АННОТАЦИЯ

В пояснительной записке представлено описание курсовой работы по дисциплине «Алгоритмизация и программирование». Описаны использованные функции, методы и структура разработанной программы, приведено описание работы приложения.

В первом разделе описана область назначения и применения программы. Второй раздел посвящен техническим характеристикам программы. В третьем разделе приведено выполнение программы.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc56851295)

[1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ 4](#_Toc56851296)

[2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ 5](#_Toc56851297)

[2.1 Постановка задачи 5](#_Toc56851298)

[2.2 Описание входных, выходных и промежуточных данных 5](#_Toc56851300)

[2.3 Описание и обоснование выбора средств разработки 6](#_Toc56851301)

[2.4 Описание алгоритмов функционирования программы 7](#_Toc56851303)

[3 ВЫПОЛНЕНИЕ ПРОГРАММЫ 27](#_Toc56851304)

[3.1 Условия выполнения программы 27](#_Toc56851306)

[3.2 Проверка работоспособности программы 27](#_Toc56851306)

[ЗАКЛЮЧЕНИЕ 40](#_Toc56851308)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 41](#_Toc56851309)

[ПРИЛОЖЕНИЕ А (КОД ПРОГРАММЫ) 42](#_Toc56851310)

# введение

Цель данного курсового проекта – разработать программу, которая представляет собой упрощённую информационную систему, осуществляющую обработку списка данных о студентах университета.

Тема данного курсового проекта – разработка программы, возможной для применения в учебных заведениях при составлении и обработке набора данных о тех, кто обучается в нём.

Основная задача курсового проекта – оптимизировать обработку данных в организации, создать удобный механизм занесения сведений в базу данных и проведения последующих операций с ними.

# 1 Назначение и область применения программы

Область применения программы, являющейся сущностью курсового проекта, – учёт данных в университетах или иных других организациях, предоставляющих образовательные услуги. Программа позволяет систематизировать сведения о студентах, поступивших на обучение в организацию в различный период времени.

# 2 технические характеристики программы

## 2.1 Постановка задачи

Выбрать структуру данных для хранения информации (бинарное дерево или двухсвязный список), обосновать причину выбора. На языке программирования C / C++ реализовать обработку элементов списка, представленных в виде таблицы.

Каждый элемент списка представляет собой структуру данных, содержащую информацию о студенте (данные студента): ФИО, номер зачётной книжки, группа обучения, дата рождения, год поступления и вступительный балл. Реализовать операции над списком: инициализацию списка (вручную и из файла, текстового или бинарного), вывод таблицы с элементами списка, добавление и удаление новых элементов, сортировку (минимум по одному из полей), поиск, перезапись (редактирование) элемента, вывод пяти студентов с наивысшим вступительным баллом за каждый из годов поступления, зафиксированный в списке, сохранение списка в файл с возможностью считывания его из этого файла внутри программы.

## 2.2 Описание входных, выходных и промежуточных данных

В качестве структуры данных для программы был выбран двухсвязный список. Список является более простой структурой данных, его нетрудно представить в виде таблицы, поскольку очерёдность элементов в нём всегда очевидна. Бинарные деревья же требуют более трудоёмкой проработки всех функций программы, являясь более запутанной структурой данных. Основное преимущество деревьев над списками – быстродействие поиска среди элементов, однако сложные алгоритмы обработки, затрачивающие дополнительные мощности компьютера, в рамках данной работы нивелируют его перед двухсвязными списками.

Входные данные программы представлены указателем на структуру List, состоящей из информационного поля (структура Student) и указателей на предыдущий и последующий элементы списка (также указатели на List).

В информационном поле элемента списка (структура Student) содержатся порядковый номер num элемента в списке (индекс, ключевое поле при обработке), полное имя студента FIO типа структуры Initial, номер зачётной книжки record\_num, группа обучения group, дата рождения студента birth типы структуры Date, год поступления студента в организацию entry\_year и вступительный балл entry\_score.

Структура Initial содержит три поля класса string: surname (фамилия), name (имя), patronym (отчество). Структура Date содержит поля типа unsigned short: day (день), month (месяц), year (год).

Также в процессе работы используются двусвязные циклические списки команд, являющиеся указателями на структуры DirList. Структура содержит код команды и указатели на следующий и предыдущий элементы того же списка.

Фрагмент кода для структуры списка:

Struct List {

Struct Student inf;

struct List\* prev;

struct List\* next;

};

Фрагмент кода для информационного поля элемента списка:

struct Student {

int num;

unsigned int record\_num;

Initial FIO;

string group;

Date birth;

unsigned short entry\_year;

unsigned short entry\_score;

};

Фрагмент кода для структуры списка команд:

struct DirList {

int num;

struct DirList\* prev;

struct DirList\* next;

};

Поле с информацией об элементе содержит следующие поля:

1. ФИО студента
2. Номер зачётной книжки
3. Группа обучения
4. Дата рождения
5. Год поступления
6. Вступительный балл

## 2.3 Описание и обоснование выбора средств разработки

Для разработки программы был использован язык программирования C++. Разработка велась в среде Microsoft Visual Studio.

Данный выбор был обусловлен тем, что Visual Studio – мощная среда разработки, содержащая массу преимуществ при написании кода: субъективно приятный интерфейс, подсветка синтаксиса, подборка вариантов для автоматического дополнения кода, широкие возможности при отладке программы.

Выбор языка C++ обусловлен тем, что он предоставляет возможность работы на низком уровне с памятью и адресами. Также для языка C++ встроено несколько библиотек, значительно упрощающих код на нём относительно родственного ему языка C.

## 2.4 Описание алгоритмов функционирования программы

При разработке программы были описаны следующие функции:

1. startMenu() – начальное меню, через которое можно выбрать способ инициализации списка;
2. tableMenu() – основное меню работы со списком, через которое его можно просматривать и редактировать;
3. tableMenuEsq() – проверяет был ли сохранён список перед выходом из основного меню работы с ним;
4. fillCommArr() – заполняет соответствующий список команд кодами, указанными через массив целых чисел;
5. selectCommand() – вывод списка команд, указанных в соответствующем списке команд, и выбор дальнейшего действия программы;
6. createList() – создание списка или добавление элемента в конец через ввод нового элемента;
7. fillStudentNote() – заполнение информационного поля элемента списка;
8. getStr() – ввод строки фиксированной длины с проверкой на отсутствие пробелов и то, чтобы она не была пустой;
9. getInt() – ввод положительно числа с проверкой на программные ошибки и не выход за пределы максимума;
10. getLastNote() – поиск последнего элемента (хвоста) списка;
11. addNote() – присоединение нового элемента к концу списка;
12. printList() – вывод страницы списка в виде таблицы;
13. findNotePos() – поиск позиции элемента в списке (на случай если позиция не совпадает с индексом);
14. printNote() – вывод строки таблицы с элементом списка или пустой строки;
15. printDate() – перевод даты (структура Date) в строку;
16. deleteNote() – удаление элемента из списка;
17. rewriteNote() – полная перезапись элемента;
18. editNote() – редактирование полей элемента;
19. sortList() – выбор направления сортировки списка по ФИО;
20. sortListAscend() – сортировка списка по возрастанию ФИО;
21. sortListDescend() – сортировка списка по убыванию ФИО;
22. initcmp() – сравнение ФИО (структуры Initial);
23. searchList() – поиск в списке элементов по запросу;
24. isSearched() – проверка элемента списка на соответствие запросу;
25. printBestList() – вывод списка лучших студентов по годам;
26. sortInt() – сортировка целочисленного массива;
27. fillEntryYearsArr() – заполнение массива всех упомянутых в списке годов поступления;
28. sortBestList() – сортировка списка по убыванию вступительного балла;
29. bestForYear() – вывод 5 лучших студентов по годам поступления и переключение между страницами;
30. saveBestList() – подготовка списка лучших студентов для сохранения в файл;
31. saveToFile() – сохранение списка в файл (выбор функции относительно типа файла или возврат ошибки);
32. saveToFileTxt() – сохранение в текстовый файл (.txt);
33. saveToFileBin() – сохранение в бинарный файл (.dat или .bin);
34. loadFromFile() – инициализация списка из файла (выбор функции относительно типа файла или возврат ошибки);
35. loadFromFileTxt() – инициализация списка из текстового файла;
36. loadElementFromTxt() – считывание элемента списка из текстового файла;
37. loadFromFileBin() – инициализация списка из бинарного файла;
38. loadElementFromTxt() – считывание элемента списка из бинарного файла;
39. cleanMemory() – освобождение памяти под список List;
40. cleanMemoryDir() – освобождение памяти под список команд DirList;

Структурная схема функции main приведена на рисунке 2.1.

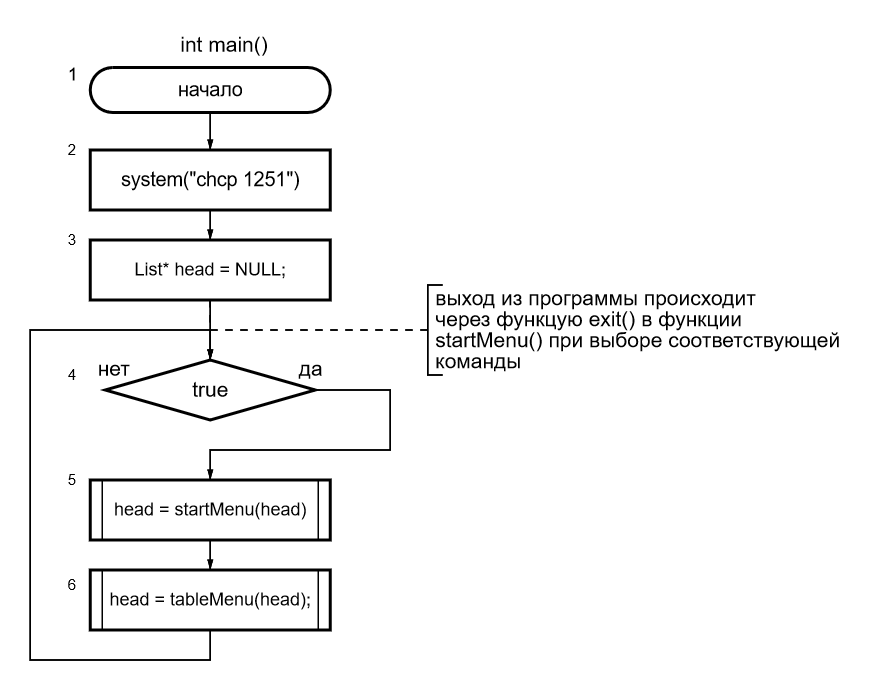


Рисунок 2.1 – Схема алгоритма функции main

Поблочное описание функции main:

Блок 1 – описание функции; 2 – установка кодировки для правильного вывода кириллицы; блок 3 – вечный цикл, выход из программы происходит в функции startMenu() путём вызова exit(0); блок 4 – переход в начальное меню для инициализации списка; блок 5 – переход в основное меню для просмотра и обработки списка.

Структурная схема функции startMenu приведена на рисунке 2.2.

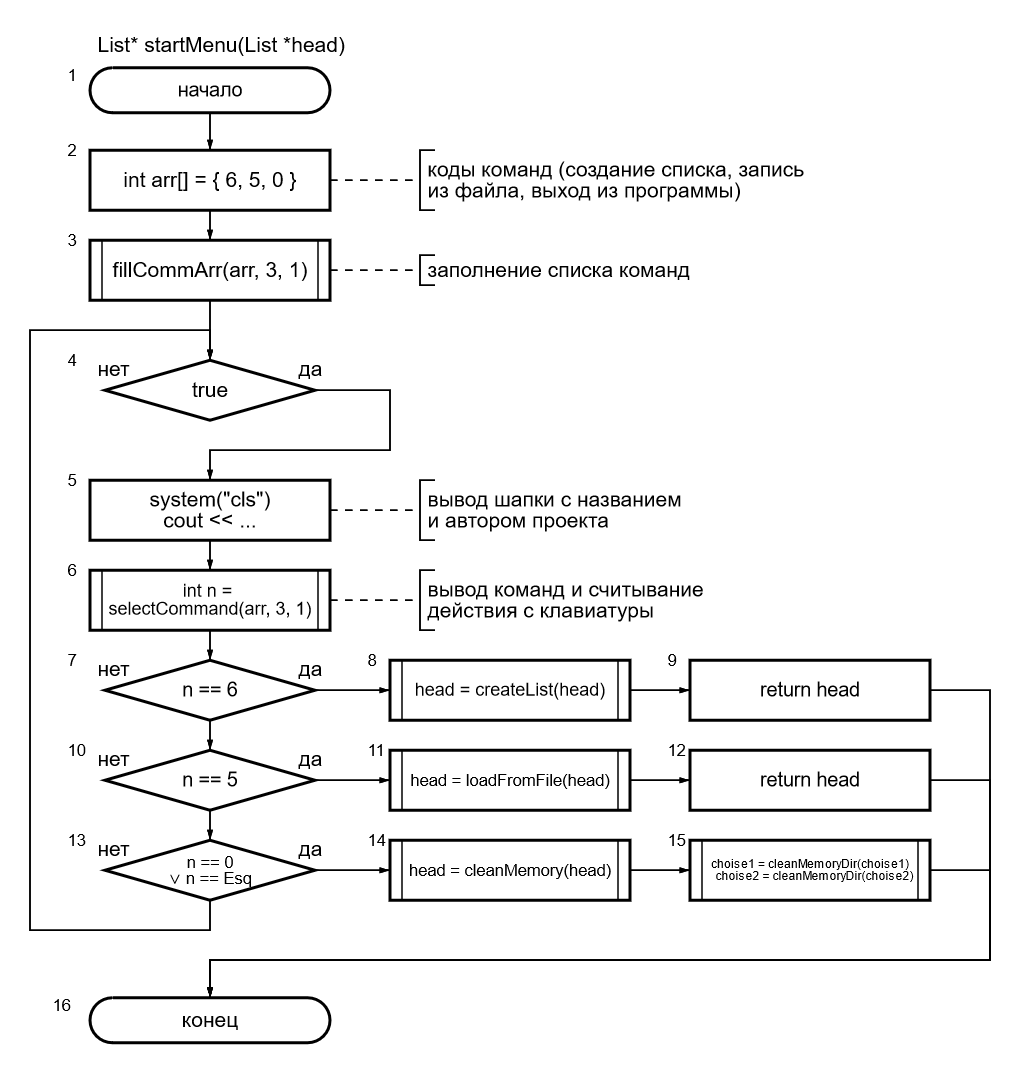


Рисунок 2.2 – Схема алгоритма функции startMenu

Поблочное описание функции startMenu:

Блок 1 – описание функции и передача параметра (указатель на список); блок 2 – заполнение массива кодов команд; блок 3 – заполнение списка команд; блок 4 – вечный цикл; блок 5 – вывод шапки с информацией о программе; блок 6 – вывод команд и считывание действия с клавиатуры; блок 7 – если считана команда создания списка; блок 8 – инициализация списка; блок 9 – возвращает указатель на созданный список; блок 10 – если считана команда загрузки списка из файла; блок 11 – считывание списка с файла; блок 12 – возвращает указатель на список; блок 13 – если считана команда выхода из программы; блок 14 – очистка памяти под список; блок 15 – очистка памяти под списки команд; блок 16 – конец функции.

Структурная схема функции tableMenu приведена на рисунке 2.3.

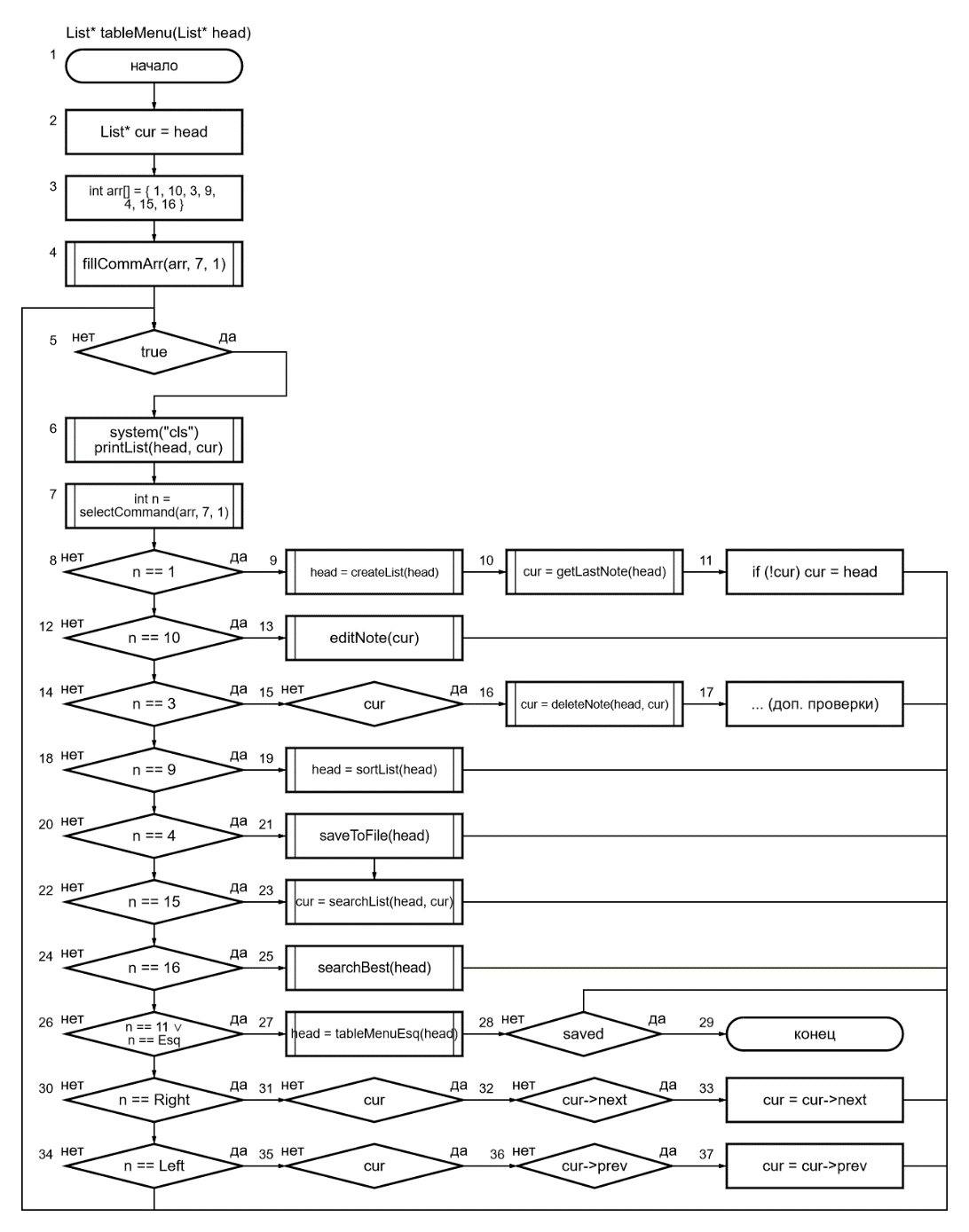


Рисунок 2.3 – Схема алгоритма функции tableMenu

Поблочное описание функции tableMenu:

Блок 1 – описание функции; блок 2 – инициализация указателя на текущий элемент списка; блок 3 – заполнение массива кодов команд; блок 4 – заполнение списка команд; блок 5 – вечный цикл; блок 6 – вывод страницы списка; блок 7 – вывод команд и считывание действия с клавиатуры; блок 8, 9, 10, 11 – проверка на пустой список и соответствующие проверки; блок 12, 13 – редактирование элемента; блок 14, 15, 16, 17 – удаление элемента и соответствующие проверки; блок 18, 19 – сортировка списка; блок 20, 21 – сохранение в файл; блок 22, 23 – поиск в списке; блок 24, 25 – поиск лучших по годам; блок 26, 27, 28, 29 – выход из программы если список сохранён или пользователю это не важно; блок 30, 31, 32, 33 – перевод текущего элемента вперёд; блок 34, 35, 36, 37 – перевод текущего элемента назад.

Структурная схема функции fillCommArr приведена на рисунке 2.4.

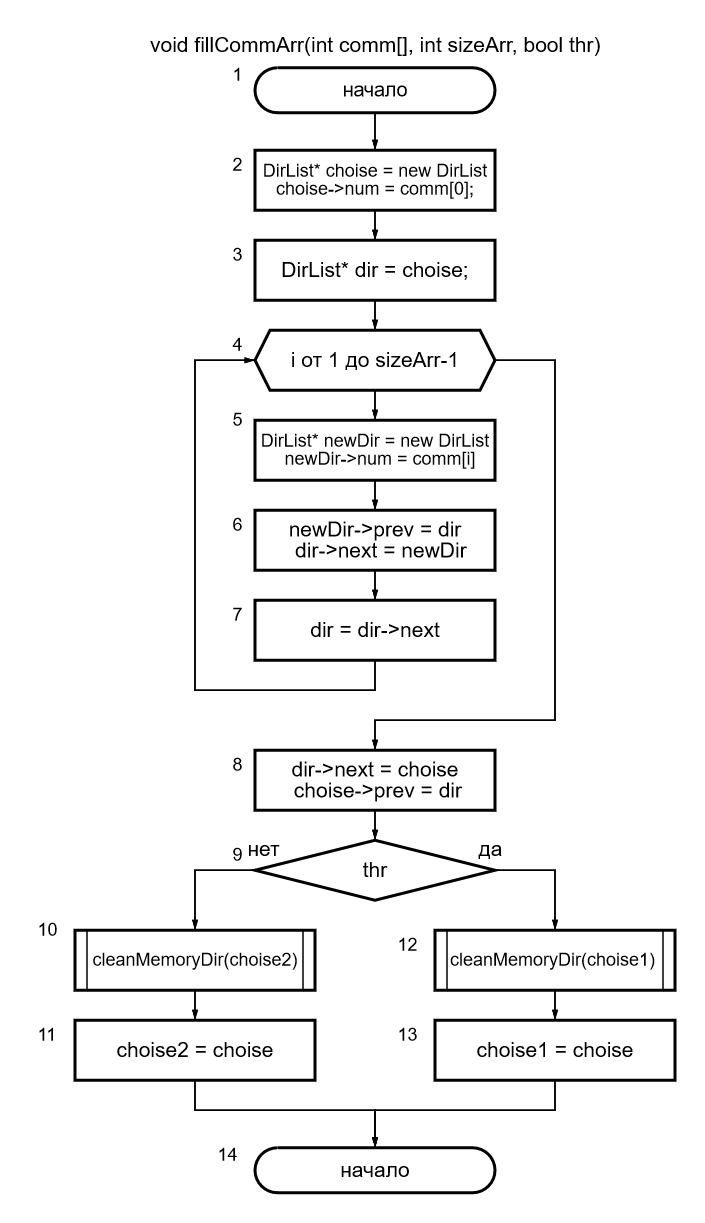


Рисунок 2.4 – Схема алгоритма функции fillCommArr

Поблочное описание функции fillCommArr:

Блок 1 – описание функции и параметров; блок 2 – инициализация первого элемента списка команд; блок 3 – переменная для добавления новых элементов в список; блок 4 – цикл обхода списка кодов; блок 5, 6, 7 – внесение кода в список команд; блок 8 – зацикливание списка; блок 9 – выбор глобального указателя; блок 10, 11 – выбор второго указателя (побочные команды); блок 12, 13 – выбор первого указателя (основные команды); блок 14 – конец функции.

Структурная схема функции selectCommand приведена на рисунке 2.5.

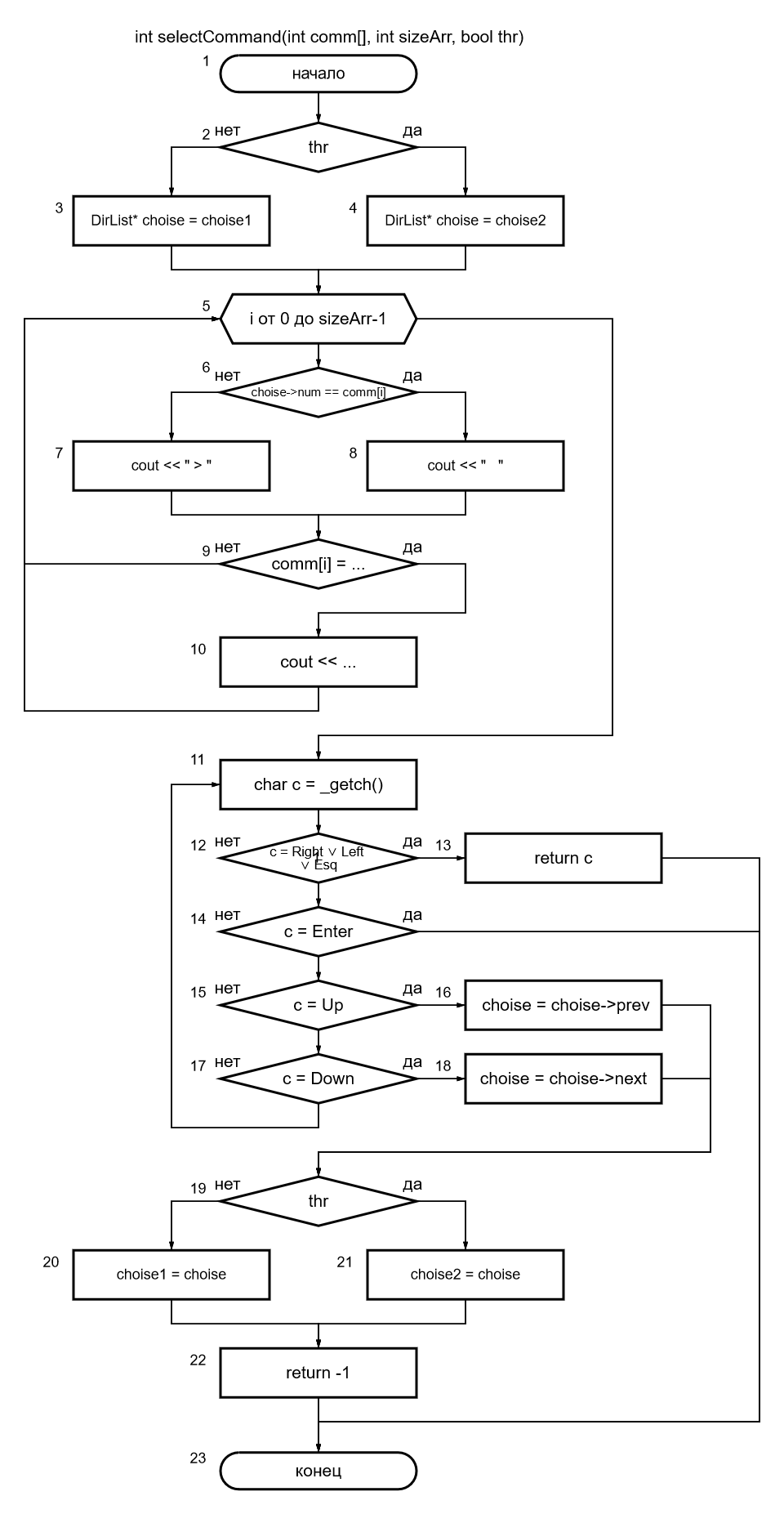


Рисунок 2.5 – Схема алгоритма функции selectCommand

Поблочное описание функции selectCommand:

Блок 1 – описание функции; блок 2, 3, 4 – выбор указателя на список; блок 5 – цикл вывода команд; блок 6, 7, 8 – вывод стрелки для текущей команды или пробела; блок 9, 10 – в зависимости от кода команды выводить определённую строчку, блок 11 – считывание действия; блок 12, 13 – команды управления списком команд; блок 14 – ввод текущей команды; блок 15, 16, 17, 18 – команды управления списком студентов; блок 19, 20, 21 – выбор указателя на список; блок 22 – команда не была выбрана; блок 23 – конец функции.

Структурная схема функции createList приведена на рисунке 2.6.

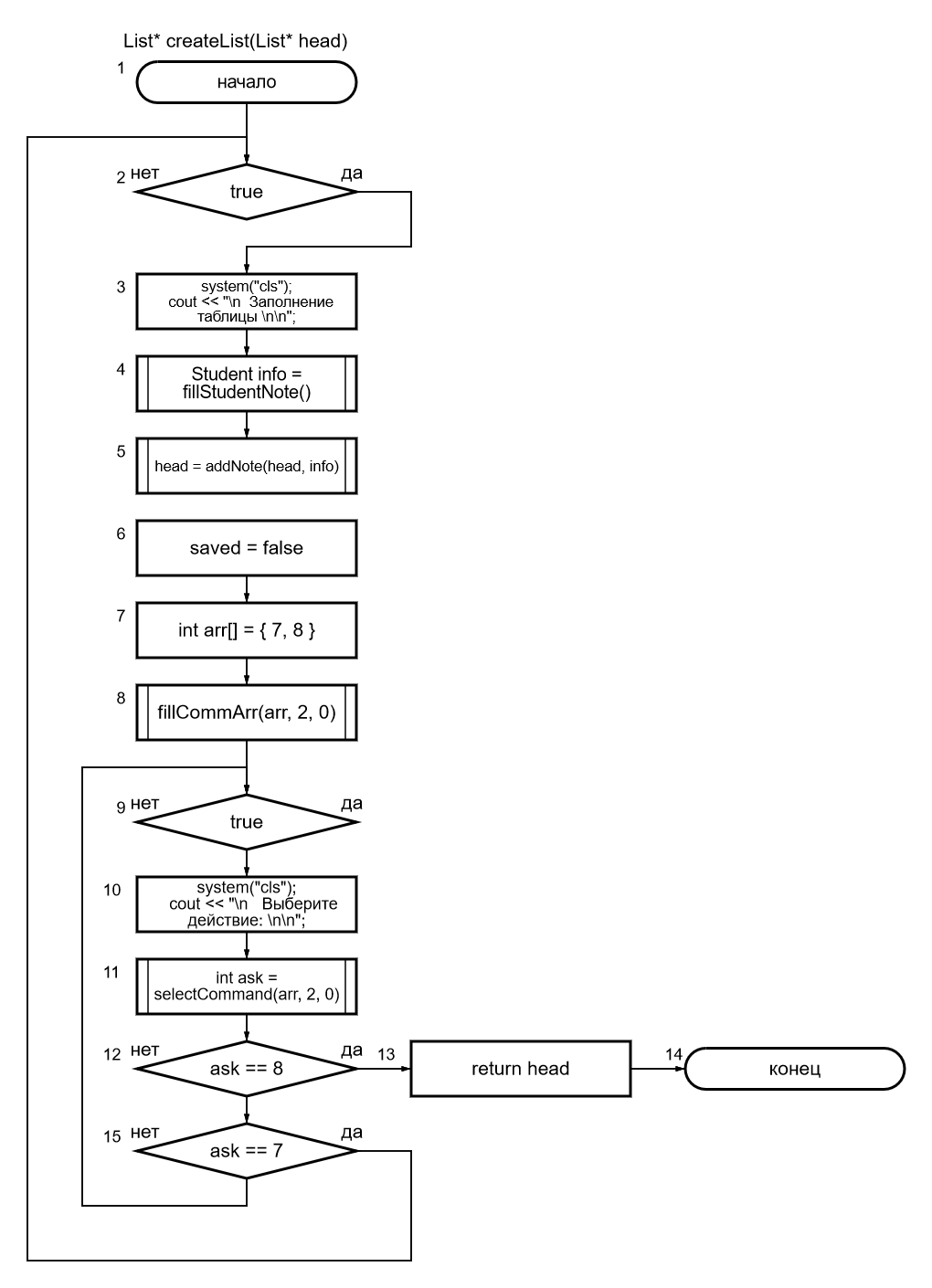


Рисунок 2.6 – Схема алгоритма функции createList

Поблочное описание функции createList:

Блок 1 – описание функции; блок 2 – вечный цикл ввода; блок 3 – вывод заголовка; блок 4 – заполнение информационного поля; блок 5 – добавление элемента в конец списка; блок 6 – флаг сохранения; блок 7 – список кодов команд; блок 8 – заполнение списка команд; блок 9 – вечный цикл выбора команды; блок 10 – вывод заголовка; блок 11 – считывание команды; блок 12, 13, 14 – окончание ввода списка; блок 15 – продолжение ввода элементов.

Структурная схема функции printList приведена на рисунке 2.7.

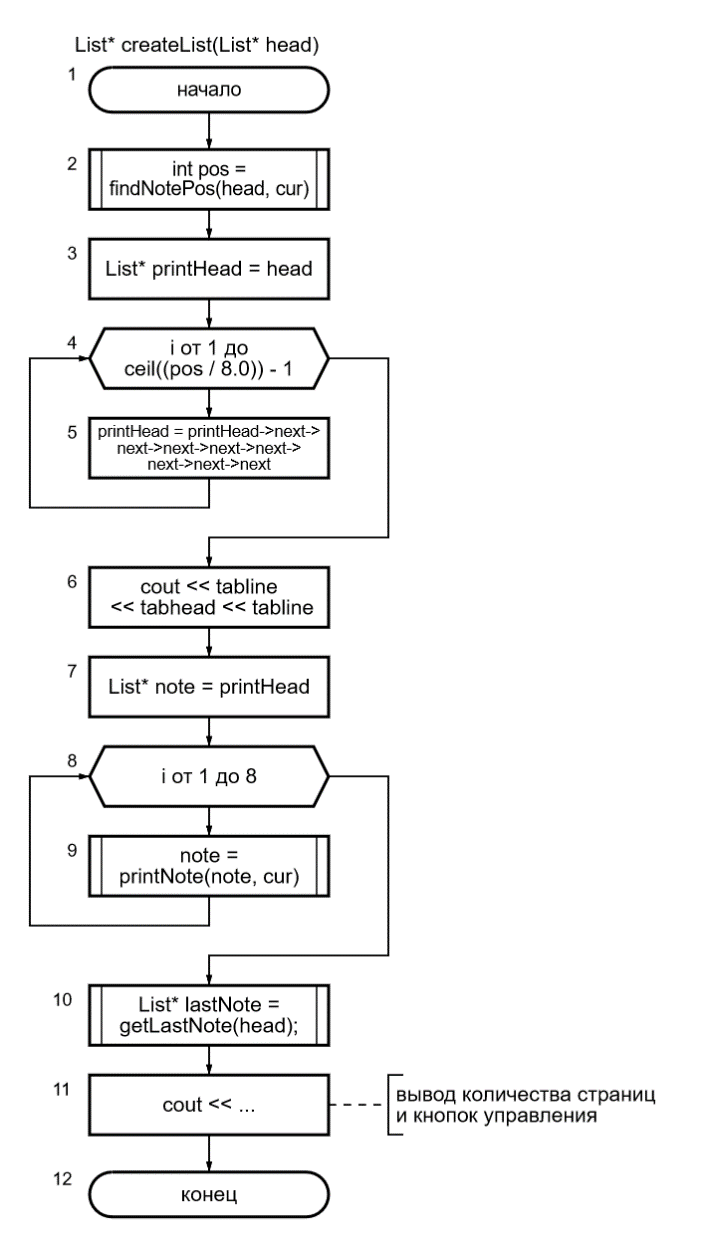


Рисунок 2.7 – Схема алгоритма функции printList

Поблочное описание функции printList:

Блок 1 – описание функции; блок 2 – поиск позиции текущего элемента; блок 3, 4, 5 - установка первого списка на странице; блок 6 – вывод шабки таблицы; блок 7, 8, 9 – вывод восьми элементов списка; блок 10, 11 – дополнительная строка с номером страницы; блок 12 – конец функции.

Структурная схема функции rewriteNote приведена на рисунке 2.8.

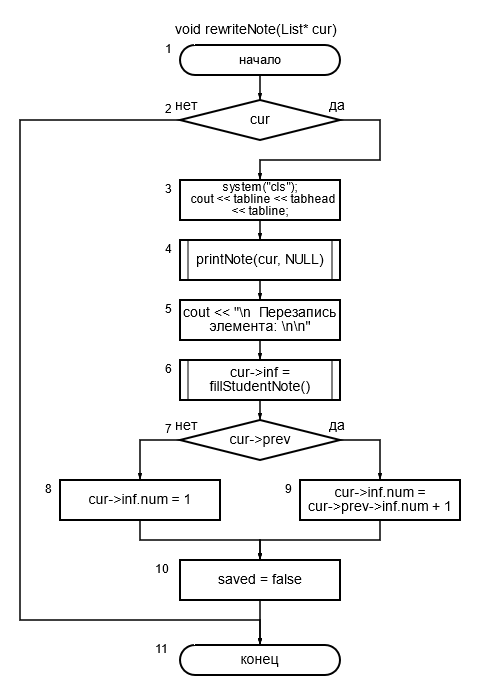


Рисунок 2.8 - Схема алгоритма функции rewriteNote

Поблочное описание функции rewriteNote:

Блок 1 – описание функции; блок 2 – проверка наличия элемента; блок 3 – вывод заголовка; блок 4 – вывод элемента; блок 5 – вывод заголовка; блок 6 – перезапись информационного поля; блок 7, 8, 9 – запись индекса относительно предыдущего элемента; блок 10 – флаг сохранения; блок 11 – конец функции.

Структурная схема функции deleteList приведена на рисунке 2.9.

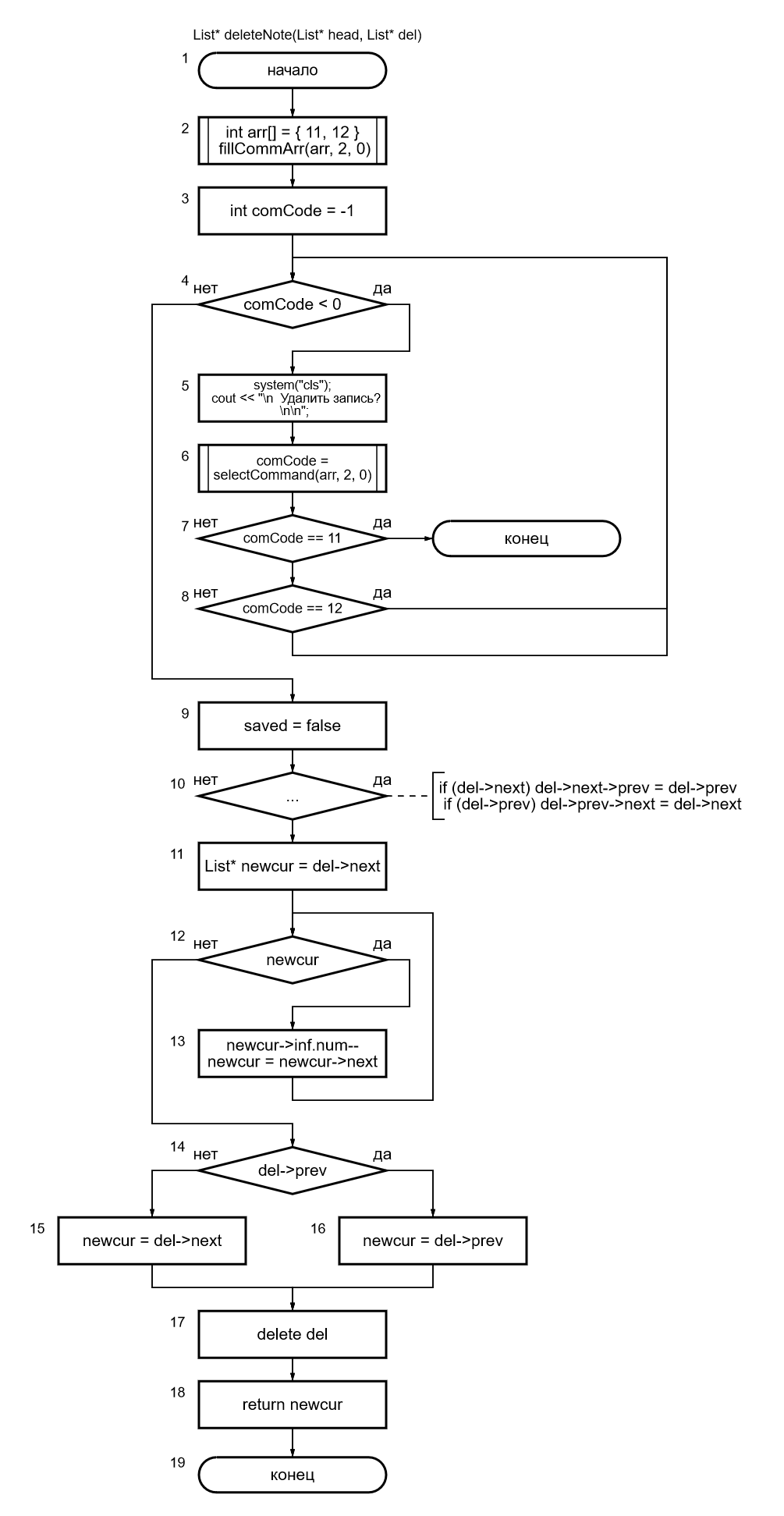


Рисунок 2.9 – Схема алгоритма функции deleteNote

Поблочное описание функции deleteNote:

Блок 1 – описание функции; блок 2 – заполнение списка команд; блок 3, 4 – цикл продолжается пока не введёшь команду; блок 5 – вывод заголовка; блок 6 – выбор команды; блок 7 – отмена удаления; блок 8 – подтвердить удаление; блок 9 – флаг сохранения; блок 10 – связывание разорванного списка; блок 11, 12, 13 – изменить индексы последующих элементов; блок 14, 15, 16 - переместить текущий в зависимости от позиции; блок 17 – удаление элемента; блок 18 – вернуть указатель; блок 19 – конец функции.

Структурная схема функции sortList приведена на рисунке 2.10.

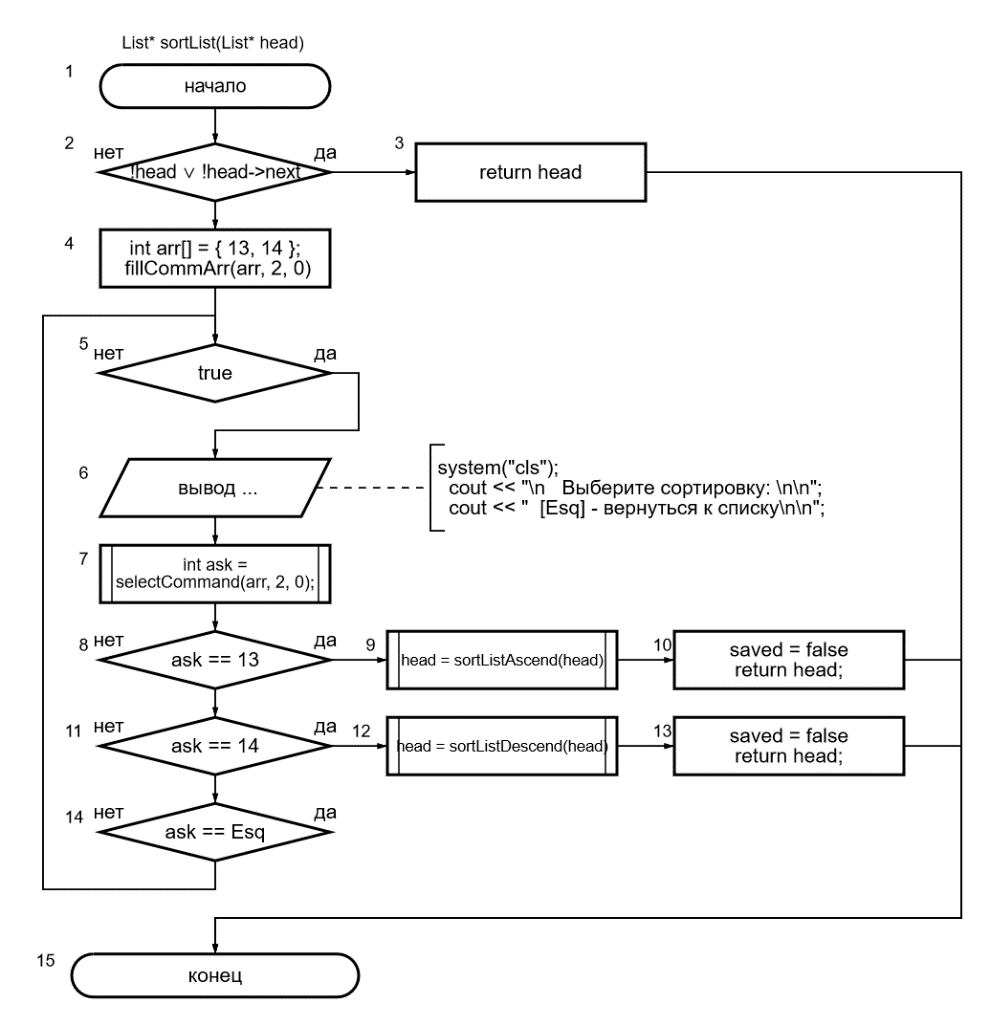


Рисунок 2.10 - Схема алгоритма функции sortList

Поблочное описание функции sortList:

Блок 1 – описание списка; блок 2, 3 – список отсутствует; блок 4 – заполнение списка команд; блок 5 – вечный список; блок 6 – вывод заголовка; блок 7 – вывод команд и выбор действия; блок 8, 9, 10 – сортировка по возрастанию; блок 11, 12, 13 – сортировка по убыванию; блок 14 – выход из сортировки; блок 15 – конец функции.

Структурная схема функции clearMemory приведена на рисунке 2.11.

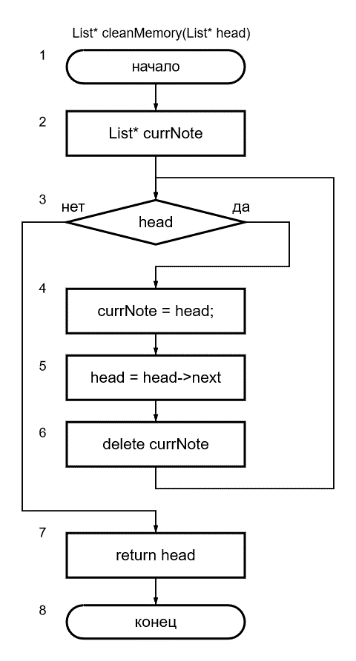


Рисунок 2.11 - Схема алгоритма функции cleanMemory

Поблочное описание функции cleanMemory:

Блок 1 – описание функции; блок 2 – вспомогательный указатель; блок 3 – обход списка; блок 4, 5, 6 – удаление элемента и переход к следующему; блок 7 – возврат указателя; блок 8 – конец функции.

Структурная схема функции clearMemoryDir приведена на рисунке 2.12.

Поблочное описание функции cleanMemoryDir:

Блок 1 – описание функции; блок 2 – вспомогательный указатель; блок 3 – обход списка; блок 4, 5, 6 – удаление элемента и переход к следующему; блок 7 – возврат указателя; блок 8 – конец функции.

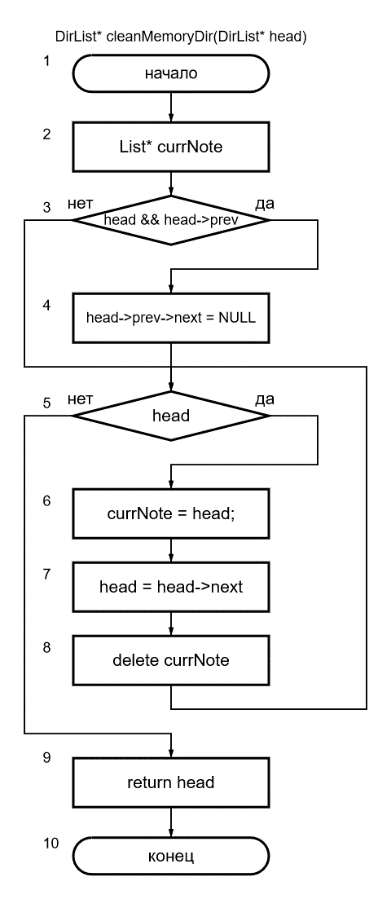


Рисунок 2.12 - Схема алгоритма функции cleanMemoryDir

# 3 выполнение программы

## 3.1 Условия выполнения программы

Написание программы осуществлялось на ноутбуке средних характеристик. Основное комплектующее – процессор AMD Ryzen 7 5700U. На ноутбуке студента с временем выполнения программы не возникало проблем. Предполагается, что другие на других ЭВМ программа выполняется с тем же успехом в силу невысокой затраты ресурсов.

Минимальные системные требования для корректной работы программы:

Минимальные системные требования для корректной работы программы:

1. Процессор с тактовой частотой не ниже 1,8 ГГц.
2. 2 ГБ оперативной памяти

Размер текстового файла (данные одного продукта) **≈** 21 байт

Размер типизированного файла (данные одного продукта) **≈** 54 байт

## 3.2 Проверка работоспособности программы

Для запуска программы достаточно иметь exe-файл или среду Microsoft Visual C++ для компиляции и запуска исходного кода.

После успешного запуска программы пользователю предлагают выбрать единственную команду для инициализации данных (Рисунок 3.1).

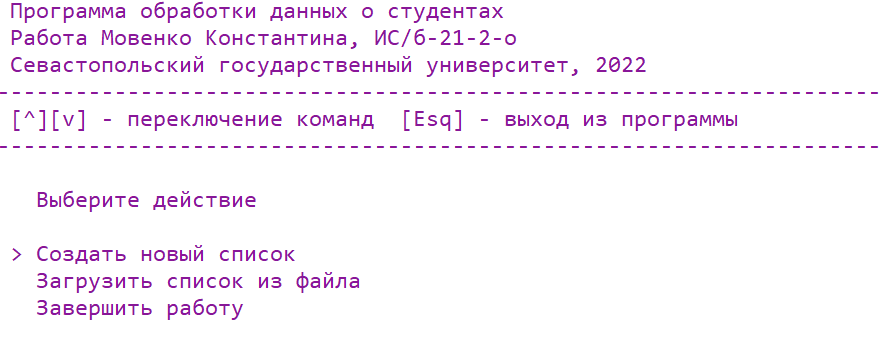


Рисунок 3.1 – Начальное меню

В начальном меню пользователю предлагается выбрать способ инициализации списка – вручную или чтением из файла. Выбор команды осуществляется через скроллинг клавишами вверх-вниз (Рисунок 3.2).

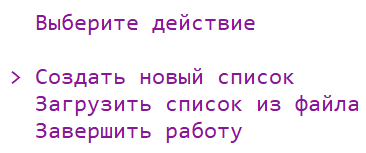


Рисунок 3.2 – Выбор типа инициализации данных

При выборе первого пункта пользователю предлагается вручную ввести данные для студента (Рисунок 3.3).

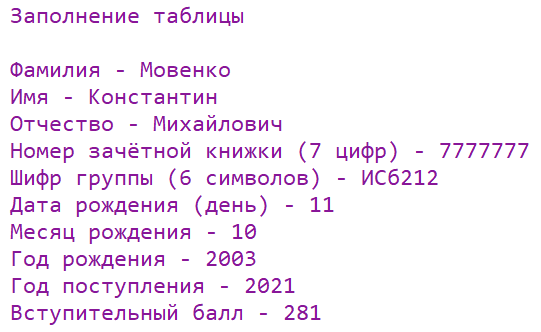


Рисунок 3.3 – Ввод данных для студента

После заполнения данных пользователю предлагают добавить в список ещё одного студента с возможностью отказаться (Рисунок 3.4).

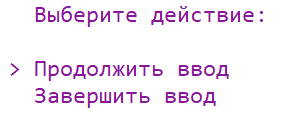


Рисунок 3.4 – Запрос на добавление еще одного студента

При выборе первой команды процесс заполнения данных повторяется. При выборе второй осуществляется переход в меню просмотра и обработки списка (Рисунок 3.5).

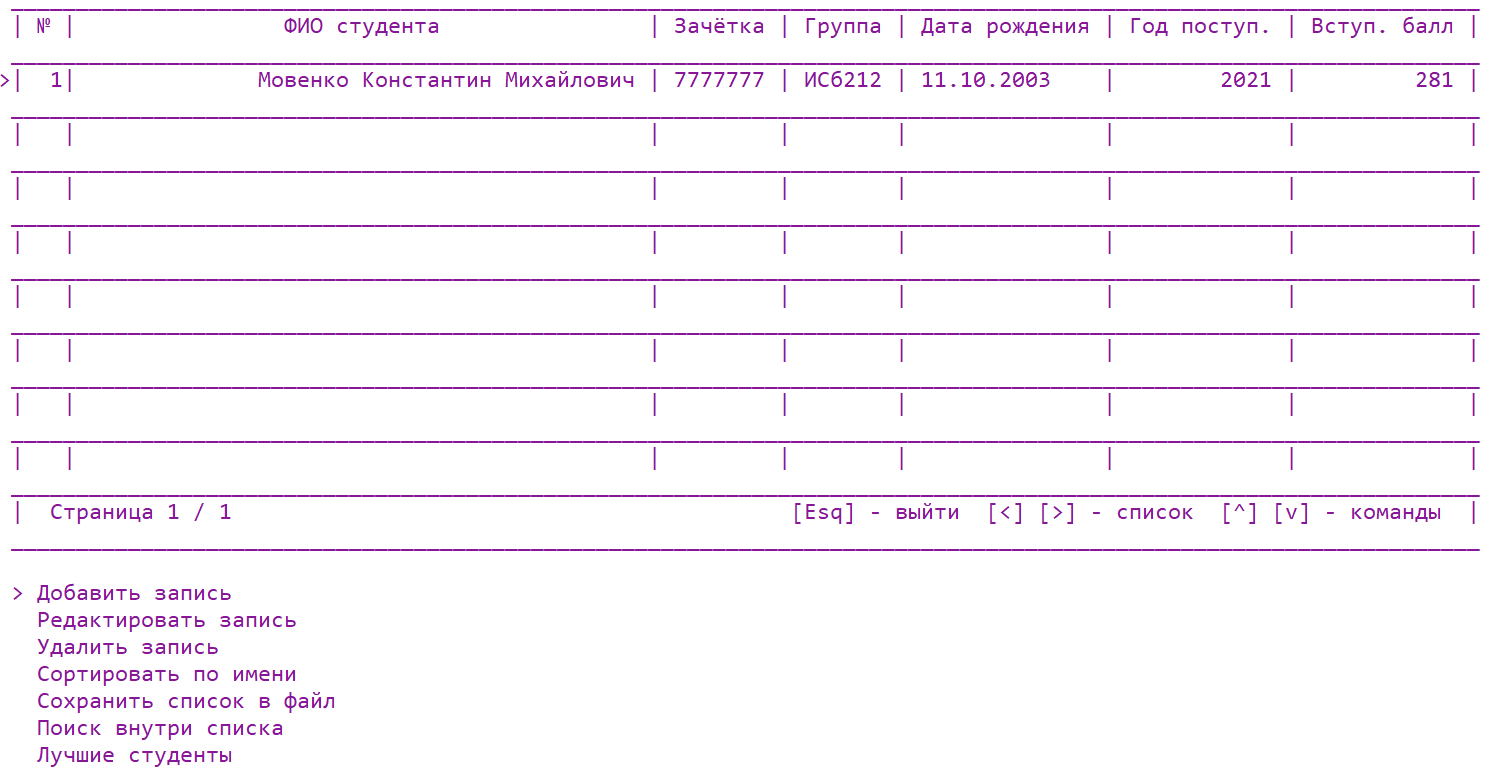


Рисунок 3.5 – Меню обработки списка

При попытке выйти из этого меню через кнопку Escape выводится сообщение о том, что список не сохранён. В нём можно выбрать два варианта: выйти из списка без сохранения или вернуться в меню обработки (Рисунок 3.6).

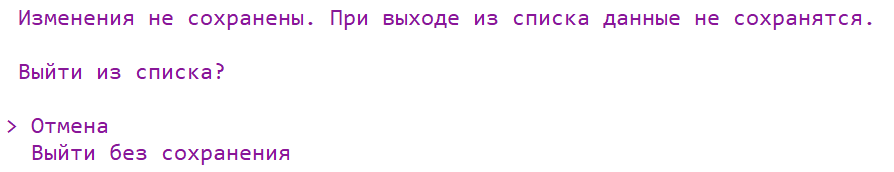


Рисунок 3.6 – Выход из меню обработки при не сохранённом списке

Если в главном меню выбрать команду инициализации списка из файла программа предложит пользователю ввести имя или полный адрес файла, из которого будет производиться ввод (Рисунок 3.7). Выбрать можно файлы формата txt, dat, bin.

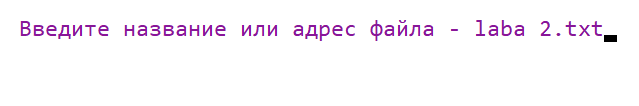


Рисунок 3.7 – Ввод имени файла

При наличии более одного элемента списка в меню обработки списка можно производить скроллинг списка по страницам посредством сдвига текущего элемента, а также скроллинг списка команд (Рисунок 3.8).

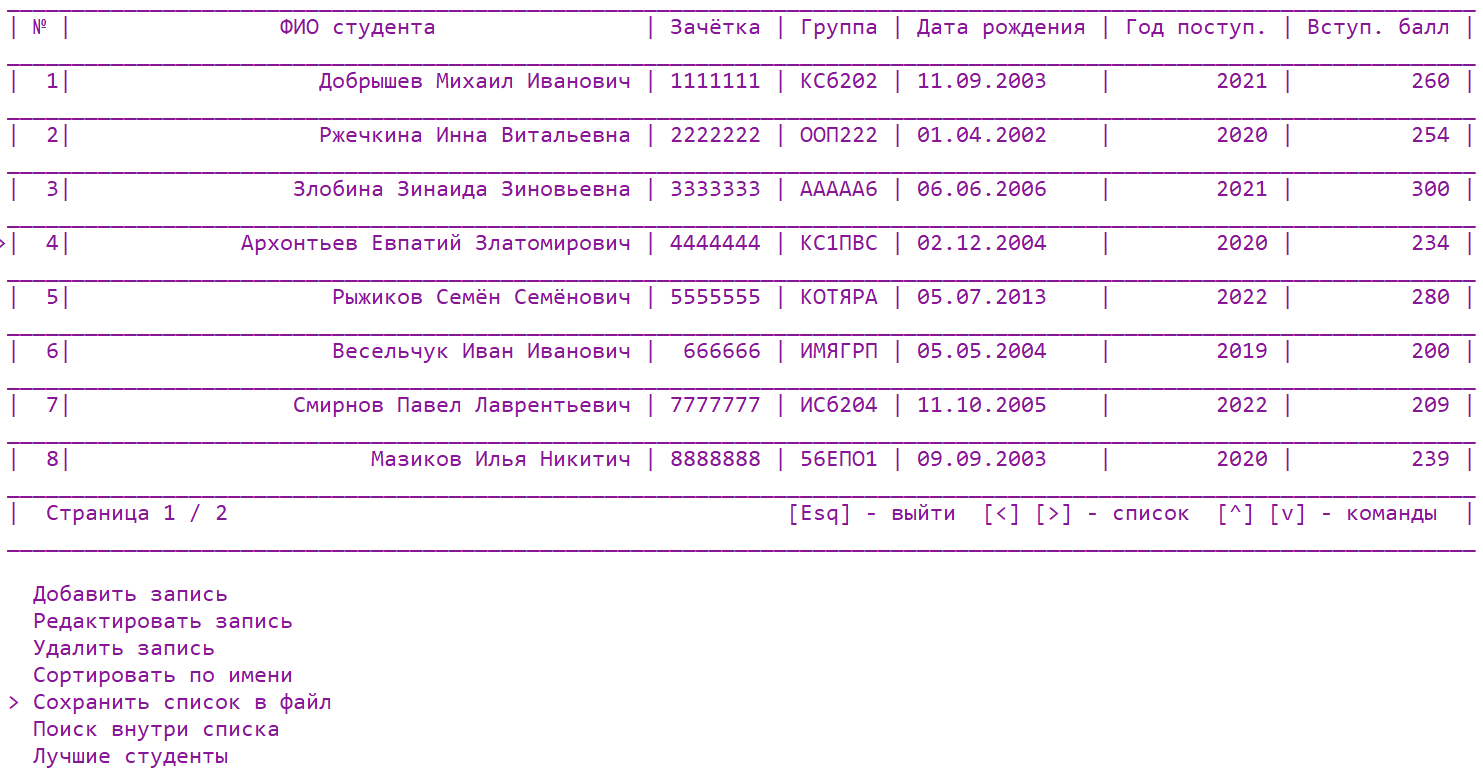


Рисунок 3.8 – Скроллинг меню обработки

При выборе первой команды запустится продемонстрированный выше алгоритм ввода нового элемента. При выборе второй команды пользователь сможет изменить одно из полей элемента списка (на выбор), либо сразу полностью перезаписать элемент (Рисунок 3.9).

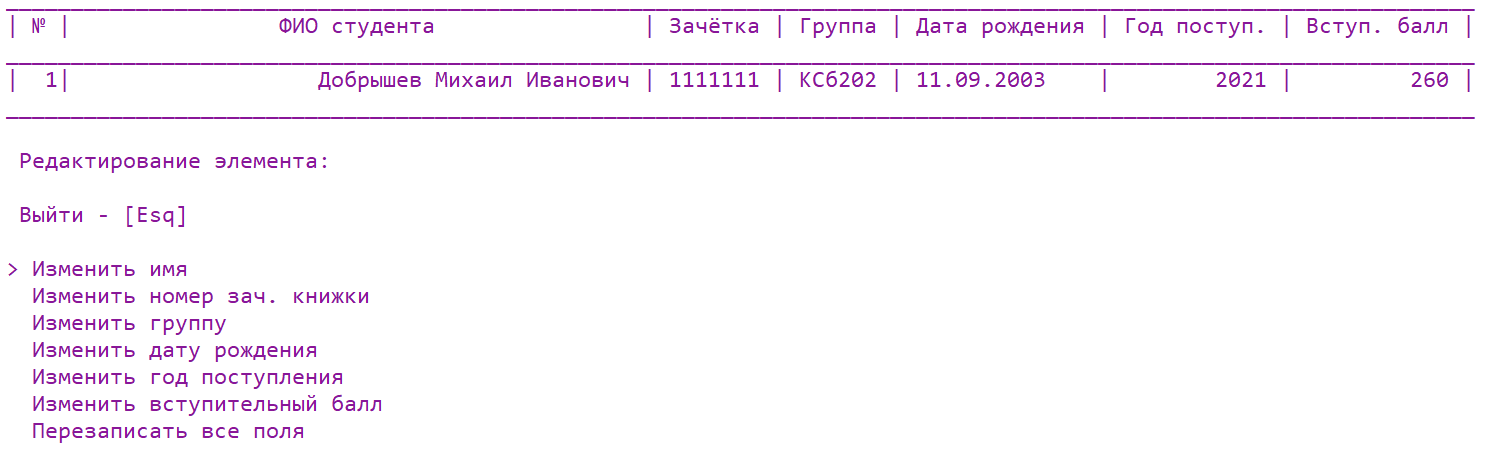


Рисунок 3.9 – Редактирование элемента

При выборе третьей команды пользователю предложат удалить текущий элемент списка (Рисунок 3.10). При удалении элемента указатель на текущий элемент сместиться на один элемент.

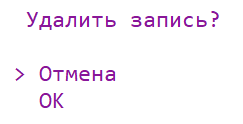


Рисунок 3.10 – Удаление элемента

При выборе четвёртой программа предложит пользователю отсортировать список по возрастанию или убыванию имени (Рисунок 3.11). При выборе одной из сортировок элементы списка будут отсортированы (Рисунок 3.12).

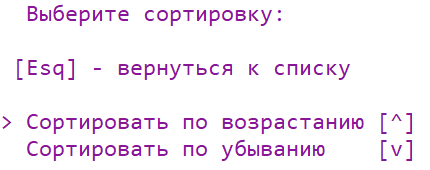


Рисунок 3.11 – Выбор сортировки



Рисунок 3.12 – Результат сортировки списка по убыванию

При выборе пятой команды обработанный список можно будет сохранить в формате txt, dat или bin. При успешном сохранении на экран будет выведено соответствующее сообщение (Рисунок 3.13), иначе будет выведено сообщение об ошибке.

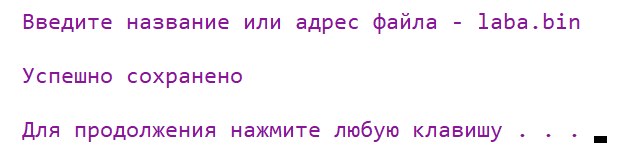


Рисунок 3.13 – Сохранение списка

При выборе шестой команды пользователь сможет ввести текстовый запрос на поиск внутри списка (Рисунок 3.14).



Рисунок 3.14 – Ввод запроса

В результате поиска на экран будет выведен список элементов, соответствующих запросу (Рисунок 3.15). Данный список можно скролить по страницам, а при вводе клавиши Enter программа вернётся в исходный список, но теперь текущем будет элемент, выбранный в списке найденных. При желании можно вернуться обратно нажатием клавиши Escape.

Если же результатов по запросу найдено не будет, программа выведет соответствующее сообщение.

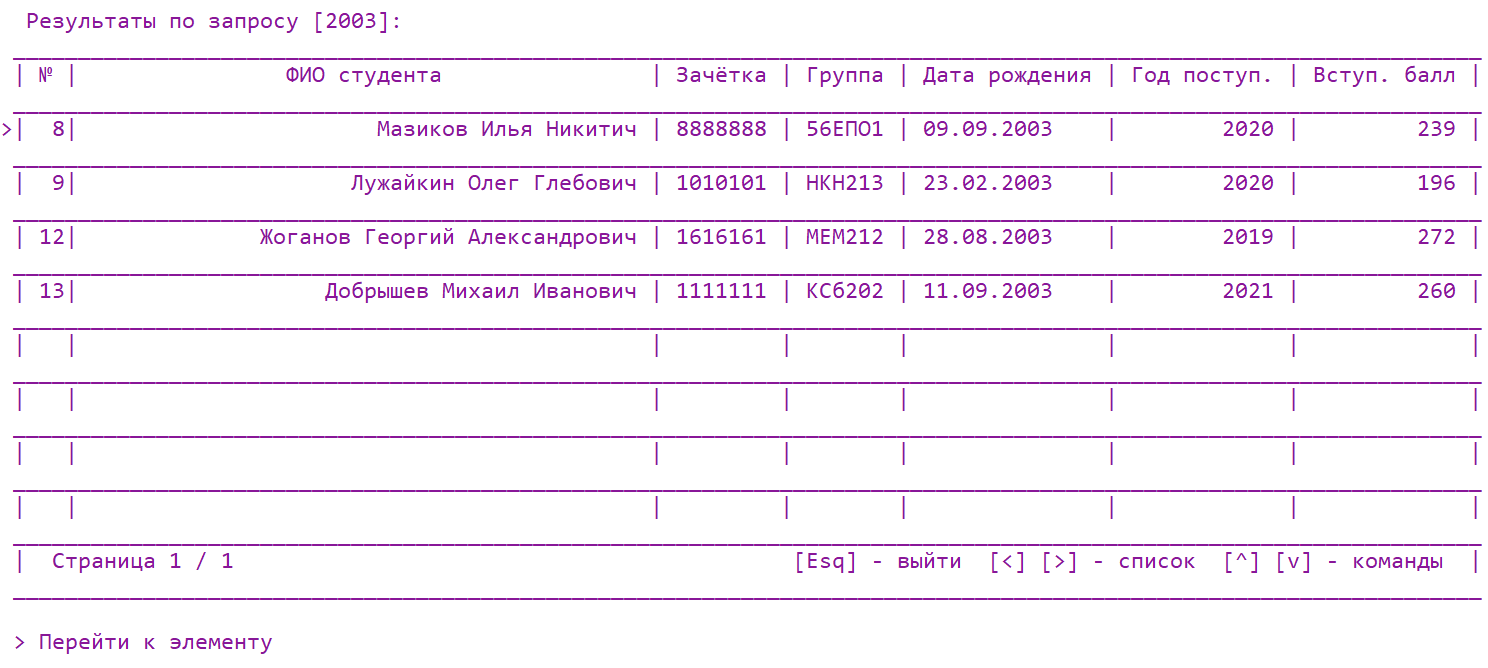


Рисунок 3.15 – Вывод результатов по запросу.

При выборе седьмой команды будет выведен список, в котором хранятся по пятеро студентов за каждый год, обладающих наивысшим вступительным былом (Рисунок 3.16). При скроллинге списка вправо-влево можно изменять год, за который выводятся лучшие студенты.

При нажатии Enter можно сохранить полученный список в файл. Для того, чтобы вернуться в основное меню, необходимо нажать Escape.

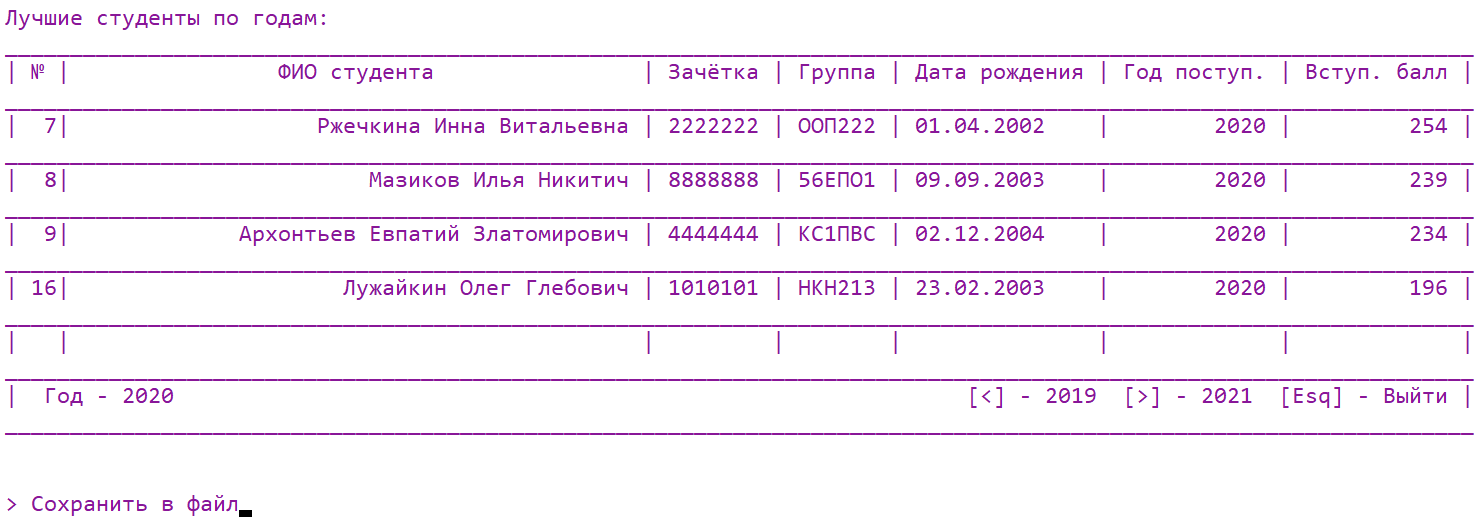


Рисунок 3.16 – Вывод лучших студентов за год

Для выхода из программы необходимо вернуться в главное меню и нажать Escape, либо выбрать команду «Завершить работу».

# Заключение

Данный курсовой проект стал итогом изучения курса «Алгоритмизация и программирования» в рамках образовательной программы института. Одной из целей задания было применить все полученные знания для разработки программы, решающей комплексный ряд задач.

В ходе написания программы были использованы знания о множестве изученных тем: структуры, работа с памятью и указателями, файловый ввод-вывод, различные типы данных, динамические структуры и др.

В качестве структуры данных для хранения информации были выбраны линейные двусвязные списки.

В результате работы была написана программа на языке C++, предоставляющая пользователю ряд опций по обработке данных в соответствии с поставленной задачей, а именно: инициализация списка, добавление элемента в конец, удаление элемента (из любой части списка), сортировка, поиск, редактирование элемента, сохранение в файл и чтение из файла.

Работа в программе реализована через меню-ориентированный интерфейс с подсказками по навигации и управлению.

В ходе тестирования было подтверждено, что программа полностью удовлетворяет поставленным требованиям и может быть использована для учёта данных в образовательных учреждениях, университетах.

# список используемых источников

1. 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.
2. 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.
3. 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.
4. Форум программистов и сисадминов: Киберфорум [Электронный ресурс]. URL: <https://www.cyberforum.ru/cpp-beginners/>
5. Система вопросов и ответов о программировании CodeNet [Электронный ресурс]. URL: <http://forum.codenet.ru/>
6. Cистема вопросов и ответов о программировании StackOverFlow (на русском) [Электронный ресурс]. URL - <https://ru.stackoverflow.com/>
7. Вопросы и ответы по IT-темам: Overcoder [Электронный ресурс]. URL: <https://overcoder.net>
8. Общедоступный веб-API Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/>
9. Библиотека программиста: proglib [Электронный ресурс]. URL: <https://proglib.io/>
10. Сервис по поиску ответов на вопросы программистов Web Answers [Электронный ресурс]. URL: <https://web-answers.ru/>

# Приложение а КОД ПРОГРАММЫ

#include <iostream>

#include <string>

#include <fstream>

#include <iomanip>

#include <conio.h>

#include <cmath>

#include <clocale>

using namespace std;

// КОНСТАНТЫ (клавиши)

const int Enter = 13;

const int Up = 72;

const int Down = 80;

const int Left = 75;

const int Right = 77;

const int Esq = 27;

// КОНСТАНТЫ (таблица)

string tabline = " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

string tabhead = " | № | ФИО студента | Зачётка | Группа | Дата рождения | Год поступ. | Вступ. балл |\n";

string tabempt = " | | | | | | | |\n";

// Структура для дат

struct Date

{

unsigned short day; // число

unsigned short month; // месяц

unsigned short year; // год

};

// Структура ФИО студента

struct Initial

{

string surname; // фамилия

string name; // имя

string patronym; // отчество

};

// Структура записи о студенте

struct Student

{

int num; // номер записи в таблице

unsigned int record\_num; // номер зачётной книжки

Initial FIO; // имя

string group; // шифр группы

Date birth; // дата рождения

unsigned short entry\_year; // год поступления

unsigned short entry\_score; // вступительный балл

};

// Структура для списка

struct List

{

struct Student inf; // инф. поле

struct List\* prev; // пред. элемент

struct List\* next; // след. элемент

};

// Структура списка комманд

struct DirList

{

int num; // номер команды

struct DirList\* prev; // пред команда

struct DirList\* next; // след. команда

};

// Списки с номерами команд

DirList\* choise1 = NULL;

DirList\* choise2 = NULL;

// Флаг сохранённости списка

bool saved = false;

// Прототипы функций

List\* createList(List\* head); // организация списка

Student fillStudentNote(); // заполнение элемента списка

void printList(List\* head, List\* cur); // просмотр списка

List\* printNote(List\* note, List\* cur); // вывод элемента списка

List\* deleteNote(List\* head, List\*cur); // удаление записи

void editNote(List\* cur); // перезапись записи

List\* sortList(List\* head); // сортировка списка

List\* searchList(List\* head, List\*cur); // поиск по списку

void saveToFile(List\* head); // выгрузка в файл

List\* loadFromFile(List\* head); // загрузка из файла

List\* cleanMemory(List\* head); // очищение памяти (таблица)

DirList\* cleanMemoryDir(DirList\* head); // очищение памяти (команды)

List\* tableMenuEsq(List\* head); // выход из основного меню

List\* getLastNote(List\* head); // поиск последнего элемента

void searchBest(List\* head); // вывод лучших студентов по годам

// Заполнение списка команд в соответствии с переданным массивом

void fillCommArr(int comm[], int sizeArr, bool thr)

{

DirList\* choise = new DirList;

choise->num = comm[0];

DirList\* dir = choise;

for (int i = 1; i < sizeArr; i++)

{

DirList\* newDir = new DirList;

newDir->num = comm[i];

newDir->prev = dir;

dir->next = newDir;

dir = dir->next;

}

dir->next = choise;

choise->prev = dir;

thr ? cleanMemoryDir(choise1) : cleanMemoryDir(choise2);

thr ? (choise1 = choise) : (choise2 = choise);

}

// Выбор команды из предложенных вариантов

int selectCommand(int comm[], int sizeArr, bool thr)

{

DirList\* choise = thr ? (choise1) : (choise2);

for (int i = 0; i < sizeArr; i++)

{

if (choise->num == comm[i]) cout << " > ";

else cout << " ";

switch (comm[i])

{

case 1: cout << "Добавить запись \n"; break;

case 2: cout << "Вывести список \n"; break;

case 3: cout << "Удалить запись \n"; break;

case 4: cout << "Сохранить список в файл \n"; break;

case 5: cout << "Загрузить список из файла \n"; break;

case 6: cout << "Создать новый список \n"; break;

case 7: cout << "Продолжить ввод \n"; break;

case 8: cout << "Завершить ввод \n"; break;

case 9: cout << "Сортировать по имени\n"; break;

case 10: cout << "Редактировать запись \n"; break;

case 11: cout << "Отмена \n"; break;

case 12: cout << "ОК \n"; break;

case 13: cout << "Сортировать по возрастанию [^] \n"; break;

case 14: cout << "Сортировать по убыванию [v] \n"; break;

case 15: cout << "Поиск внутри списка \n"; break;

case 16: cout << "Лучшие студенты \n"; break;

case 17: cout << "Выйти без сохранения \n"; break;

case 18: cout << "Изменить имя \n"; break;

case 19: cout << "Изменить номер зач. книжки \n"; break;

case 20: cout << "Изменить группу \n"; break;

case 21: cout << "Изменить дату рождения \n"; break;

case 22: cout << "Изменить год поступления \n"; break;

case 23: cout << "Изменить вступительный балл \n"; break;

case 24: cout << "Перезаписать все поля \n"; break;

case 0: cout << "Завершить работу \n"; break;

}

}

char c = 0;

while ((c != Up) && (c != Down) && (c != Enter) && (c != Left) && (c != Right) && (c != Esq))

c = \_getch();

switch (c)

{

case Up: choise = choise->prev; break;

case Down: choise = choise->next; break;

case Enter: return choise->num; break;

case Left: return Left; break;

case Right: return Right; break;

case Esq: return Esq; break;

}

thr ? (choise1 = choise) : (choise2 = choise);

return - 1;

}

// Меню выбора команды

List\* startMenu(List \*head)

{

int arr[] = { 6, 5, 0 };

fillCommArr(arr, 3, 1);

while (1)

{

system("cls");

cout << " Программа обработки данных о студентах\n";

cout << " Работа Мовенко Константина, ИС/б-21-2-о \n";

cout << " Севастопольский государственный университет, 2022\n";

cout << "--------------------------------------------------------------------\n";

cout << " [^][v] - переключение команд [Esq] - выход из программы \n";

cout << "--------------------------------------------------------------------\n";

cout << "\n Выберите действие\n\n";

switch (selectCommand(arr, 3, 1))

{

case 6: // создание нового списка

head = createList(head);

return head;

break;

case 5: // загрузка списка из файла

head = loadFromFile(head);

if (head) return head;

break;

case Esq: case 0: // завершение программы

head = cleanMemory(head);

choise1 = cleanMemoryDir(choise1);

choise2 = cleanMemoryDir(choise2);

exit(0);

default: break;

}

}

}

// Вывод списка и выбор команды по его обработке

List\* tableMenu(List\* head)

{

List\* cur = head;

int arr[] = { 1, 10, 3, 9, 4, 15, 16 };

fillCommArr(arr, 7, 1);

while (1)

{

system("cls");

printList(head, cur);

switch (selectCommand(arr, 7, 1))

{

case 1: // добавление элемента

head = createList(head);

cur = getLastNote(head);

if (!cur) cur = head; break;

case 10: // перезапись элемента

editNote(cur);

break;

case 3: // удаление элемента

if (cur) cur = deleteNote(head, cur);

if (!cur) head = NULL;

else if (cur->inf.num == 1) head = cur;

break;

case 9: // сортировка по имени

head = sortList(head);

break;

case 4: // сохранение в файл

saveToFile(head);

break;

case 15: // поиск в списке

cur = searchList(head, cur);

break;

case 16: // вывод лучших

searchBest(head);

break;

case 11: // конец работы со списком

case Esq:

head = tableMenuEsq(head);

if (saved) return NULL;

break;

// Перемещение по списку

case Right: if (cur) if (cur->next) cur = cur->next; break;

case Left: if (cur) if (cur->prev) cur = cur->prev; break;

}

}

}

// Выход из основного меню рботы со списком

List\* tableMenuEsq(List\* head)

{

if (!saved)

{

int arr[] = { 11, 17 };

fillCommArr(arr, 2, 0);

while (1) // выбор действия

{

system("cls");

cout << "\n Изменения не сохранены. При выходе из списка данные не сохранятся.";

cout << "\n\n Выйти из списка? \n\n";

int ask = selectCommand(arr, 2, 0);

if (ask == 11) return head;

if (ask == 17) break;

}

}

head = cleanMemory(head);

choise1 = cleanMemoryDir(choise1);

saved = true;

return head;

}

// НАЧАЛО ПРОГРАММЫ

int main() {

system("chcp 1251");

List\* head = NULL;

while (1) {

head = startMenu(head); // начальное меню

head = tableMenu(head); // полный список команд

}

}

// Возвращает указатель на конец списка

List\* getLastNote(List\* head)

{

if (!head) return head;

while (head->next)

head = head->next;

return head;

}

// Ввод полож. целого числа с проверкой

int getInt(string mass, int max)

{

long int x;

string str;

while (1)

{

cout << " " << mass << " - ";

getline(cin, str);

int i = 0;

if (!str.empty())

{

while (i < str.size())

{

if (!isdigit((unsigned char) str[i])) break;

i++;

}

}

if ((str.find(" ") < string::npos) || str.empty() || (i < str.size()))

{

cout << "\n ОШИБКА: Некорректный ввод данных\n\n";

continue;

}

if ((str.size() > to\_string(max).size()) || ((x = stoi(str)) > max))

{

cout << "\n ОШИБКА: Значение превышает " << max << "\n\n";

continue;

}

break;

}

return x;

}

// Ввод строки без пробела

string getStr(string mass, int max)

{

string str;

while (1)

{

cout << " " << mass << " - ";

getline(cin, str);

if ((str.find(" ") >= string::npos) && !str.empty())

if (str.size() <= max) break;

else

cout << "\n ОШИБКА: Чрезмерно длинное имя\n\n";

else

cout << "\n ОШИБКА: Некорректный ввод данных\n\n";

}

return str;

}

// Заполнение информационного поля

Student fillStudentNote()

{

Student stud;

stud.num = 0;

while (1)

{

stud.FIO.surname = getStr("Фамилия", 38);

stud.FIO.name = getStr("Имя", 38);

stud.FIO.patronym = getStr("Отчество", 38);

string f = stud.FIO.surname + " " + stud.FIO.name + " " + stud.FIO.patronym;

if (f.length() <= 42) break;

else cout << "\n ОШИБКА: Чрезмерно длинное имя\n\n";

}

stud.record\_num = getInt("Номер зачётной книжки (7 цифр)", 9999999);

stud.group = getStr("Шифр группы (6 символов)", 6);

stud.birth.day = getInt("Дата рождения (день)", 31);

stud.birth.month = getInt("Месяц рождения", 12);

stud.birth.year = getInt("Год рождения", 9999);

stud.entry\_year = getInt("Год поступления", 9999);

stud.entry\_score = getInt("Вступительный балл", 999);

return stud;

}

// Присоединение элемента к списку

List\* addNote(List\* head, Student info)

{

if (!head) // создание головы списка

{

head = new List;

head->inf = info;

head->inf.num = 1;

head->next = NULL;

head->prev = NULL;

}

else // добавление элемента в конец

{

List\* newNote = new List;

List\* lastNote = getLastNote(head);

newNote->inf = info;

newNote->inf.num = lastNote->inf.num + 1;

lastNote->next = newNote;

newNote->next = NULL;

newNote->prev = lastNote;

}

return head;

}

// Организация списка

List\* createList(List\* head)

{

while (1)

{

system("cls");

cout << "\n Заполнение таблицы \n\n";

Student info = fillStudentNote();

head = addNote(head, info);

saved = false;

int arr[] = { 7, 8 };

fillCommArr(arr, 2, 0);

while (1) // выбор действия

{

system("cls");

cout << "\n Выберите действие: \n\n";

int ask = selectCommand(arr, 2, 0);

if (ask == 8) return head;

if (ask == 7) break;

}

}

}

// Перевод даты в строку

string printDate(Date date)

{

string str;

if (date.day > 9)

str = to\_string(date.day) + '.';

else str = '0' + to\_string(date.day) + '.';

if (date.month > 9)

str += to\_string(date.month) + '.';

else str += '0' + to\_string(date.month) + '.';

str += to\_string(date.year);

return str;

}

// Вывод элемента списка

List\* printNote(List\* note, List\* cur)

{

if (note)

{

cout << ((cur) && (note->inf.num == cur->inf.num) ? (">") : (" ")) << "|" << setw(3) << note->inf.num;

string fullName = note->inf.FIO.surname + " " + note->inf.FIO.name + " " + note->inf.FIO.patronym;

cout << "|" << setw(43) << fullName;

cout << " |" << setw(8) << note->inf.record\_num;

cout << " |" << setw(7) << note->inf.group << " | ";

cout << setw(14) << left << printDate(note->inf.birth);

cout << "|" << right << setw(12) << note->inf.entry\_year;

cout << " |" << setw(12) << note->inf.entry\_score << " |\n";

cout << tabline;

return note->next;

}

else

cout << tabempt << tabline;

return note;

}

// Поиск позиции (не индекса) элемента в списке

int findNotePos(List\* head, List\* cur)

{

List\* note = head;

int pos = 1;

while ((note) && (note->inf.num != cur->inf.num))

{

pos++;

note = note->next;

}

return pos;

}

// Вывод списка

void printList(List\* head, List\* cur)

{

int pos = findNotePos(head, cur); // выбор страницы списка

List\* printHead = head;

for (int i = 1; i < ceil((pos / 8.0)); i++)

{

printHead = printHead->next->next->next->next->next->next->next->next;

}

cout << tabline << tabhead << tabline; // вывод списка в виде таблицы

List\* note = printHead;

for (int i = 1; i <= 8; i++)

{

note = printNote(note, cur);

}

List\* lastNote = getLastNote(head);

int lNum = findNotePos(head, lastNote);

string page = "Страница " + to\_string((int)(ceil(pos / 8.0)));

page += " / " + to\_string((int)(ceil(lNum / 8.0)));

cout << " | " << setw(18) << left << page << right << setw(93) << "[Esq] - выйти [<] [>] - список [^] [v] - команды |\n";

cout << tabline << endl;

}

// Удаление элемента

List\* deleteNote(List\* head, List\* del)

{

int arr[] = { 11, 12 };

fillCommArr(arr, 2, 0);

int comCode = -1;

while (comCode < 0)

{

system("cls");

cout << "\n Удалить запись? \n\n";

switch (comCode = selectCommand(arr, 2, 0))

{

case 11: return del;

case 12: break;

}

}

saved = false;

if (del->next) del->next->prev = del->prev;

if (del->prev) del->prev->next = del->next;

List\* newcur = del->next;

while (newcur)

{

newcur->inf.num--;

newcur = newcur->next;

}

if (del->prev) newcur = del->prev;

else newcur = del->next;

delete del;

return newcur;

}

// Перезапись элемента

void rewriteNote(List\* cur)

{

system("cls");

cout << "\n Перезапись элемента: \n\n";

if (cur)

{

cur->inf = fillStudentNote();

if (cur->prev) cur->inf.num = cur->prev->inf.num + 1;

else cur->inf.num = 1;

saved = false;

}

return;

}

// Редактирование элемента списка

void editNote(List\* cur)

{

if (!cur) return;

int arr[] = { 18, 19, 20, 21, 22, 23, 24 };

fillCommArr(arr, 7, 0);

while (1)

{

system("cls");

cout << tabline << tabhead << tabline;

printNote(cur, NULL);

cout << "\n Редактирование элемента: \n";

cout << "\n Выйти - [Esq] \n\n";

int com = selectCommand(arr, 7, 0);

cout << endl;

switch (com)

{

case 18:

while (1)

{

cur->inf.FIO.surname = getStr("Фамилия", 38);

cur->inf.FIO.name = getStr("Имя", 38);

cur->inf.FIO.patronym = getStr("Отчество", 38);

string f = cur->inf.FIO.surname + " " + cur->inf.FIO.name + " " + cur->inf.FIO.patronym;

saved = false;

if (f.length() <= 42) break;

else cout << "\n ОШИБКА: Чрезмерно длинное имя \n\n";

}

break;

case 19:

cur->inf.record\_num = getInt("Номер зачётной книжки (7 цифр)", 9999999);

saved = false;

break;

case 20:

cur->inf.group = getStr("Шифр группы (6 символов)", 6);

saved = false;

break;

case 21:

cur->inf.birth.day = getInt("Дата рождения (день)", 31);

cur->inf.birth.month = getInt("Месяц рождения", 12);

cur->inf.birth.year = getInt("Год рождения", 9999);

saved = false;

break;

case 22:

cur->inf.entry\_year = getInt("Год поступления", 9999);

saved = false;

break;

case 23:

cur->inf.entry\_score = getInt("Вступительный балл", 999);

saved = false;

break;

case 24:

rewriteNote(cur);

break;

case Esq: return;

}

}

}

// Сравнение по ФИО

int initcmp(Initial n1, Initial n2)

{

string name1 = n1.surname + " " + n1.name + " " + n1.patronym;

string name2 = n2.surname + " " + n2.name + " " + n2.patronym;

if (name1 > name2) return -1;

if (name1 < name2) return 1;

return 0;

}

// Сортировка списка по возрастанию

List\* sortListAscend(List\* head)

{

List\* il = getLastNote(head);

while (il->prev)

{

List\* jl = il->prev;

while (jl)

{

if (initcmp(jl->inf.FIO, il->inf.FIO) < 0)

{

Student swap = il->inf;

il->inf = jl->inf;

jl->inf = swap;

jl->inf.num = il->inf.num;

il->inf.num = swap.num;

}

jl = jl->prev;

}

il = il->prev;

}

return head;

}

// Сорировка списка по убыванию

List\* sortListDescend(List\* head)

{

List\* il = getLastNote(head);

while (il->prev)

{

List\* jl = il->prev;

while (jl)

{

if (initcmp(jl->inf.FIO, il->inf.FIO) > 0)

{

Student swap = il->inf;

il->inf = jl->inf;

jl->inf = swap;

jl->inf.num = il->inf.num;

il->inf.num = swap.num;

}

jl = jl->prev;

}

il = il->prev;

}

return head;

}

// Сортировка списка

List\* sortList(List\* head)

{

if ((!head) || (!head->next)) return head;

int arr[] = { 13, 14 };

fillCommArr(arr, 2, 0);

while (1)

{

system("cls");

cout << "\n Выберите сортировку: \n\n";

cout << " [Esq] - вернуться к списку\n\n";

int ask = selectCommand(arr, 2, 0);

if (ask == 13)

{

head = sortListAscend(head);

saved = false;

return head;

}

if (ask == 14)

{

head = sortListDescend(head);

saved = false;

return head;

}

if (ask == Esq) return head;

}

}

// Проверка элемента на соответствие поиску

bool isSearched(List\* el, string srch)

{

if (!el) return false;

string s;

s = to\_string(el->inf.num); // по инексу

if (s.find(srch) < string::npos) return true;

s = el->inf.FIO.surname + " " + el->inf.FIO.name; // по ФИО

s += " " + el->inf.FIO.patronym;

if (s.find(srch) < string::npos) return true;

s = el->inf.group; // по группе

if (s.find(srch) < string::npos) return true;

s = to\_string(el->inf.record\_num); // по зачётке

if (s.find(srch) < string::npos) return true;

s = printDate(el->inf.birth); // по дате

if (s.find(srch) < string::npos) return true;

s = to\_string(el->inf.entry\_score); // по баллу

if (s.find(srch) < string::npos) return true;

s = to\_string(el->inf.entry\_year); // по году п.

if (s.find(srch) < string::npos) return true;

return false;

}

// Поиск внутри списка

List\* searchList(List\* head, List\* cur)

{

system("cls");

string srch;

if (!head) return head;

while (1) // ввод запроса

{

cout << "\n Что искать - ";

getline(cin, srch);

if (!srch.empty()) break;

else cout << endl;

}

List\* shead = NULL; // заполнение списка результатов

List\* run = head;

while (run)

{

if (isSearched(run, srch))

{

shead = addNote(shead, run->inf);

List\* lastNote = getLastNote(shead);

lastNote->inf.num = run->inf.num;

}

run = run->next;

}

if (!shead) // результатов не найдено

{

cout << "\n Не найдено результатов по запросу \n\n";

system("pause");

return cur;

}

run = shead; // вывод результатов

while (1)

{

system("cls");

cout << "\n Результаты по запросу [" << srch << "]: \n";

printList(shead, run);

cout << " > Перейти к элементу \n";

char c = 0;

while ((c != Right) && (c != Left) && (c != Esq) && (c != Enter))

c = \_getch();

if ((c == Right) && run) if (run->next) run = run->next;

if ((c == Left) && run) if (run->prev) run = run->prev;

if (c == Esq) return cur;

if (c == Enter) break;

}

cur = head;

while (cur->inf.num != run->inf.num)

cur = cur->next;

shead = cleanMemory(shead);

return cur;

}

// Сортировка целочисленного массива по возрастанию

void sortInt(int arr[], int len)

{

for (int i = 1; i < len; i++)

for (int j = 0; j < (len - i); j++)

{

if (arr[j] > arr[j + 1])

{

int swap = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = swap;

}

}

return;

}

// Заполнение массива годов поступления

int fillEntryYearsArr(int years[], List\* head)

{

int len = 0;

List\* run = head;

while (run)

{

bool hmm = true;

for (int i = 0; i < len; i++)

if (years[i] == run->inf.entry\_year)

hmm = false;

if (hmm)

{

years[len] = run->inf.entry\_year;

len++;

}

run = run->next;

}

return len;

}

// Сортировка списка по убыванию оценки

List\* sortBestList(List\* head)

{

if ((!head) || (!head->next)) return head;

List\* il = getLastNote(head);

while (il->prev)

{

List\* jl = il->prev;

while (jl)

{

if (jl->inf.entry\_score < il->inf.entry\_score)

{

Student swap = il->inf;

il->inf = jl->inf;

jl->inf = swap;

jl->inf.num = il->inf.num;

il->inf.num = swap.num;

}

jl = jl->prev;

}

il = il->prev;

}

return head;

}

// Заполнить список 5 лучших студентов за год

List\* bestForYear(List\* head, int year)

{

List\* shead = NULL;

List\* run = head;

int count = 0;

while (run)

{

if (run->inf.entry\_year == year)

{

shead = addNote(shead, run->inf);

List\* lastNote = getLastNote(shead);

lastNote->inf.num = run->inf.num;

count++;

}

run = run->next;

}

shead = sortBestList(shead);

if (count > 5)

{

List\* last = shead->next->next->next->next;

List\* del = last->next;

last->next = NULL;

del = cleanMemory(del);

}

return shead;

}

// Вывод списка лучших

void printBestList(List\* shead, int y, int years[], int len)

{

cout << tabline << tabhead << tabline;

for (int i = 0; i < 5; i++)

shead = printNote(shead, NULL);

string str1 = "Год - " + to\_string(years[y]);

string str2 = "[Esq] - Выйти";

if (y < (len - 1))

str2 = "[>] - " + to\_string(years[y + 1]) + " " + str2;

if (y > 0)

str2 = "[<] - " + to\_string(years[y - 1]) + " " + str2;

cout << " | " << setw(18) << left << str1 << right << setw(90) << str2 << " | \n";

cout << tabline << endl;

return;

}

// Сохранение лучших в файл

void saveBestList(List\* head, int years[], int len)

{

List\* full = bestForYear(head, years[0]);

for (int i = 1; i < len; i++)

{

List\* shead = bestForYear(head, years[i]);

List\* last = getLastNote(full);

last->next = shead;

shead->prev = last;

}

saveToFile(full);

return;

}

// Поиск лучших студентов по годам

void searchBest(List\* head)

{

if (!head) return;

int years[1000];

int len = fillEntryYearsArr(years, head);

sortInt(years, len);

int y = 0;

while (1)

{

system("cls");

cout << "\n Лучшие студенты по годам: \n";

List\* shead = bestForYear(head, years[y]);

printBestList(shead, y, years, len);

cout << "\n > Сохранить в файл";

shead = cleanMemory(shead);

char c = 0;

while ((c != Right) && (c != Left) && (c != Esq) && (c != Enter))

c = \_getch();

if (c == Right) if (y < (len - 1)) y++;

if (c == Left) if (y > 0) y--;

if (c == Enter) saveBestList(head, years, len);

if (c == Esq) break;

}

return;

}

// Сохранение списка в текстовый файл

void saveToFileTxt(List\* head, string fileName)

{

ofstream fout(fileName, ios::out); // открытие потока

if (!fout)

{

cout << "\n Ошибка открытия файла \n\n";

system("pause");

return;

}

while (head)

{

fout << head->inf.FIO.surname << " ";

fout << head->inf.FIO.name << " ";

fout << head->inf.FIO.patronym << " ";

fout << head->inf.record\_num << " ";

fout << head->inf.group << " ";

fout << head->inf.birth.day << " ";

fout << head->inf.birth.month << " ";

fout << head->inf.birth.year << " ";

fout << head->inf.entry\_year << " ";

fout << head->inf.entry\_score << endl;

head = head->next;

}

fout.close();

saved = true;

cout << "\n Успешно сохранено \n\n ";

return;

}

// Сохранение списка в типизированный файл

void saveToFileDat(List\* head, string fileName)

{

ofstream f(fileName, ios::binary | ios::out); // открытие потока

if (!f)

{

cout << "\n Ошибка открытия файла \n\n";

return;

}

while (head)

{

size\_t len;

len = head->inf.FIO.surname.length() + 1;

f.write((char\*)&len, sizeof(len));

f.write((char\*)head->inf.FIO.surname.c\_str(), len);

len = head->inf.FIO.name.length() + 1;

f.write((char\*)&len, sizeof(len));

f.write((char\*)head->inf.FIO.name.c\_str(), len);

len = head->inf.FIO.patronym.length() + 1;

f.write((char\*)&len, sizeof(len));

f.write((char\*)head->inf.FIO.patronym.c\_str(), len);

f.write((char\*)&head->inf.record\_num, sizeof(head->inf.record\_num));

len = head->inf.group.length() + 1;

f.write((char\*)&len, sizeof(len));

f.write((char\*)head->inf.group.c\_str(), len);

f.write((char\*)&head->inf.birth.day, sizeof(head->inf.birth.day));

f.write((char\*)&head->inf.birth.month, sizeof(head->inf.birth.month));

f.write((char\*)&head->inf.birth.year, sizeof(head->inf.birth.year));

f.write((char\*)&head->inf.entry\_year, sizeof(head->inf.entry\_year));

f.write((char\*)&head->inf.entry\_score, sizeof(head->inf.entry\_score));

head = head->next;

}

f.close();

saved = true;

cout << "\n Успешно сохранено \n\n ";

return;

}

// Сохранение списка в файл

void saveToFile(List\* head)

{

system("cls");

string fileName;

while (1) // чтение имени файла

{

system("cls");

cout << "\n Введите название или адрес файла - ";

getline(cin, fileName);

if (fileName.empty()) continue;

if (fileName.size() < 5)

{

cout << "\n Ошибка: не достигнута минимальная длина имени файла \n\n ";

system("pause");

}

else break;

}

string type = fileName.substr(fileName.size() - 3);

if (type == "txt") // формат файла

saveToFileTxt(head, fileName);

else if ((type == "dat") || (type == "bin"))

saveToFileDat(head, fileName);

else

{

cout << "\n Ошибка - не поддерживаемый формат файла";

cout << "\n Поддерживаемые форматы - txt, dat, bin\n\n ";

system("pause");

}

system("pause");

return;

}

// Чтение из текстового файла элемента списка

List\* loadElementFromTxt(ifstream &f, List\* t, List\* l)

{

t->inf.num = (l) ? (l->inf.num + 1) : 1;

if (!f.eof()) f >> t->inf.FIO.surname; else return NULL;

if (!f.eof()) f >> t->inf.FIO.name; else return NULL;

if (!f.eof()) f >> t->inf.FIO.patronym; else return NULL;

if (!f.eof()) f >> t->inf.record\_num; else return NULL;

if (!f.eof()) f >> t->inf.group; else return NULL;

if (!f.eof()) f >> t->inf.birth.day; else return NULL;

if (!f.eof()) f >> t->inf.birth.month; else return NULL;

if (!f.eof()) f >> t->inf.birth.year; else return NULL;

if (!f.eof()) f >> t->inf.entry\_year; else return NULL;

if (!f.eof()) f >> t->inf.entry\_score; else return NULL;

return t;

}

// Загрузка из текстового файла

List\* loadFromFileTxt(string fileName)

{

ifstream fin(fileName, ios::in); // открытие потока

if (!fin)

{

cout << "\n Ошибка открытия файла \n\n";

system("pause");

return NULL;

}

List\* head = NULL; // чтение списка

List\* temp;

while (1)

{

temp = new List;

temp = loadElementFromTxt(fin, temp, getLastNote(head));

if (!temp)

{

delete temp;

fin.close();

if (!head) // файл пуст

{

cout << "\n Ошибка: файл пустой \n\n";

system("pause");

}

else saved = true;

return head;

}

head = addNote(head, temp->inf);

delete temp;

}

}

// Чтение из файла элемента списка

List\* loadElementFromBin(ifstream& f, List\* t, List\* l)

{

if (f.eof()) return NULL;

t->inf.num = (l) ? (l->inf.num + 1) : 1;

try

{

size\_t len;

char\* buf;

f.read((char\*)&len, sizeof(len));

buf = new char[len];

f.read(buf, len);

t->inf.FIO.surname = buf;

delete[] buf;

buf = NULL;

f.read((char\*)&len, sizeof(len));

buf = new char[len];

f.read(buf, len);

t->inf.FIO.name = buf;

delete[] buf;

buf = NULL;

f.read((char\*)&len, sizeof(len));

buf = new char[len];

f.read(buf, len);

t->inf.FIO.patronym = buf;

delete[] buf;

buf = NULL;

f.read((char\*)&t->inf.record\_num, sizeof(t->inf.record\_num));

f.read((char\*)&len, sizeof(len));

buf = new char[len];

f.read(buf, len);

t->inf.group = buf;

delete[] buf;

buf = NULL;

f.read((char\*)&t->inf.birth.day, sizeof(t->inf.birth.day));

f.read((char\*)&t->inf.birth.month, sizeof(t->inf.birth.month));

f.read((char\*)&t->inf.birth.year, sizeof(t->inf.birth.year));

f.read((char\*)&t->inf.entry\_year, sizeof(t->inf.entry\_year));

f.read((char\*)&t->inf.entry\_score, sizeof(t->inf.entry\_score));

}

catch (...)

{

return NULL;

}

return t;

}

// Загрузка из бинарного файла

List\* loadFromFileBin(string fileName)

{

ifstream fin(fileName, ios::binary | ios::in);

if (!fin)

{

cout << "\n Ошибка открытия файла \n\n ";

system("pause");

return NULL;

}

List\* head = NULL; // чтение списка

List\* temp;

while (1)

{

temp = new List;

temp = loadElementFromBin(fin, temp, getLastNote(head));

if (!temp)

{

fin.close();

if (!head) // файл пуст

{

cout << "\n Ошибка: файл пустой \n\n ";

system("pause");

}

else saved = true;

return head;

}

head = addNote(head, temp->inf);

delete temp;

}

}

// Загрузка списка из файла

List\* loadFromFile(List\* head)

{

head = cleanMemory(head);

string fileName;

while (1) // чтение имени файла

{

system("cls");

cout << "\n Введите название или адрес файла - ";

getline(cin, fileName);

if (fileName.empty()) continue;

if (fileName.size() < 5)

{

cout << "\n Ошибка: не достигнута минимальная длина имени файла \n\n ";

system("pause");

}

else break;

}

string type = fileName.substr(fileName.size() - 3);

if (type == "txt") // формат файла

head = loadFromFileTxt(fileName);

else if ((type == "dat") || (type == "bin"))

head = loadFromFileBin(fileName);

else

{

cout << "\n Ошибка - не поддерживаемый формат файла";

cout << "\n Поддерживаемые форматы - txt, dat, bin\n\n ";

system("pause");

}

return head;

}

// Очищение памяти главного списка

List\* cleanMemory(List\* head)

{

List\* currNote;

while (head)

{

currNote = head;

head = head->next;

delete currNote;

}

return head;

}

// Очищение памяти списка комманд

DirList\* cleanMemoryDir(DirList\* head)

{

DirList\* currNote;

if (head && head->prev)

head->prev->next = NULL;

while (head)

{

currNote = head;

head = head->next;

delete currNote;

}

return head;

}