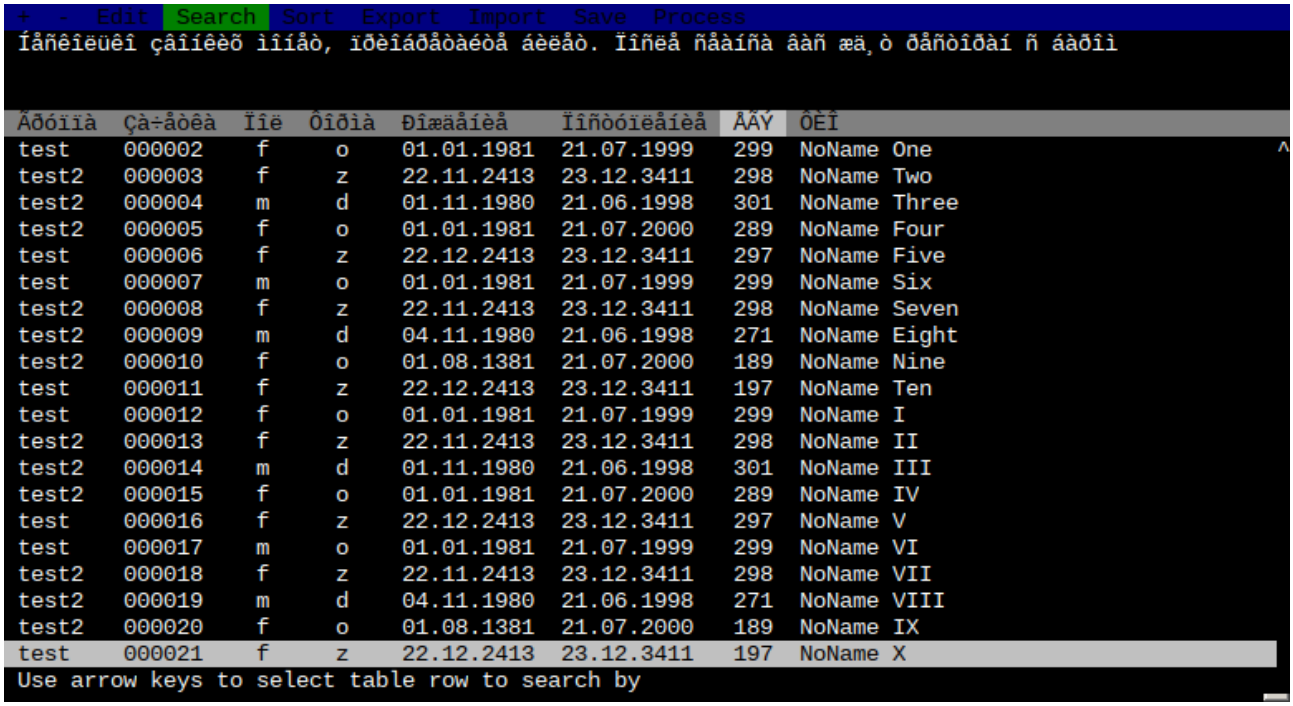


Рисунок 3.15 – Выбор поля для ведения поиска

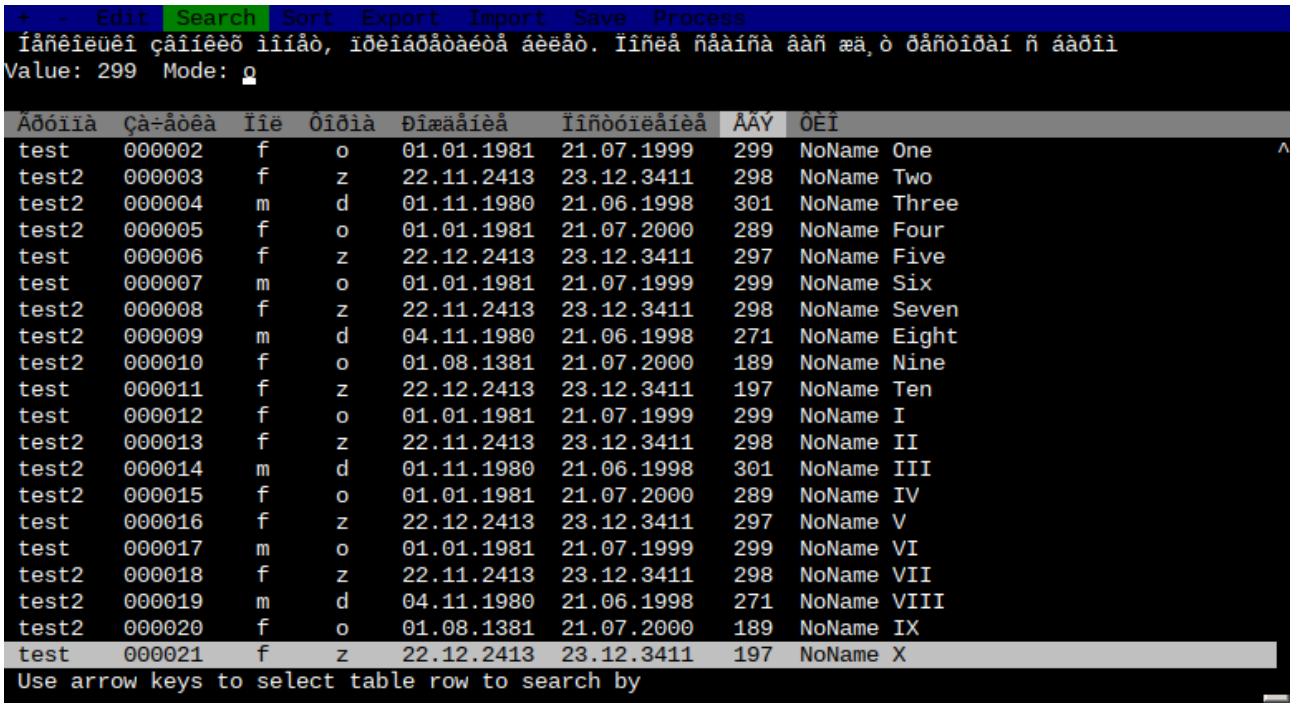


Рисунок 3.16 – Введение искомого значения

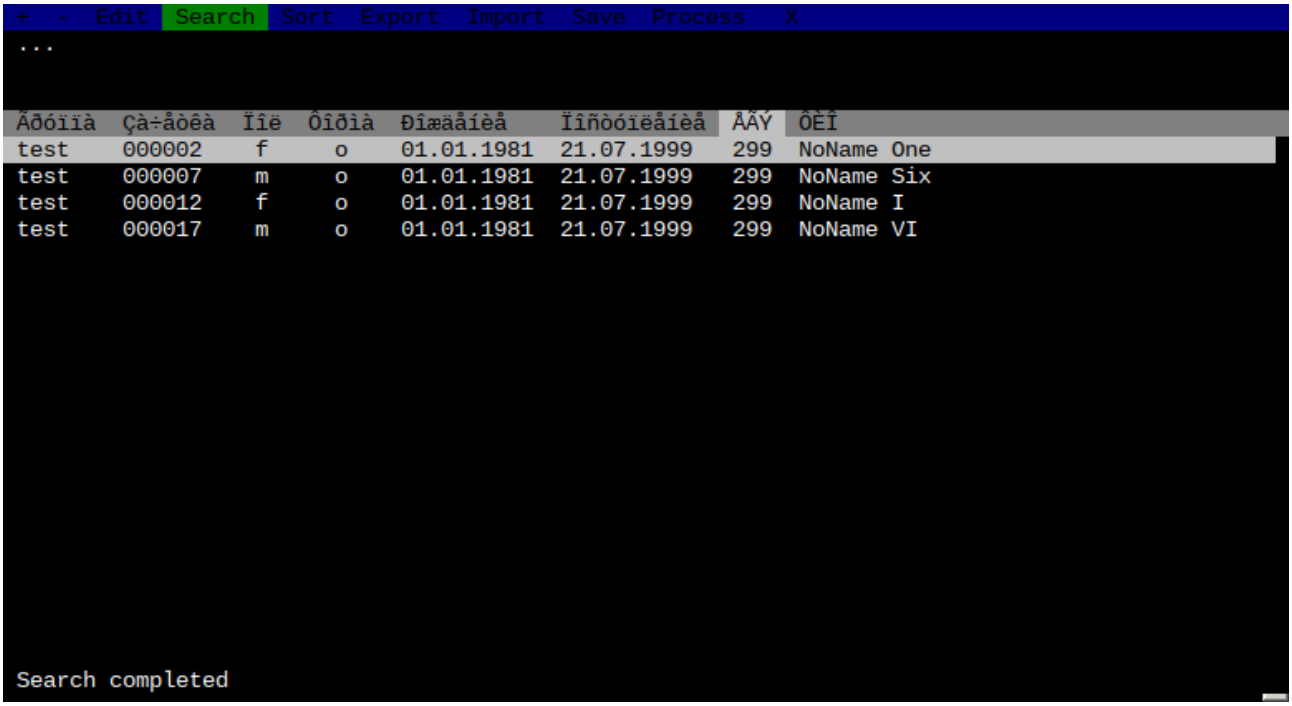


Рисунок 3.17 – Результаты поиска

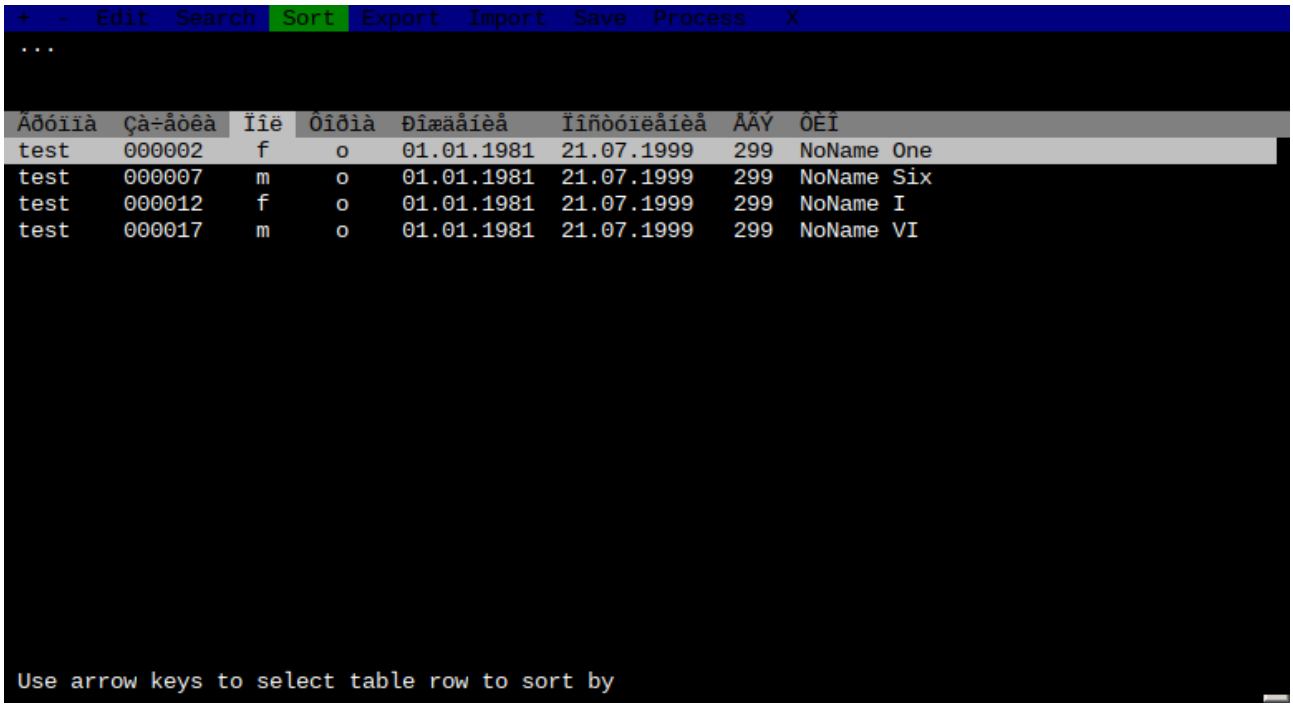


Рисунок 3.18 – Выбор поля для сортировки

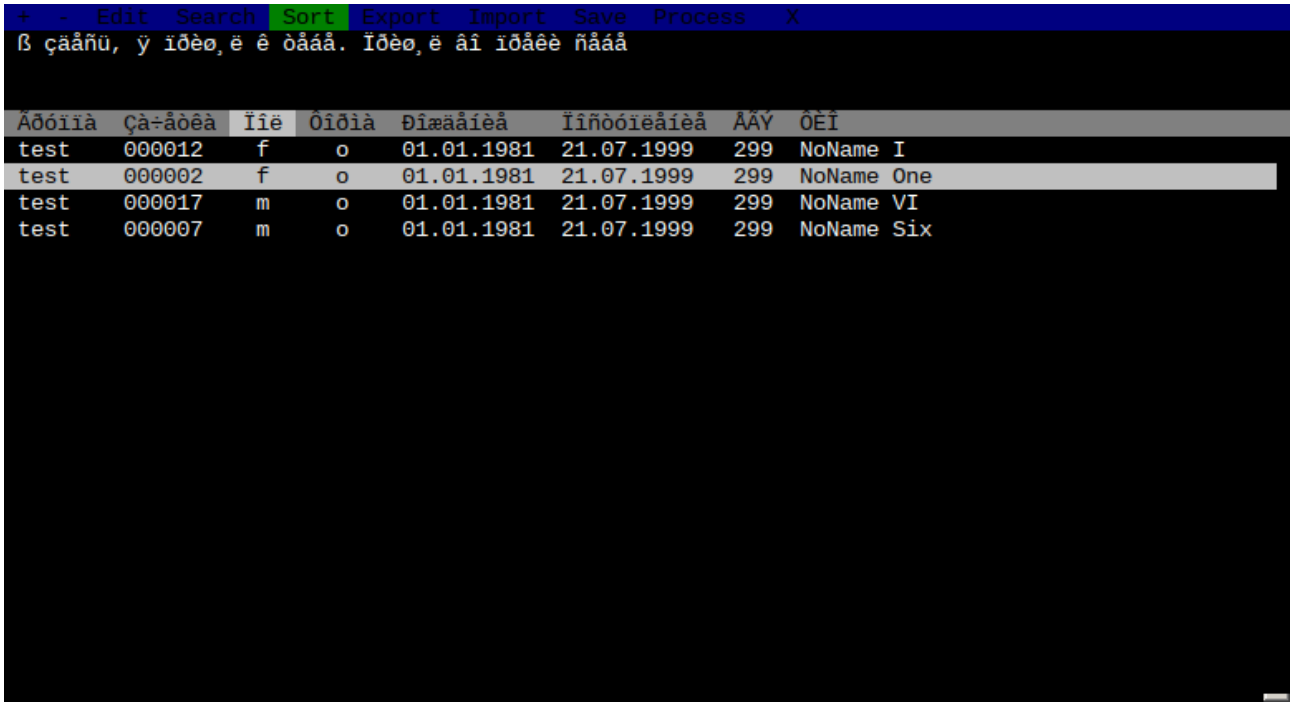


Рисунок 3.19 – Результат сортировки

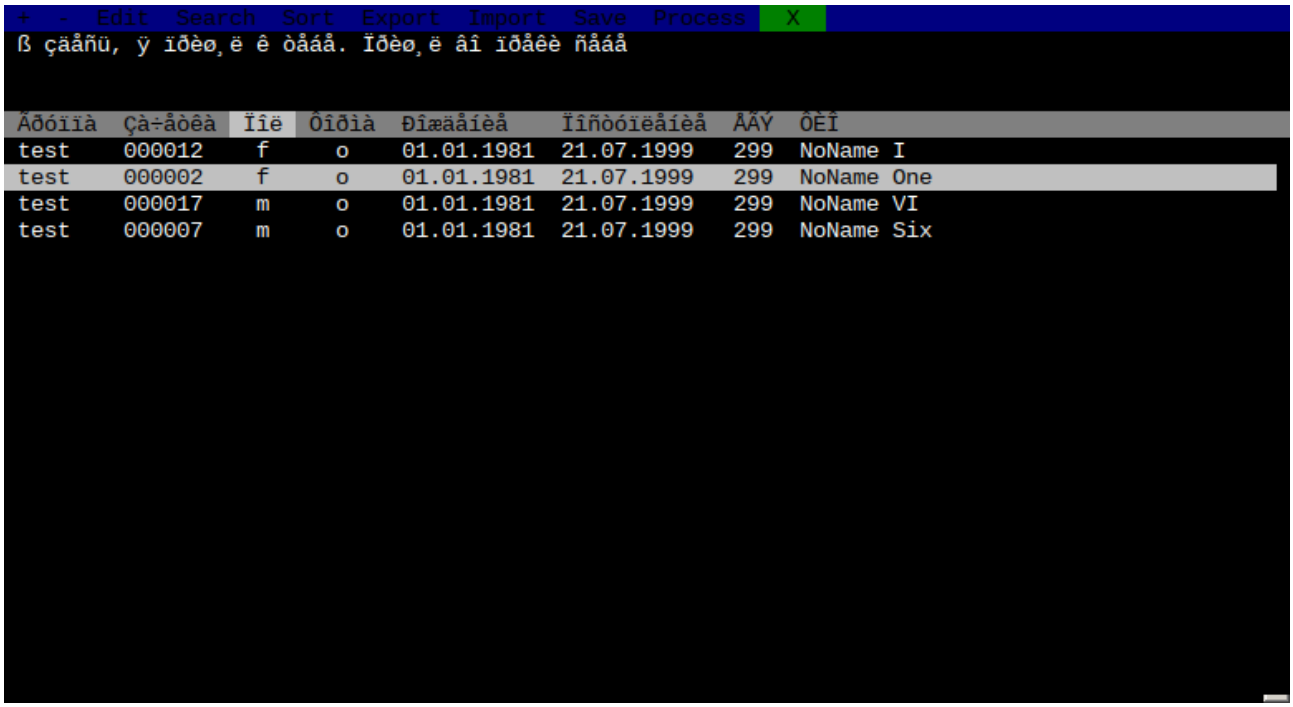


Рисунок 3.20 – Возвращение к основному списку

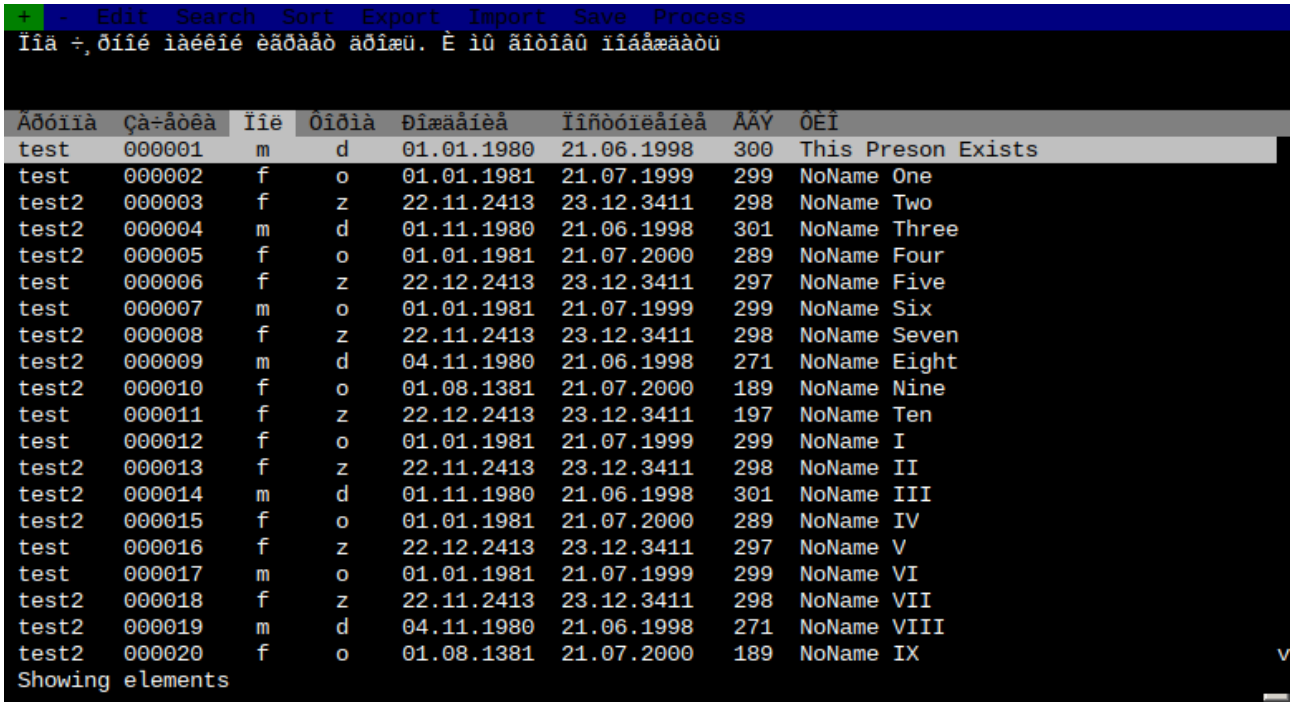


Рисунок 3.21 – Основной список остался прежним

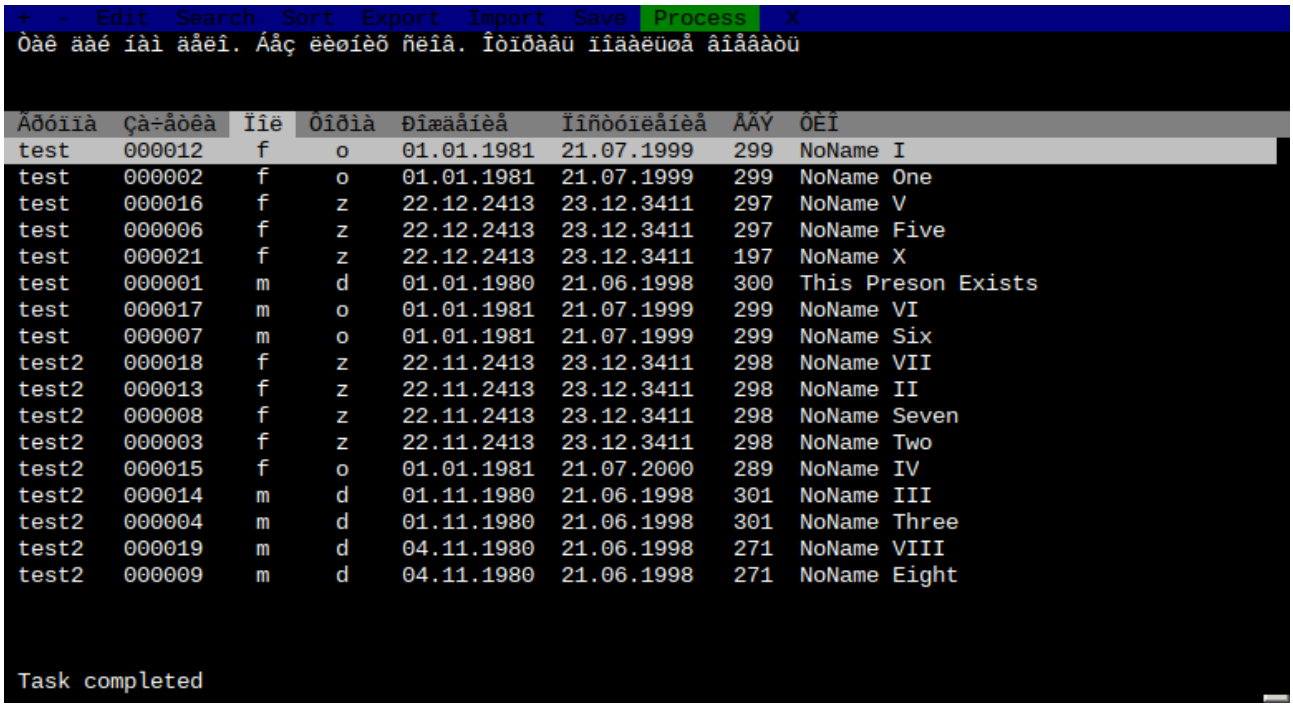
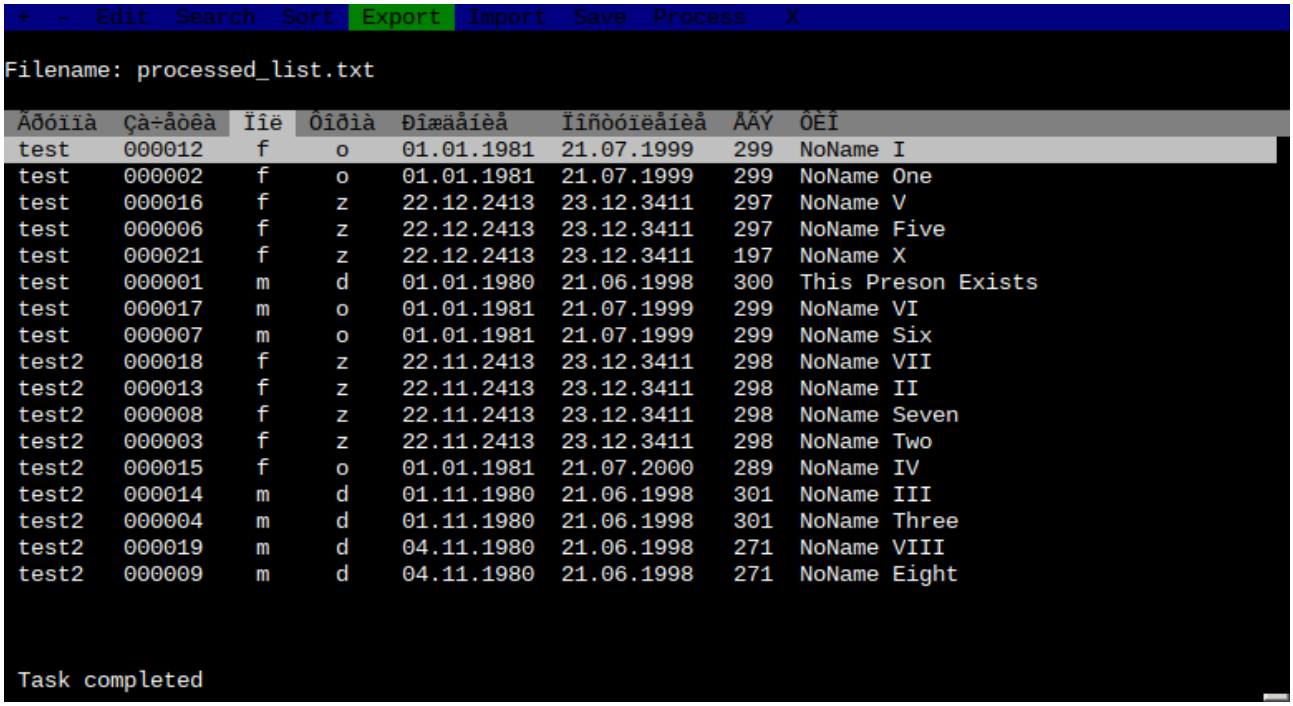


Рисунок 3.22 – Результат обработки списка



Ãðóïïà	Çà÷åòêà	Ïîë	Óíèòà	Ðîæäåíèå	Ïîñòóïëåíèå	ÂÀÝ	ÔÈÎ
test	000012	f	o	01.01.1981	21.07.1999	299	NoName I
test	000002	f	o	01.01.1981	21.07.1999	299	NoName One
test	000016	f	z	22.12.2413	23.12.3411	297	NoName V
test	000006	f	z	22.12.2413	23.12.3411	297	NoName Five
test	000021	f	z	22.12.2413	23.12.3411	197	NoName X
test	000001	m	d	01.01.1980	21.06.1998	300	This Preson Exists
test	000017	m	o	01.01.1981	21.07.1999	299	NoName VI
test	000007	m	o	01.01.1981	21.07.1999	299	NoName Six
test2	000018	f	z	22.11.2413	23.12.3411	298	NoName VII
test2	000013	f	z	22.11.2413	23.12.3411	298	NoName II
test2	000008	f	z	22.11.2413	23.12.3411	298	NoName Seven
test2	000003	f	z	22.11.2413	23.12.3411	298	NoName Two
test2	000015	f	o	01.01.1981	21.07.2000	289	NoName IV
test2	000014	m	d	01.11.1980	21.06.1998	301	NoName III
test2	000004	m	d	01.11.1980	21.06.1998	301	NoName Three
test2	000019	m	d	04.11.1980	21.06.1998	271	NoName VIII
test2	000009	m	d	04.11.1980	21.06.1998	271	NoName Eight

Рисунок 3.23 – Экспортирование обработанного списка

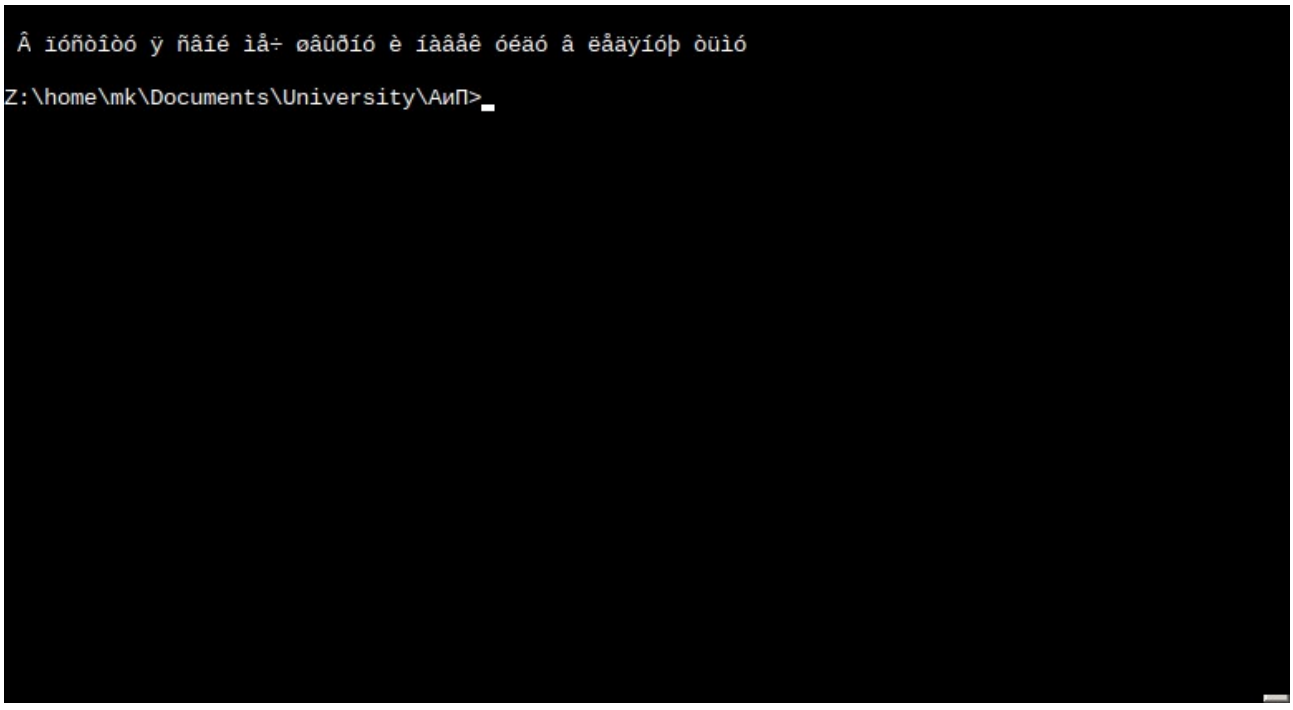


Рисунок 3.24 – Завершение программы

ВЫВОДЫ

В соответствии с вариантом задания разработана программа, в основу алгоритма которой положена структура данных в виде двунаправленного списка с двумя независимыми уровнями свзязывания, позволяющая выполнять просмотр данных в двух направлениях и формировать два списка, использующих одни и те же элементы, размещённые в памяти. Так же особенностью программы является возможность производить поиск и сортировку по всем полям записей, она достигается путём унификации функций сравнения и ввода, а так же введению массива с описаниями всех полей. Разработка преимущественно велась на базе дистрибутива Linux OpenSUSE в редакторе Visual Studio Code с испольованием MinGW для компиляции и Wine для запуска, отдельные компоненты прежде компилировались с помощью GCC и запускались напрямую для отладки. В процессе разработки возникла проблемма с кодировкой строк, которая была решена путём добавления опций компилятора при запуске оного.

Таким образом, была достигнута цель курсового проектирования – углублены, пусть и не значительно, знания по языку Си, получен навык разработки программ с использованием методологии структурного программирования. Результаты проектирования рекомендуется использовать в качестве топлива или вторсырья, или, в случае, если это не возможно, уничтожить любым другим способом перед прочтением.

Перечень ссылок

1. Белецкий Я. Энциклопедия языка Си / Я. Белецкий ; пер. с польск. — М. : Мир, 1992. — 687 с.
2. Вирт Н. Алгоритмы и структуры данных / Н. Вирт ; пер. с англ. — М. : Мир, 1989. — 360 с.
3. Керниган Б., Ритчи Д. Язык программирования СИ: пер. с англ./Под ред. и с предисл. В.С. Штаркмана. —2-е изд., перераб. и доп. — М. ; СПб. ; К. : Вильямс, 2006. —272с.
4. Павловская Т.А. С/ С++. Программирование на языке высокого уровня : учеб. для студ. вузов, обуч. по напр. «Информатика и вычислительная техника» / Т. А. Павловская. – СПб.: Питер, 2009. – 461 с.
5. Павловская Т.А. С/С++. Структурное программирование: практикум / Т.А. Павловская, Ю.А. Щупак. — СПб. : Питер, 2004.—239 с.
6. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : практикум / Т. А. Павловская. — СПб. : Питер, 2006. — 317 с.
7. Павловская Т. А. Паскаль. Программирование на языке высокого уровня : учеб. для вузов / Т. А. Павловская. — СПб. : Питер, 2008. — 393 с.
8. Методические указания к лабораторным работам по дисциплине «Информатика» для студентов дневной и заочной форм обучения направления 09.03.02 — «Информационные системы и технологии», часть 1 / Сост. В.Н. Бондарев, Т.И. Сметанина.— Севастополь: Изд-во СевГУ, 2014. — 44 с.
9. Методические указания к лабораторным работам по дисциплине «Информатика» для студентов дневной и заочной форм обучения направления 09.03.02 — «Информационные системы и технологии», часть 2 / Сост. В.Н. Бондарев, Т.И. Сметанина — Севастополь: Изд-во СевГУ, 2014. — 64с.
10. Структурное программирование на языке С/С++: методические указания к лабораторным работам по дисциплине «Основы программирования и алгоритмические языки» для студентов дневной и заочной форм обучения направления 09.03.02 — «Информационные системы и технологии», часть 1 / Сост. В.Н. Бондарев, Т.И. Сметанина.— Севастополь: Изд-во СевГУ, 2015. —60 с.
11. Структурное программирование на языке С/С++: методические указания к лабораторным работам по дисциплине «Основы программирования и алгоритмические языки» для студентов дневной и заочной форм обучения направления 09.03.02 — «Информационные системы и технологии», часть 2 / Сост. В.Н. Бондарев, Т.И. Сметанина.— Севастополь: Изд-во СевГУ, 2015. —60 с.
12. Разработка САПР : в 10 кн. Кн. 3. Проектирование программного обеспечения САПР : практ. пособие / Б. С. Федоров, Н. Б. Гуляев ; под ред. А.В. Петрова. — М. : Высш. шк., 1990. — 159с.

Приложение А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <conio.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#ifndef CUSTOM_DEFINES
#define CUSTOM_DEFINES


#define loop while(1)
#define new(T) malloc(sizeof(T))
#define len(x)  (sizeof(x) / sizeof((x)[0]))
#define TRUE 1
#define FALSE 0

typedef struct {
    char d, m;
    short y;
} Date;

void print_date(Date date){
    printf("%02hu.%02hu.%04hu", (short) date.d, (short) date.m, date.y);
}

#endif
#ifndef CONIO_EXTENSIONS
#define CONIO_EXTENSIONS


#define EDITOR_POSY    2

#define ARROW_UP 72
#define ARROW_DOWN 80
#define ARROW_LEFT 75
#define ARROW_RIGHT 77
#define KEY_ENTER 13
#define KEY_ESC 27
#define KEY_BACKSPACE 8
#define KEY_TAB 9

#define  READ_FUNC_SIGNATURE(name)  char  (*  name)(char  enter_dir,  short
posx, void * dest, struct field_struct field)
typedef int (* CompareFunc) (void * a, void * b);

/* Структура, применяемая для описания параметров полей элемента списка.
Используется редактором элементов и функциями ввода */
typedef struct field_struct {
    unsigned short posx;// Позиция на экране
    unsigned char len;    // Длинна на экране
    unsigned char size;    // Размер в памяти
    READ_FUNC_SIGNATURE(read_func); // Указатель на функцию чтения
    CompareFunc comp_func;    // Указатель на функцию для сравнения
    char * name;            // Название поля
    size_t offset;            // Смещение относительно начала элемента
    union {
        char values[8];    // Допустимые значения для char’ов
```

```
            char allow;        // Допустимые значения для строк
            #define ALLOW_NOTHING   0   // Ничего не разрешать кроме букв
            #define ALLOW_DIGITS    1   // Разрешить цифры
            #define ALLOW_SPECIAL   2   // Разрешить специальные символы
          //#define ALLOW_SPACES    4
        } prop; // Дополнительные параметры для функций чтения
    } Field;

    #ifdef LINUX
        #define read_char NULL
        #define read_string NULL
        #define read_fixed_int NULL
        #define read_fixed_short NULL
        #define read_fixed_date NULL
    #else


    #define    BACKGROUND_WHITE    (BACKGROUND_BLUE    |    BACKGROUND_GREEN    |
BACKGROUND_RED)
    #define BACKGROUND_GRAY BACKGROUND_INTENSITY

    HANDLE stdout_handle;
    void setCursorPosition(SHORT x, SHORT y){
        COORD coord = { x, y };
        SetConsoleCursorPosition(stdout_handle, coord);
    }

    CONSOLE_CURSOR_INFO cursor_info;
    void setCursorVisibility(char state){
        cursor_info.bVisible = state;
        SetConsoleCursorInfo(stdout_handle, &cursor_info);
    }

    CONSOLE_SCREEN_BUFFER_INFO buffer_info;
    void clear_lines(short from, short to){
        COORD coord = { 0, from };
        DWORD written;
         FillConsoleOutputCharacter(stdout_handle, ' ', buffer_info.dwSize.X *
to, coord, &written);
           FillConsoleOutputAttribute(stdout_handle,  buffer_info.wAttributes,
buffer_info.dwSize.X * to, coord, &written);
        //setCursorPosition(0, from);
    }

    CONSOLE_SCREEN_BUFFER_INFO orig_buffer_info;
    void adjust_buffer(){
        //SMALL_RECT rect = { 32, 32, 128, 32 };
        //SetConsoleWindowInfo(stdout_handle, TRUE, &rect);
        GetConsoleScreenBufferInfo(stdout_handle, &orig_buffer_info);
        buffer_info = orig_buffer_info;
                  buffer_info.dwSize.X    =    buffer_info.srWindow.Right    -
buffer_info.srWindow.Left + 1;
                 buffer_info.dwSize.Y    =    buffer_info.srWindow.Bottom    -
buffer_info.srWindow.Top + 1;
        SetConsoleScreenBufferSize(stdout_handle, buffer_info.dwSize);
    }

    void restore_buffer(){
        SetConsoleScreenBufferSize(stdout_handle, orig_buffer_info.dwSize);
    }

    void setColor(short from_x, short from_y, DWORD _len, WORD attr){
```

```
        DWORD written;
        COORD coord = { from_x, from_y };
              FillConsoleOutputAttribute(stdout_handle,  attr,  _len,  coord,
&written);
    }

    void repeat(short from_x, short from_y, DWORD _len, char c){
        DWORD written;
        COORD coord = { from_x, from_y };
        FillConsoleOutputCharacter(stdout_handle, c, _len, coord, &written);
    }

    char read_string(char enter_dir, short posx, void * dest, Field field){

        #define dest ((char *) dest)

        char buffer_size = field.len;
        char allow_digits = field.prop.allow & ALLOW_DIGITS;
        char allow_special = field.prop.allow & ALLOW_SPECIAL;
        //char allow_spaces = field.prop.allow & ALLOW_SPACES;

        char * buffer = &dest[1];
        unsigned char last_char = dest[0];
        unsigned char cursor_pos = 0;
        if(enter_dir == ARROW_LEFT)
            cursor_pos = last_char;

        //setCursorPosition(posx, EDITOR_POSY);
        //printf("%s", buffer);
        setCursorPosition(posx + cursor_pos, EDITOR_POSY);

        int ch;
        loop {

            ch = _getch();
            //printf("DEBUG: %d\n", ch);
            if(ch == 224){
                ch = _getch();
                //printf("DEBUG: %d\n", ch);
                if(ch == ARROW_LEFT){
                    if(cursor_pos != 0){
                        setCursorPosition(posx + cursor_pos - 1, EDITOR_POSY);
                        cursor_pos--;
                    } else
                        goto exit;
                } else if(ch == ARROW_RIGHT){
                    if(cursor_pos != last_char){
                        setCursorPosition(posx + cursor_pos + 1, EDITOR_POSY);
                        cursor_pos++;
                    } else
                        goto exit;
                } else if(ch == ARROW_UP){
                    cursor_pos = 0;
                    setCursorPosition(posx, EDITOR_POSY);
                } else if(ch == ARROW_DOWN){
                    cursor_pos = last_char;
                    setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                }
            } else {
                //printf("DEBUG: %c not 244\n", ch);
                if(ch == KEY_ENTER || ch == KEY_ESC || ch == KEY_TAB)
                    goto exit;
```

```
                else if(ch == KEY_BACKSPACE){
                    if(cursor_pos != 0){
                        setCursorPosition(posx + cursor_pos - 1, EDITOR_POSY);
                        if(cursor_pos == last_char)
                            putchar(' ');
                        else {
                             for(int i = cursor_pos - 1; i < last_char - 1; i+
+)
                                 buffer[i] = buffer[i + 1];
                            buffer[last_char - 1] = '\0';
                            printf("%s ", &buffer[cursor_pos - 1]);
                        }
                        cursor_pos--;
                        last_char--;
                        setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                    }
                } else if
                (
                    (allow_digits && ch >= '0' && ch <= '9') ||
                    (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') ||
                    (ch >= 'а' && ch <= 'я') || (ch >= 'А' && ch <= 'Я') ||
                     (allow_special && (ch == '.' || ch == ':' || ch == '/' ||
ch == '\\' || ch == '_')) ||
                       // ќќќќќќ ќќ ќќќќќ ќќќќ ќќќќќќ, ќќ ќќќќќ ќќќќ ќќќ ќќќќ
ќќќќќќ
                        (/*allow_spaces &&*/ ch == ' ' && (cursor_pos != 0 &&
buffer[cursor_pos - 1] != ' '))
                ){

                    if(ch == ' ' && buffer[cursor_pos] == ' '){
                        setCursorPosition(posx + (++cursor_pos), EDITOR_POSY);
                        continue;
                    }

                    if(last_char == buffer_size){
                        setCursorPosition(posx, EDITOR_POSY + 1);
                        puts("Too many letters!");
                        setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                        continue;
                    }

                    if(cursor_pos == last_char){
                        buffer[cursor_pos] = ch;
                        putchar(ch);
                    } else {
                        for(int i = last_char; i > cursor_pos; i--)
                            buffer[i] = buffer[i - 1];
                        buffer[cursor_pos] = ch;
                        printf("%s", &buffer[cursor_pos]);
                    }
                    cursor_pos++;
                    last_char++;

                    setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                }
            }
        }
    exit:
        if(buffer[last_char - 1] == ' ')
            last_char--;
        buffer[last_char] = '\0';
        dest[0] = last_char;
```

```
    return ch;

    #undef dest
}

char read_fixed_int(char enter_dir, short posx, void * dest, Field field){

    #define dest ((unsigned int *) dest)
    unsigned char n_digits = field.len;

    char buffer[11];
    unsigned char cursor_pos = 0;
    if(enter_dir == ARROW_LEFT)
        cursor_pos = n_digits - 1;

    int temp = *dest;
    for(int i = n_digits - 1; i >= 0; i--){
        buffer[i] = '0' + (temp % 10);
        temp /= 10;
    }
    buffer[n_digits] = '\0';

    //setCursorPosition(posx, EDITOR_POSY);
    //printf("%s", buffer);
    setCursorPosition(posx + cursor_pos, EDITOR_POSY);

    int ch;
    loop {
        ch = _getch();
        if(ch == 224){
            ch = _getch();
            if(ch == ARROW_LEFT){
                if(cursor_pos != 0){
                    cursor_pos--;
                    setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                } else
                    goto exit;
            } else if(ch == ARROW_RIGHT){
                if(cursor_pos != n_digits - 1){
                    cursor_pos++;
                    setCursorPosition(posx + cursor_pos, EDITOR_POSY);
                } else
                    goto exit;
            }
        } else {
            if(ch == KEY_ENTER || ch == KEY_ESC || ch == KEY_TAB)
                goto exit;
            else if(ch >= '0' && ch <= '9'){
                buffer[cursor_pos] = ch;
                putchar(ch);
                cursor_pos++;

                if(cursor_pos == n_digits)
                    goto exit;

                setCursorPosition(posx + cursor_pos, EDITOR_POSY);
            }
        }
    }

exit:
    *dest = atoi(buffer);
```

```
        return ch;

        #undef dest
    }

    char read_fixed_short(char enter_dir, short posx, void * dest, Field
field){
        #define dest ((unsigned short *) dest)
        unsigned int temp = *dest;
        char ret = read_fixed_int(enter_dir, posx, &temp, field);
        *dest = temp & 0xFFFF;
        return ret;
        #undef dest
    }

    char read_fixed_date(char enter_dir, short posx, void * dest, Field field)
{

        #define dest ((Date *) dest)

        char buffer[11] = {
            '0' + (dest->d / 10) % 10,
            '0' + (dest->d % 10),
            '.',
            '0' + (dest->m / 10) % 10,
            '0' + (dest->m) % 10,
            '.',
            '0' + (dest->y / 1000) % 10,
            '0' + (dest->y / 100) % 10,
            '0' + (dest->y / 10) % 10,
            '0' + (dest->y % 10),
            '\0'
        };

        unsigned char cursor_pos = 0;
        if(enter_dir == ARROW_LEFT)
            cursor_pos = 9;

        //setCursorPosition(posx, EDITOR_POSY);
        //puts(buffer);
        setCursorPosition(posx + cursor_pos, EDITOR_POSY);

        #define inc_cursor_pos() { \
            cursor_pos += 1 + (cursor_pos == 1 || cursor_pos == 4); \
            setCursorPosition(posx + cursor_pos, EDITOR_POSY); \
        }
        #define dec_cursor_pos() { \
            cursor_pos -= 1 + (cursor_pos == 3 || cursor_pos == 6); \
            setCursorPosition(posx + cursor_pos, EDITOR_POSY); \
        }

        int ch;
        loop {
            ch = _getch();
            if(ch == 224){
                ch = _getch();
                if(ch == ARROW_LEFT){
                    if(cursor_pos != 0){
                        dec_cursor_pos();
                    } else
                        goto exit;
                } else if(ch == ARROW_RIGHT){
```

```
                    if(cursor_pos != 9){
                        inc_cursor_pos();
                    } else
                        goto exit;
                }
            } else {
                if(ch == KEY_ENTER || ch == KEY_ESC || ch == KEY_TAB)
                    goto exit;
                else if(
                     (cursor_pos == 0 && ch >= '0' && (ch <= '2' || (ch == '3'
&& buffer[1] <= '1'))) ||
                     (cursor_pos == 1 && ch >= '0' && (ch == '1' || (ch <= '9'
&& buffer[0] != '3'))) ||
                     (cursor_pos == 3 && (ch == '0' || (ch == '1' && buffer[4]
<= '2'))) ||
                     (cursor_pos == 4 && ch >= '0' && (ch <= '2' || (ch <= '9'
&& buffer[3] != '1'))) ||
                    (cursor_pos >= 6 && (ch >= '0' && ch <= '9'))
                ){
                    buffer[cursor_pos] = ch;
                    putchar(ch);

                    if(cursor_pos == 9)
                        goto exit;

                    inc_cursor_pos();
                }
            }
        }

        #undef inc_cursor_pos
        #undef dec_cursor_pos

    exit:
        dest->d = (buffer[0] - '0') * 10 + (buffer[1] - '0');
        dest->m = (buffer[3] - '0') * 10 + (buffer[4] - '0');
        dest->y = (buffer[6] - '0') * 1000 + (buffer[7] - '0') * 100 +
                  (buffer[8] - '0') * 10   + (buffer[9] - '0');
        return ch;

        #undef dest
    }

    char read_char(char enter_dir, short posx, void * dest, Field field){

        #define dest ((char *) dest)

        char * values = field.prop.values;

        char n_values = (values++)[0];
        char buffer = *dest;

        //setCursorPosition(posx, EDITOR_POSY);
        //putchar(buffer);
        setCursorPosition(posx, EDITOR_POSY);

        int ch;
        loop {
            ch = _getch();
            if(ch == 224){
                ch = _getch();
                if(ch == ARROW_LEFT || ch == ARROW_RIGHT)
```

```
                    goto exit;
            } else {
                if(ch == KEY_ENTER || ch == KEY_ESC || ch == KEY_TAB)
                    goto exit;
                else {
                    for(unsigned char i = 0; i < n_values; i++)
                        if(ch == values[i]){
                            buffer = ch;
                            putchar(ch);
                            goto exit;
                        }
                }
            }
        }

    exit:
        *dest = buffer;
        return ch;

        #undef dest
    }

    #endif // ifndef LINUX

    #endif

    #ifndef CUSTOM_LIST
    #define CUSTOM_LIST


    #define FILEDATA_NAME_LEN 32
    #define AUTOSAVE_FILE_NAME "test.bin"

    typedef struct {
        char group_name[6 + 2]; // 8
        unsigned int gradebook_number; // 4
        char full_name[FILEDATA_NAME_LEN]; // 32
        char gender, education_form; // 1 + 1
        Date birth_date, admission_date; // 4 + 4
        unsigned short USE_score; // 2
    } FileData;

    typedef struct list_element {
        FileData data;
        struct list_element * link[4]; // 4 * 4
    } ListElement;

    typedef enum { SHOW = 0, SEARCH = 1 } ListLinkLayer;
    ListLinkLayer link_layer = SHOW;

    #define DIR(d) link[d * 2 + link_layer]
    #define PREV   link[0 + link_layer]
    #define NEXT   link[2 + link_layer]

    #define connect(a, b)  { a->NEXT = b; b->PREV = a; }
    #define connect3(a, b, c) { connect(a, b); connect(b, c); }

    /*
    char read_string(char enter_dir, short posx, char * dest, Field field);
    char read_fixed_int(char enter_dir, short posx, unsigned int * dest, Field
field);
    char read_char(char enter_dir, short posx, char * dest, Field field);
```

```
      char read_fixed_date(char enter_dir, short posx, Date * dest, Field
field);
      char read_fixed_short(char enter_dir, short posx, unsigned short * dest,
Field field);
      */

      #define compare_func(name, type) int name(void * a, void * b){ \
          return *(type *) a - *(type *) b; \
      }

      compare_func(intcmp, int)
      compare_func(chrcmp, char)
      compare_func(shrcmp, short)

      int cmpdte(Date * a, Date * b){
          if(a->y == b->y){
              if(a->m == b->m)
                  return a->d - b->d;
              return a->m - b->m;
          }
          return a->y - b->y;
      }

      // Группа   Зачетка   Пол   Форма   Рождение      Поступление   ЕГЭ   ФИО
//
      //   ......      ......            .               .           00.00.0000
00.00.0000   ...  ............................. //

      #define field(x, l, s, r, c, n, o, p) { x, l, s, r, c, n,
offsetof(FileData, o), p }
      Field list_element_fields[] = {
          field( 1,  6,            6, read_string     ,  strcmp, "Группа"     ,
group_name      , { allow: ALLOW_DIGITS }),
          field( 9,  6,   sizeof(int), read_fixed_int  ,  intcmp, "Зачетка"    ,
gradebook_number, {0}),
          field(19,  1,            1, read_char       ,  chrcmp, "Пол"        ,
gender          , { values: "\2mf"  }),
          field(25,  1,            1, read_char       ,  chrcmp, "Форма"      ,
education_form  , { values: "\3ozd" }),
          field(30, 10,  sizeof(Date), read_fixed_date ,  cmpdte, "Рождение  " ,
birth_date      , {0}),
          field(42, 10,  sizeof(Date), read_fixed_date ,  cmpdte, "Поступление",
admission_date  , {0}),
          field(55,  3, sizeof(short), read_fixed_short,  shrcmp, "ЕГЭ"        ,
USE_score       , {0}),
          field(60, 32,           32, read_string     ,  strcmp, "ФИО"        ,
full_name       , { allow: ALLOW_NOTHING })
      };
      #undef field

      ListElement * first_alloc_begin = NULL;
      ListElement * first_alloc_end = NULL;
      ListElement * freeded_elements = NULL;
      ListElement * last_readed;

      ListElement * list_element_new(){
          if(freeded_elements != NULL){
              ListElement * popped = freeded_elements;
              freeded_elements = popped->link[2]; // 2 = SHOW NEXT
              return popped;
          }
          return new(ListElement);
```

```
    }

    void list_element_free(ListElement * el){
        ListElement * head = freeded_elements;
        freeded_elements = el;
        el->link[2] = head; // 2 = SHOW NEXT
    }

    void element_print(ListElement * cur){
        FileData * _ = &cur->data;
          printf(" %-6s   %06u      %c       %c      ", &_->group_name[1], _->gradebook_number, _->gender, _->education_form);
        print_date(_->birth_date);
        printf("  ");
        print_date(_->admission_date);
        printf("   %03hu  %-79s\n", _->USE_score, &_->full_name[1]);
    }

    // returns negative number on falture
    int element_print_to_txt(FILE * f, ListElement * cur){
        FileData * _ = &cur->data;
        return fprintf(
             f, "%-6s %06u %c %c %02hu.%02hu.%04hu %02hu.%02hu.%04hu %3hu %s\n",
                      &_->group_name[1],  _->gradebook_number,  _->gender,  _->education_form,
            (short) _->birth_date.d, (short) _->birth_date.m, _->birth_date.y,
                (short) _->admission_date.d, (short) _->admission_date.m, _->admission_date.y,
            _->USE_score, &_->full_name[1]
        );
    }

    int element_read_from_txt(FILE * f, ListElement * cur){
        FileData * _ = &cur->data;
        {
            int ret = fscanf(
                f, "%6c %u %c %c %hhu.%hhu.%hu %hhu.%hhu.%hu %hu ",
                     &_->group_name[1], &_->gradebook_number, &_->gender, &_->education_form,
                &_->birth_date.d, &_->birth_date.m, &_->birth_date.y,
                             &_->admission_date.d,  &_->admission_date.m,  &_->admission_date.y,
                &_->USE_score
            );

            if(ret <= 0)
                return -1;

            unsigned char i = 6;
            for(; i > 0 && _->group_name[i] == ' '; i--);
            _->group_name[0] = i;
        }{
            char * ret = fgets(&_->full_name[1], FILEDATA_NAME_LEN, f);
            if(ret == NULL)
                return -1;

            unsigned char i = 1;
            for(; i < FILEDATA_NAME_LEN - 1 && !(_->full_name[i] == '\r' || _->full_name[i] == '\n' || _->full_name[i] == '\0'); i++);
            _->full_name[i] = '\0';
            _->full_name[0] = i - 1;
```

```
    }
    return 1;
}

void element_zerofill(ListElement * elem){
    memset(elem, 0, sizeof(ListElement));
    elem->data.gender = 'm';
    elem->data.education_form = 'd';
}

ListElement * heads[2] = { 0 };
ListElement * tails[2] = { 0 };
#define HEAD heads[link_layer]
#define TAIL tails[link_layer]

unsigned int list_lengths[2] = { 0 };
#define list_len list_lengths[link_layer]

void list_add(ListElement * el){

    if(!HEAD){
        HEAD = el;
        HEAD->PREV = NULL;
    } else {
        TAIL->NEXT = el;
        el->PREV = TAIL;
    }
    TAIL = el;
    TAIL->NEXT = NULL;

    list_len++;
}

size_t get_file_size(FILE * f){
    fseek(f, 0, SEEK_END);
    size_t file_size = ftell(f);
    rewind(f);
    return file_size;
}

void list_autoload(){
    FILE * file = fopen(AUTOSAVE_FILE_NAME, "rb");

    if(file){

        size_t file_size = get_file_size(file);
        if(file_size % sizeof(FileData) != 0)
            puts(AUTOSAVE_FILE_NAME " seems to be corrupted");
        file_size /= sizeof(FileData);

        first_alloc_begin = malloc((file_size + 1) * sizeof(ListElement));
        first_alloc_end = first_alloc_begin + file_size;
        last_readed = first_alloc_begin;
        while(fread(last_readed, sizeof(FileData), 1, file)){
            list_add(last_readed);
            last_readed++;
        }

        fclose(file);

    } else {
        first_alloc_begin = last_readed = new(ListElement);
```

```
            first_alloc_end = first_alloc_begin + 1;
        }

    }

    void list_autosave(){
        FILE * file = fopen(AUTOSAVE_FILE_NAME, "wb");
        if(!file){
            puts("can't open " AUTOSAVE_FILE_NAME " for writing");
            return;
        }

        for(ListElement * cur = HEAD; cur; cur = cur->NEXT)
            if(!fwrite(&cur->data, sizeof(FileData), 1, file)){
                puts("can't write to file");
                goto exit;
            }

    exit:
        fflush(file);
        fclose(file);
    }

    void list_remove(ListElement * el){

        if(el == HEAD){
            HEAD = el->NEXT;
            if(HEAD)
                HEAD->PREV = NULL;
        } else
            el->PREV->NEXT = el->NEXT;

        if(el == TAIL){
            TAIL = el->PREV;
            if(TAIL)
                TAIL->NEXT = NULL;
        } else
            el->NEXT->PREV = el->PREV;

        list_len--;

        if(link_layer == SHOW)
            list_element_free(el);
    }

    CompareFunc list_element_compare_func;
    size_t field_to_compare_by_offset;
    int list_element_compare(ListElement * a, ListElement * b){
                    return    list_element_compare_func((char    *)    a    +
field_to_compare_by_offset, (char *) b + field_to_compare_by_offset);
    }

    typedef struct {
        ListElement * first, * last;
        ListElement * next; // указатель на начало следующего отрезка
    } Cut;

    Cut merge(Cut l1, Cut l2){
        Cut l[] = { l1, l2 };
        Cut cut = { .next = l2.next };
        ListElement * cur, * cur_[2];
```

```
        unsigned char min = list_element_compare(l1.first, l2.first) >= 0;
        cut.first = cur = l[min].first;
        cur_[min] = cur->NEXT;
        cur_[!min] = l[!min].first;

        while(1){
            min = list_element_compare(cur_[0], cur_[1]) >= 0;
            connect(cur, cur_[min]);
            cur = cur_[min];
            if(cur == l[min].last){
                /*
                Q: why doesn't it fails when cur_[!min] == NULL
                A: cur_[!min] is never null
                    A: Потому что когда хоть в одном из списков встречается
последний элемент...
                В эту функцию всегда передаются два отрезка, каждый из которых
в длинну > 3
                 При этом первый может быть короче второго на 1 элемент, но не
наоборот
                И второй отрезок идёт после первого в исходном списке
                    A: Но, да если вызвать функцию отдельно с некорректными
данными, она может упасть
                */
                connect(cur, cur_[!min]);
                cut.last = l[!min].last;
                return cut;
            } else
                cur_[min] = cur->NEXT;
        }
    }

    Cut recursion(ListElement * first, int len){
        //printf("r %i %i\n", from, to);
        if(len > 3){
            int half_len = len / 2;
            Cut cut1 = recursion(first, half_len);
            Cut cut2 = recursion(cut1.next, len - half_len); // uses next
            return merge(cut1, cut2);
        } else if(len == 3){
            Cut cut = recursion(first->NEXT, 2);
            if(list_element_compare(first, cut.first) <= 0){
                connect(first, cut.first);
                cut.first = first;
            } else if(list_element_compare(first, cut.last) < 0){
                connect3(cut.first, first, cut.last);
            } else {
                connect(cut.last, first);
                cut.last = first;
            }
            return cut;
        } else /* if(len == 2) */ {
            Cut cut = { first, first->NEXT };
            cut.next = cut.last->NEXT;
            if(list_element_compare(cut.first, cut.last) > 0){
                ListElement * tmp = cut.first;
                cut.first = cut.last;
                cut.last = tmp;

                connect(cut.first, cut.last);
            }
            return cut;
        }
```

```
    }

    //WARN: list_len should be >= 2
    void merge_sort(unsigned char field_id){
        Field field = list_element_fields[field_id];
        field_to_compare_by_offset = field.offset;
        list_element_compare_func = field.comp_func;

        if((char *) field.read_func == (char *) read_string)
            field_to_compare_by_offset++;

        Cut cut = recursion(HEAD, list_len);
        HEAD = cut.first;
        TAIL = cut.last;
        HEAD->PREV = NULL;
        TAIL->NEXT = NULL;
    }

    void list_free(){
        link_layer = SHOW;
        freeded_elements = heads[SHOW];
        heads[SHOW] = heads[SEARCH] = NULL;
        tails[SHOW] = tails[SEARCH] = NULL;
        list_len = 0;
    }

    void list_release_memory(){
        #define link_layer SHOW
        if(TAIL){
            // freeded_elements = HEAD -> freeded_elements
              // becouse we cannot connect anything to the untracked end of
freeded_elements
            TAIL->NEXT = freeded_elements;
            freeded_elements = HEAD;
        }
        // last_readed -> freeded_elements
        last_readed->NEXT = freeded_elements;
        for(ListElement * cur = last_readed; cur;){
            ListElement * next = cur->NEXT;
            if(cur <= first_alloc_begin && cur > first_alloc_end)
                free(cur);
            cur = next;
        }
        /*
        first_alloc_begin = first_alloc_end = 0;
        freeded_elements = last_readed = NULL;
        heads[SHOW] = heads[SEARCH] = NULL;
        tails[SHOW] = tails[SEARCH] = NULL;
        list_len = 0;
        */
        #undef link_layer
    }

    void list_copy_to_search_layer(){
        heads[SEARCH] = heads[SHOW];
        tails[SEARCH] = tails[SHOW];
        for(ListElement * cur = heads[SHOW]; cur; cur = cur->NEXT){
            cur->link[1] = cur->link[0];
            cur->link[3] = cur->link[2];
        }
        list_lengths[SEARCH] = list_lengths[SHOW];
    }
```

```
      int list_process_cmp(FileData * a, FileData * b){
          int ret = strcmp(a->group_name, b->group_name);
          if(!ret){
              ret = a->gender - b->gender;
              if(!ret){
                  ret = b->USE_score - a->USE_score;
                  if(!ret){
                      ret = (
                          a->admission_date.y * 365 + a->admission_date.m * 31 +
a->admission_date.m
                      ) - (
                          b->admission_date.y * 365 + b->admission_date.m * 31 +
b->admission_date.m
                      );
                  }
              }
          }
          return ret;
      }

      void list_process(){
          //assert(list_len > 2)

          list_copy_to_search_layer();
          link_layer = SEARCH;

          field_to_compare_by_offset = offsetof(ListElement, data);
          list_element_compare_func = list_process_cmp;
          Cut cut = recursion(HEAD, list_len);
          HEAD = TAIL = cut.first;
          cut.last->NEXT = NULL;
          HEAD->PREV = NULL;
          list_len = 1;

          char i = 1;
          ListElement * cur = HEAD->NEXT;
          while(cur){
              if(i < 5){
                  connect(TAIL, cur);
                  TAIL = cur;
                  list_len++;
                  i++;
              }
              ListElement * next = cur->NEXT;
              if(next && (strcmp(next->data.group_name, cur->data.group_name) ||
next->data.gender != cur->data.gender))
                  i = 0;
              cur = next;
          }

          TAIL->NEXT = NULL;
      }
      /*
      int main(){
          {
          ListElement list[] = {
              { { .USE_score = 200 }, {     NULL, &list[1] } },
              { { .USE_score = 300 }, { &list[0], &list[2] } },
              { { .USE_score = 500 }, { &list[1], &list[3] } },
              { { .USE_score = 350 }, { &list[2], &list[4] } },
              { { .USE_score = 349 }, { &list[3], &list[5] } },
```

```
            { { .USE_score = 250 }, { &list[4], NULL     } }
        };
        HEAD = list;
        TAIL = &list[len(list) - 1];
        list_len = len(list);
        }

        int scores[] = { 200, 300, 500, 350, 349, 250 };
        for(unsigned char i = 0; i < len(scores); i++){
            ListElement * last_readed = new(ListElement);
            last_readed->data.USE_score = scores[i];
            list_add(last_readed);
        }

        for(ListElement * cur = HEAD; cur; cur = cur->NEXT)
            element_print(cur);
        merge_sort();
        for(ListElement * cur = HEAD; cur; cur = cur->NEXT)
            element_print(cur);
        return 0;
    }
    */
    #endif

    #ifndef QUOTES_ENGINE
    #define QUOTES_ENGINE


    #define roll(i, len) (i = (i + 1) % len)
    #define len(x)  (sizeof(x) / sizeof((x)[0]))

    typedef struct {
        int exit, remove, mistake, start, edit;
    } QuotesState;

    QuotesState quotes_state = {};

    const char * const start_quotes[] = {
        "Я здесь, я пришёл к тебе. Пришёл во преки себе",
        "Под чёрной майкой играет дрожь. И мы готовы побеждать",
        "Так дай нам дело. Без лишних слов. Отправь подальше воевать",
        "Ооо. Хозяин тот идёт. Ооо. Душа как ночь черна",
        "Пора. Готовься сделать шаг. Тебе дадут знак!",
        "Заготовки молот бьёт. Кровь ликует в теле. Льёт горячий, жгучий пот. Здесь куют метал!",
        "Раздуваются меха. Сложен ход процесса. Это вам не чепуха, здесь куют метал!",
        "Несколько звонких монет, приобретайте билет. После сеанса вас ждёт ресторан с баром",
        "..."
    };

    const char * const mistake_quotes[] = {
        "Дай мне сойти с ума, ведь с безумца и спроса нет!",
        "Дай мне хоть раз сломать этот слишком нормальный свет!",
        "Бесы! Бесы всё злей и злей",
        "Бесы! Бесы в душе моей",
        "Господи, я не твой - ближних я не могу любить",
        "Безумец! Беглец. Дороги нет. Ты видишь неверный свет",
        "Он копит силы, лимита ждёт, направив в небо радар. Одна ошибка, случайный взлёт и неизбежен удар!",
        "Ты не предатель, ты просто глупец и таких миллионы кругом",
```

```
        "Небо, слабых не милуй, всем не под силу бремя свобод",
        "Но, вдруг, резкий сбой и рвет ночь громкий вой сирен",
        "..."
    };

    const char * const edit_quotes[] = {
          "Всё меняет время в мировой сети. Ты бессмертный воин, лучше не
найти",
        "Всё меняет время. Залпы батарей. Под огнём идёшь ты по чужой земле",
         "Всё изменилось в прозривших глазах: кто был крысой, стал тигром и
львом",
        "Отредактировано"
    };

    const char * const remove_quotes[] = {
        "Путь в никуда. Я искал тебе пути иного",
        "Путь в никуда. Ничего уже исправить нельзя",
        "Ты мстил за грусть нелюбви прошлых лет",
         "Когда мы солнце отдадим на расправу тёмным силам, три цифры будут
править бал",
        "Пускай свершится ритуал и оставит смертельный шквал пепел веков",
         "В последний огненый полёт улетают книги птицы. Их не вернуть уже
назад",
        "Шипит змеёй мой огнемёт, бьются крыльями страницы",
        "Они навеки замолчат и отправятся прямо в ад, в пепел веков",
        "Снится. Мне снова снится, как вырываются крики из дыма",
         "Найти и уничтожить - твой дивиз такой. Трупы и руины, кровь течёт
рекой",
        "Он не просит - жжёт и рушит. В ночь уносит наши души",
        "Всадник отслужит по тебе чёрную мессу по тебе",
        "Машина смерти сошла с ума. Она летит сметая всех",
        "Ооо. И дьявол не берёт. Кому она нужна? Ах-ха-ха-ха",
        "Я был скрипачём. Мой талант в игре жизнью и смычком",
        "Всё превращая в быль крутят адское колесо",
         "Всё, во что ты навеки влюблён, уничтожит разом тыщеглавый убийца
дракон. Должен быть повержен он!",
        "Этот урок слишком жесток. И никто не ответит теперь, почему и за что"
    };

    const char * const exit_quotes[] = {
        "В пустоту я свой меч швырну и навек уйду в ледяную тьму",
        "Уходи и не возвращайся. И не возвращайся ко мне",
        "До скорой встречи на костре",
        "Путь в никуда. Я зову, но мне в ответ ни слова",
        "Путь в никуда. Из под ног моих уходит земля",
        "За дверь я выгнан в ночь. Но выйти вон и сам не прочь",
        "Да я уйду. И мне плевать. Ты знаешь, где меня искать",
        "Я смогу тебя забыть быстрей, это дело двух ночей",
        "Закат багровой краской твой бы красил щит и враги бы знали, ты лишён
души",
         "Смотри же в мои глаза. Твой взгляд не понять нельзя. Ты хочешь меня
убить. Убить и про всё забыть",
        "Грянул хор и качнулся храм. Смерть пришла за мной в пять часов утра",
        "Сила приносит свободу. Побеждай и станешь звездой. А может, обретёшь
покой",
        "Блаженство рая я оставлю для нищих. У нищих духом должен быть царь и
бог",
         "Я тварь земная и на небе я лишний. И к чёрту вечность. Какой в ней
прок?",
        "Беги-беги за солнцем, стирая ноги в кровь",
        "Прощай Норфолк! За честь короны мы умёрм",
        "Прощай Норфолк! Мы в рай едва ли попадём",
```

```
          "Если вдруг тебе не вмочь, мы не просим дважды - уходи скорее прочь,
здесь куют метал!",
      };

      #define event_quote(event) \
      void event##_quote(){ \
          setCursorPosition(1, 1); \
          puts(event##_quotes[quotes_state.event]); \
          roll(quotes_state.event, len(event##_quotes)); \
      }

      event_quote(start)
      event_quote(exit)
      event_quote(edit)
      event_quote(mistake)
      event_quote(remove)

      #endif
      #ifndef CUSTOM_UI
      #define CUSTOM_UI

      #define MENU_POSY    0
      #define EDITOR_POSY    2 // +-1
      #define HEADER_POSY 4
      #define SCROLL_POSY 5
      #define E404_POSY    6


      ListElement * scroll_first_element_on_screen = NULL;
      ListElement * scroll_selected_element = NULL;
      unsigned int  scroll_selected_element_pos = 0;

      unsigned short scroll_first_line = SCROLL_POSY;
      unsigned short scroll_last_line = 5;

      void scroll_set_head(ListElement * el){
          scroll_first_element_on_screen = el;
          scroll_selected_element = el;
          scroll_selected_element_pos = 0;
      }

      #define draw_scroll redraw_scroll
      void redraw_scroll(){

          setCursorPosition(0, scroll_first_line); //clear_lines(0, BUFFER_Y);

          if(HEAD){
              ListElement * cur = scroll_first_element_on_screen;
              unsigned int drawn = 0;
               for(; cur && drawn < (scroll_last_line - scroll_first_line + 1);
drawn++){
                  if(drawn == scroll_selected_element_pos){
                      SetConsoleTextAttribute(stdout_handle, BACKGROUND_WHITE);
                      element_print(cur);
                                          SetConsoleTextAttribute(stdout_handle,
buffer_info.wAttributes);
                  } else
                      element_print(cur);
                  cur = cur->NEXT;
              }
                 //setColor(0, scroll_selected_element_pos + scroll_first_line,
buffer_info.dwSize.X - 1, BACKGROUND_WHITE);
```

```
            setCursorPosition(buffer_info.dwSize.X - 1, scroll_first_line);
            if(scroll_first_element_on_screen != HEAD)
                putchar('^');
            else
                putchar(' ');

            setCursorPosition(buffer_info.dwSize.X - 1, scroll_last_line);
            if(cur)
                putchar('v');
            else
                putchar(' ');

            clear_lines(scroll_first_line + drawn, scroll_last_line);

        } else {
            clear_lines(scroll_first_line, scroll_last_line);
            setCursorPosition(0, E404_POSY);
            puts(" Nothing to show");
        }
    }

    typedef enum { UP, DOWN } Vertical;
    void scroll_scroll(Vertical dir){
        if(!HEAD) return;
        if(scroll_selected_element->DIR(dir)){

            if(dir == UP && scroll_selected_element_pos == 0){
                                            scroll_first_element_on_screen   =
scroll_first_element_on_screen->PREV;
                scroll_selected_element = scroll_first_element_on_screen;
                redraw_scroll();
                  } else if(dir == DOWN && scroll_selected_element_pos ==
(scroll_last_line - scroll_first_line)){
                                            scroll_first_element_on_screen   =
scroll_first_element_on_screen->NEXT;
                scroll_selected_element = scroll_selected_element->NEXT;
                redraw_scroll();
            } else {

                  setColor(0, scroll_selected_element_pos + scroll_first_line,
buffer_info.dwSize.X - 1, buffer_info.wAttributes);

                // char offset[2] = { -1, 1 };
                scroll_selected_element_pos += dir * 2 - 1; //offset[dir];
                scroll_selected_element = scroll_selected_element->DIR(dir);

                  setColor(0, scroll_selected_element_pos + scroll_first_line,
buffer_info.dwSize.X - 1, BACKGROUND_WHITE);
            }
        } //else
          //  Beep(750, 100);
    }

    char draw_editor(ListElement * elem){
        unsigned char current_field = 0;
        char ret = ARROW_RIGHT;

        clear_lines(EDITOR_POSY - 1, EDITOR_POSY + 1);
        setCursorPosition(0, EDITOR_POSY);
        element_print(elem);
        //repeat(0, 3, buffer_info.dwSize.X, '_');
```

```
    setCursorVisibility(TRUE);

    loop {
        Field field = list_element_fields[current_field];
        char * offset = (char * ) elem + field.offset;
        ret = field.read_func(ret, field.posx, offset, field);

        if(ret == ARROW_LEFT){
            if(current_field == 0)
                current_field = len(list_element_fields) - 1;
            else
                current_field--;
        } else if(ret == KEY_ESC || ret == KEY_ENTER)
            break;
        else if(current_field == len(list_element_fields) - 1){
            if(ret == KEY_ENTER)
                break;
            else
                current_field = 0;
        } else
            current_field++;
    }

    setCursorVisibility(FALSE);
    clear_lines(EDITOR_POSY - 1, EDITOR_POSY + 1);

    return ret;
}

typedef struct {
    char * name;
    char   name_len;
    void (* func)();
} MenuItem;

void empty_func(){}

void menu_add();
void menu_remove();
void menu_edit();
void menu_sort();
void menu_search();
void menu_close_search();
void menu_export();
void menu_import();
void menu_process();

#define item(name, func) { name, len(name) - 1, func }
MenuItem menu_items[] = {
    item("+",  menu_add ),
    item("-",  menu_remove),
    item("Edit",  menu_edit),
    item("Search", menu_search),
    item("Sort", menu_sort),
    item("Export", menu_export),
    item("Import", menu_import),
    item("Save", empty_func),
    item("Process", menu_process),
    item(" X ", menu_close_search)
};

signed char selected_menu_item = 0;
```

```
void redraw_menu(){
    setCursorPosition( 0, 0);
    SetConsoleTextAttribute(stdout_handle, BACKGROUND_BLUE);
    for(unsigned char i = 0; i < len(menu_items); i++){
        if(i == selected_menu_item){
            SetConsoleTextAttribute(stdout_handle, BACKGROUND_GREEN);
            printf(" %s ", menu_items[i].name);
            SetConsoleTextAttribute(stdout_handle, BACKGROUND_BLUE);
        } else if(i == len(menu_items) - 1 && link_layer == SHOW)
            printf("     ");
        else
            printf(" %s ", menu_items[i].name);
    }
    SetConsoleTextAttribute(stdout_handle, buffer_info.wAttributes);
}

void draw_menu(){
    setColor(0, 0, buffer_info.dwSize.X, BACKGROUND_BLUE);
    redraw_menu();
}

typedef enum { LEFT, RIGHT } Horizontal;
void scroll_menu(Horizontal dir){

    unsigned char menu_len = len(menu_items) - (link_layer == SHOW);
    selected_menu_item += dir * 2 - 1;
    if(selected_menu_item < 0)
        selected_menu_item = menu_len - 1;
    else if(selected_menu_item == menu_len)
        selected_menu_item = 0;

    redraw_menu();
}

void redraw_header(unsigned char selected_header_item){
    setCursorPosition(0, HEADER_POSY);
    SetConsoleTextAttribute(stdout_handle, BACKGROUND_GRAY);
    for(unsigned char i = 0; i < len(list_element_fields); i++){
        if(i == selected_header_item){
            SetConsoleTextAttribute(stdout_handle, BACKGROUND_WHITE);
            printf(" %s ", list_element_fields[i].name);
            SetConsoleTextAttribute(stdout_handle, BACKGROUND_GRAY);
        } else
            printf(" %s ", list_element_fields[i].name);
    }
    SetConsoleTextAttribute(stdout_handle, buffer_info.wAttributes);
}

void draw_header(unsigned char selected_header_item){
    setColor(0, HEADER_POSY, buffer_info.dwSize.X, BACKGROUND_GRAY);
    redraw_header(selected_header_item);
}

char header_select_column(unsigned char * in_out){
    unsigned char selected_column = *in_out;
    int ch;
    loop {
        ch = _getch();
        if(ch == 224){
            ch = _getch();
            if(ch == ARROW_RIGHT){
```

```
                                    selected_column  =  (selected_column  +  1)  %
len(list_element_fields);
                        redraw_header(selected_column);
                    } else if(ch == ARROW_LEFT){
                                    selected_column  =  (selected_column  -  1)  %
len(list_element_fields);
                        redraw_header(selected_column);
                    }
                } else if(ch == KEY_ENTER || ch == KEY_ESC)
                    break;
            }
            *in_out = selected_column;
            return ch;
        }

        #endif

        // Agreement:
        // Main is a bridge between UI, List and Quotes
        // Defines should be undefined as soon as they not needed anymore
        // Naming format: class_name_method
        // And remember, not a word in Russian!


        #define BOTTOM_LINE (buffer_info.dwSize.Y - 1)
        #define RIGHT_CHAR  (buffer_info.dwSize.X - 1)

        void print_to_status(char * str){
            setCursorPosition(0, BOTTOM_LINE);
            printf(" %-50s", str);
        }

        void print_error_or_mistake(char * str){
            clear_lines(1, 3);
            mistake_quote();
            print_to_status(str);
        }

        void menu_add(){
            if(link_layer == SEARCH)
                return;

            print_to_status("Enter to submit. Esc to exit");

            char ret = draw_editor(last_readed);
            if(ret == KEY_ENTER){
                for(ListElement * cur = HEAD; cur; cur = cur->NEXT)
                                    if(cur->data.gradebook_number  ==  last_readed-
>data.gradebook_number){
                        print_error_or_mistake("Key already exist!");
                        return;
                    }
                list_add(last_readed);
                if(last_readed == HEAD)
                    scroll_set_head(HEAD);
                last_readed = list_element_new();
                element_zerofill(last_readed);
                start_quote();
                redraw_scroll();
                print_to_status("Element added");
            }
        }
```

```
      void menu_remove(){
          //if(link_layer == SEARCH)
          //    return;
          if(scroll_selected_element){
              ListElement * to_delete = scroll_selected_element;

              if(scroll_first_element_on_screen == HEAD){
                  // to_delete == scroll_first_element_on_screen
                  if(scroll_selected_element_pos == 0)
                      scroll_set_head(to_delete->NEXT);
                  else {
                      scroll_selected_element_pos--;
                      scroll_selected_element = to_delete->PREV;
                  }
              } else {
                                                  scroll_first_element_on_screen   =
scroll_first_element_on_screen->PREV;
                  scroll_selected_element = to_delete->PREV;
              }

              if(link_layer == SEARCH){
                  list_remove(to_delete);
                  link_layer = SHOW;
                  list_remove(to_delete);
                  link_layer = SEARCH;
              } else
                  list_remove(to_delete);

              remove_quote();
              redraw_scroll();
              print_to_status("Element removed");
          } else
              print_error_or_mistake("Nothing to remove");
      }

      void menu_edit(){
          if(scroll_selected_element){
              print_to_status("Editing element. Changes applied immidiately");
              draw_editor(scroll_selected_element);
              edit_quote();
              redraw_scroll();
              print_to_status("Element changed");
          } else
              print_error_or_mistake("Nothing to edit");
      }

      unsigned char sort_by_field = 0;
      void menu_sort(){
          if(list_len > 1){

              print_to_status("Use arrow keys to select table row to sort by");
              if(header_select_column(&sort_by_field) == KEY_ESC){
                  print_to_status("Aborted. Showing elements");
                  return;
              }

              merge_sort(sort_by_field);
              //scroll_set_head(HEAD);
              ListElement * cur = scroll_selected_element;
              for(unsigned int i = 0; i < scroll_selected_element_pos; i++){
                  if(cur->PREV)
```

```
                    cur = cur->PREV;
                else {
                    scroll_selected_element_pos = i;
                    break;
                }
            }
            scroll_first_element_on_screen = cur;
            start_quote();
            redraw_scroll();
            print_to_status("List sorted");
        } else
            print_error_or_mistake("Nothing to sort");
    }

    unsigned char search_by_field = 0;
    void menu_search(){
        redraw_header(search_by_field);

        print_to_status("Use arrow keys to select table row to search by");
        if(header_select_column(&search_by_field) == KEY_ESC){
            print_to_status("Aborted. Showing elements");
            goto exit;
        }

        Field field = list_element_fields[search_by_field];
        setCursorPosition(0, EDITOR_POSY);
        printf("Value: ");
        setCursorPosition(len("Value: ") + field.len, EDITOR_POSY);
        char mode = 'o';
        printf(" Mode: %c", mode);

        setCursorVisibility(TRUE);

        char ret = ARROW_RIGHT;
        char * field_value_ptr = (char * ) last_readed + field.offset;
        Field mode_field = { prop: { values: "\3o+-" } };

        loop {
             ret = field.read_func(ret, len("Value: ") - 1, field_value_ptr,
field);
            if(ret == KEY_ENTER)
                break;
            else if(ret == KEY_ESC)
                goto exit;
            ret = read_char(ret, len("Value: ") + field.len + len(" Mode: ") -
1, &mode, mode_field);
            if(ret == KEY_ENTER)
                break;
            else if(ret == KEY_ESC)
                goto exit;
        }

        setCursorVisibility(FALSE);

        link_layer = SEARCH;
        char field_size = field.size;
        size_t field_offset = field.offset;

        if((char *) field.read_func == (char *) read_string){
            field_size = field_value_ptr[0];
            field_offset++;
            field_value_ptr++;
```

```
        }

        if(mode == 'o'){
            heads[SEARCH] = NULL;
            tails[SEARCH] = NULL;
            list_lengths[SEARCH] = 0;
        }
        if(mode != '-')
            for(ListElement * cur = heads[SHOW]; cur; cur = cur->link[2])
                    if(memcmp((char *) cur + field_offset, field_value_ptr,
field_size) == 0)
                    list_add(cur);


        redraw_menu();
        scroll_set_head(heads[SEARCH]);
        redraw_scroll();
        print_to_status("Search completed");

    exit:
        clear_lines(1, 3);
        start_quote();
    }

    void menu_close_search(){
        link_layer = SHOW;
        selected_menu_item = 0;
        redraw_menu();
        start_quote();
        scroll_set_head(HEAD);
        redraw_scroll();
        print_to_status("Showing elements");
    }

    char read_filename(char * filename){
        clear_lines(EDITOR_POSY - 1, EDITOR_POSY + 1);
        setCursorPosition(0, EDITOR_POSY);
        printf("Filename: ");

        setCursorVisibility(TRUE);

            Field file_field = { len: FILEDATA_NAME_LEN, prop: { allow:
ALLOW_DIGITS | ALLOW_SPECIAL } };
        int ret = ARROW_RIGHT;
        loop {
                ret = read_string(ret, len("Filename: ") - 1, filename,
file_field);
            if(ret == KEY_ENTER || ret == KEY_ESC){
                print_to_status("Aborted. Showing elements");
                goto exit;
            }
        }

    exit:
        setCursorVisibility(FALSE);
        return ret;
    }

    void menu_export(){
        //link_layer = SHOW;
        char filename[FILEDATA_NAME_LEN] = {0};
        char ret = read_filename(filename);
```

```
    if(ret == KEY_ESC)
        goto exit;

    FILE * file = fopen(&filename[1], "w");
    if(!file){
        print_error_or_mistake("Error creating file");
        return;
    }

    for(ListElement * cur = HEAD; cur; cur = cur->NEXT)
        if(element_print_to_txt(file, cur) <= 0){
            print_error_or_mistake("Error writing to file");
            fclose(file);
            return;
        }

    fflush(file);
    fclose(file);
    print_to_status("Successfully exported");

exit:
    clear_lines(1, 3);
    start_quote();
}

void menu_import(){
    link_layer = SHOW;
    char filename[FILEDATA_NAME_LEN] = {0};
    char ret = read_filename(filename);
    if(ret == KEY_ESC)
        goto exit;

    FILE * file = fopen(&filename[1], "r");
    if(!file){
        print_error_or_mistake("Error opening file");
        return;
    }

    list_free();

    while(element_read_from_txt(file, last_readed) >= 0){
        list_add(last_readed);
        last_readed = list_element_new();
    }

    element_zerofill(last_readed);
    scroll_set_head(HEAD);
    redraw_scroll();
    print_to_status("Successfully imported");

exit:
    clear_lines(1, 3);
    start_quote();
}

void menu_process(){
    if(list_len > 1){
        list_process();
        redraw_menu();
        start_quote();
        scroll_set_head(HEAD);
        redraw_scroll();
```

```
        print_to_status("Task completed");
    } else {
        print_error_or_mistake("Nothing to process");
    }
}

int main(){

    //SetConsoleCP(866);
    //SetConsoleOutputCP(866);

    stdout_handle = GetStdHandle(STD_OUTPUT_HANDLE);

    GetConsoleCursorInfo(stdout_handle, &cursor_info);
    adjust_buffer();

    list_autoload(); // also allocs last_readed
    scroll_set_head(HEAD);
    element_zerofill(last_readed);

    setCursorVisibility(FALSE);
    clear_lines(0, buffer_info.dwSize.Y);
    draw_menu();
    start_quote();
    draw_header(0);
    scroll_last_line = BOTTOM_LINE - 1;
    draw_scroll();
    print_to_status("Showing elements");

    loop {

        //clear_lines(BOTTOM_LINE, BOTTOM_LINE);
        //setCursorPosition(0, BOTTOM_LINE);
        //printf("Showing elements...");

        int ch = _getch();
        if(ch == 224){
            ch = _getch();
            if(ch == ARROW_UP)
                scroll_scroll(UP);
            else if(ch == ARROW_DOWN)
                scroll_scroll(DOWN);
            else if(ch == ARROW_LEFT)
                scroll_menu(LEFT);
            else if(ch == ARROW_RIGHT)
                scroll_menu(RIGHT);
        } else if(ch == KEY_ENTER)
            menu_items[selected_menu_item].func();
        else if(ch == KEY_ESC)
            break;
    }

    clear_lines(0, buffer_info.dwSize.Y);
    setCursorPosition(0, 0);
    exit_quote();
    setCursorVisibility(TRUE);
    restore_buffer();
    list_autosave();
    list_release_memory();

    return 0;
}
```