|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Институт информационных технологий | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(название института полностью)* | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Кафедра/департамент | | | | | | «Информационные системы» | | | | | | | | | | | | | | | | | | | | | | |
| *(наименование кафедры/департамента полностью)*  09.03.04 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(код и наименование направления подготовки/специальности)*  Программная инженерия | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(наименование профиля/специальности)* | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **КУРСОВАЯ РАБОТА / КУРСОВОЙ ПРОЕКТ** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| по дисциплине | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Объектно-ориентированное программирование** | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| *(наименование дисциплины)* | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| на тему | | | | **Обработка табличной информации (вариант № 5)** | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Выполнил: обучающийся | | | | | | | | | | | | | | |
| группы: | | | | | | ИТ/б-22-8-о | | | | | | | | |
|  | | | | | | | | | | | | | | |
| Д.Р. Зиберт | | | | | | | | | | | | | | |
| *(инициалы, фамилия)* | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | |
| « |  | | » | |  | | | 20 | | 23 | | г. | | | |
|  | | | | | | | | | | | | | | | |
| Научный руководитель: | | | | | | | | | | | | | | |
| Тимофеев Илья Сергеевич | | | | | | | | | | | | | | |
| *(инициалы, фамилия)* | | | | | | | | | | | | | | |
| « |  | | | » | | |  | | 20 | | 23 | | г. | | | | |
|  | | | | | | | | | | | | | | | |
| Оценка | | | |  | | | | | |  | | | | | | | | | | | | | | | |
|  | | | | | | | | | |  | | | | | | | | | | | | | | | |
| « |  | » | |  | | 20 | 23 | г. |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Севастополь | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | 23 | | |  | | | | | | | | | | | |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

|  |
| --- |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** |

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Институт информационных технологий | | | | | | | |
| *(название института полностью)* | | | | | | |
| Кафедра/ департамент | | | «Информационные системы» | | | | | | |
| *(наименование кафедры/департамента полностью)* | | | | | | | | | |
|  | | | | | | | | | |
| *(код и наименование направления подготовки/специальности)* | | | | | | | | | |
| *(наименование профиля/специальности)* | | | | | | | | | |
|  | | | | | | | | | |
| Курс | 2 | Группа | | ИТ/б-22-8-о | Семестр | третий | | |

## **З А Д А Н И Е**

**НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| обучающегося | Зиберта Даниила Руслановича | | | | | | |
| *(фамилия, имя, отчество)* | | | | | | | |
| 1. Тема работы (проекта) | | | Обработка табличной информации | | | | |
| вариант 5 | | | | | | | |
|  | | | | | | | |
|  | | | | | | | |
|  | | | | | | | |
| 2. Срок сдачи обучающимся законченного проекта (работы) | | | | |  | | |
| 3. Исходные/входные данные к проекту (работе) | | | |  | | | |
| 1. Язык программирования — С/ С++ | | | | | | | |
| 2. Операционная система — Windows | | | | | | | |
| 3. Среда программирования — Dev-C++ | | | | | | | |
| 4. Класс вычислительной машины — персональная настольная ЭВМ | | | | | | | |
| 5. Меню-ориентированный интерфейс | | | | | | | |
| 6. Исходные данные — файл с записями о проданных товарах | | | | | | | |
| 7. Выходные данные — файл с записями о проданных товарах, сумма выручки | | | | | | | |
|  | | | | | | | |
| 4. Содержание пояснительной записки (перечень вопросов, подлежащих разработке) | | | | | | |  |
| техническое задание, аннотация, содержание, введение, назначение и область применения | | | | | | | |
| программы, технические характеристики программы, выполнение программы, выводы, | | | | | | | |
| перечень ссылок, приложение (текст программы). | | | | | | | |
|  | | | | | | | |
| 5. Перечень графического материала (с точным указанием обязательных чертежей) | | | | | |  | |
| презентация | | | | | | | |
| 6. Дата выдачи задания | | 01.09.2023 | | | | | |

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Название этапов проекта (работы) | Срок выполнения этапов проекта (работы) | Примечание |
| **1** | Постановка задачи | 01.09.23 — 08.09.23 | 2-ая неделя |
| **2** | Функции организации и просмотра списка 0 | 01.09.23 — 08.09.23 | 2-ая неделя |
| **3** | Функции добавления и удаления | 08.09.23 — 22.09.23 | 2-4 неделя |
| **4** | Функции корректировки и сортировки | 22.09.23 — 6.10.23 | 4-6 неделя |
| **5** | Функции сохранения списка (в текстовый и типизированный файл) | 06.10.23 — 20.10.23 | 6-8 неделя |
| **6** | Функции чтения списка (из текстового и типизированного файлов) | 20.10.23 — 3.11.23 | 8-10 неделя |
| **7** | Функции поиска и обработки | 03.11.23 — 17.11.23 | 10-12 неделя |
| **8** | Разработка интерфейса | 17.11.23 — 1.12.23 | 12-14 неделя |
| **9** | Тестирование и отладка | 03.11.23 — 1.12.23 | 10-14 неделя |
| **10** | Разработка программных документов (ПЗ и презентации) | 1.12.23 — 15.12.23 | 14-16 неделя |
| **11** | Защита | 15.12.23 — 25.12.23 | 16-17 неделя |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся |  |  |  |
|  | *(фамилия и инициалы)* |  | *(подпись)* |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель курсового проекта (работы) | | | | | | | | | | | |
|  | | | | | | | | | |  |  |
|  | | | | | | *(фамилия, инициалы, должность)* | | | |  | *(подпись)* |
| « |  | » |  | 20 | | 23 | г. |

**АННОТАЦИЯ**

Настоящий документ представляет собой пояснительную записку  
курсового проекта на тему «Обработка табличной информации». В состав пояснительной записки входят следующие разделы. Сначала делается постановка задачи, описываются основные теоретические положения. В первом разделе описывается проектирование модели базы данных. Во втором разделе описан процесс разработки базы данных. В третьем разделе описано справочное руководство пользователя и руководство оператора. В приложении помещены схемы базы данных, они выполнены в соответствии с требованиями ГОСТ 2.701-2008 и ГОСТ 2.743-91. Документ составлен так, чтобы его можно было использовать как часть пояснительной записки курсовой работы.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ............................................................................................................6

1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ…………….7

2. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ…………………...8

2.1 Постановка задачи на разработку программы……………………………..8

2.2 Описание и обоснование выбора метода организации входных и выходных данных………………………………….…………………………………...9

2.3 Обоснование выбора языка и среды программирования.………………..10

2.4 Описание алгоритмов функционирования программы……………… ….11

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ……………………………………………56

3.1 Условия выполнения программы………………………………….56

3.2 Загрузка и запуск программы………………………………………57

ЗАКЛЮЧЕНИЕ………………………………………………………………...65

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ……….………………….66

ПРИЛОЖЕНИЕ A………………………………………………………………67

**ВВЕДЕНИЕ**

Целью данного курсовой работы является разработка программы учёта движения и наличия материальных ценностей на заводах, которая обеспечивает получение оперативных данных о состоянии складов во множестве заводов и подконтрольных им филиалов.

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 (кодирование) программных модулей на алгоритмическом языке;

- разработка тестовых примеров;

- тестирование и отладка программы;

- разработка программных документов в соответствии с действующими стандартами.

В последующих разделах работы будут подробно рассмотрены основные этапы разработки программы, ее функциональные характеристики, а также результаты обработки данных.

**1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ**

Данная программа предназначена для хранения данных о наличии и движении материальных ценностей на множестве заводов. Применяться она может для складского учёта и в бухгалтерских целях в разрезе множества филиалов, относящихся к разным заводам. С её помощью можно производить подсчёт количества филиалов заводов, схожих в некоторых значениях, определять ответственного за материальные ценности, можно расширять список филиалов, сохранять эти данные в текстовые файлы и точно так же создавать таблицу с помощью созданных пользователем файлов.

Программа спроектирована с учетом простоты использования и минимального времени обучения пользователя. Она обладает структурированным интерфейсом, построенным на основе меню, что делает ее доступной для широкого круга пользователей без специфических навыков программирования. Таким образом, разработанное программное обеспечение предназначено для упрощения и улучшения процессов учета пропусков студентов, а также предоставления администрации и преподавателям необходимой информации для анализа и принятия обоснованных управленческих решений в организации образовательного процесса.

**2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ**

**2.1 Постановка задачи на разработку программы**

Вариант задания задаёт сформировать структуру записи данных о каждом филиале на разных заводах. Запись будет содержать:номер завода, номер филиала, фамилия ответственного за материальные ценности, наличие материальных ценностей на начало периода, получено материальных ценностей на сумму, выбыло на сумму.

Основной задачей является получение ведомостей движения материальных ценностей за отчетный период, содержащую: стоимость материальных ценностей по каждому филиалу на конец отчетного периода, по всему заводу на конец периода, а также итоговые цифры по каждому виду и по всему заводу в целом.

Исходя из выдвинутых вариантом требований, программа должна содержать следующие функции:

а) Создание записи в списке;

б) Вывод всех записей списка на экран в удобном для пользователя виде;

в) Удаление записи из списка;

г) Корректировка записи в списке;

д) Поиск записей по заданному ключу;

е) Сортировка списка в выбранными пользователем направлении и ключу;

ж) Создание списка на основании предоставленным пользователем файла (текстовом и типизированном);

з) Сохранение списка на указанный пользователем файл (текстовый и типизированный);

и) Получение ведомостей в соответствии с вариантом

к) Освобождение памяти, занимаемой списком и завершение работы программы.

**2.2 Описание и обоснование выбора метода организации входных, выходных и промежуточных данных**

В качестве структуры для организации базы данных для курсовой работы был выбран двунаправленный список. Выбор обусловлен следующими преимуществами:

- Изменить двусвязный список очень просто.

- Он может легко выделять или перераспределять память во время своего выполнения.

- Как и в случае с односвязным списком, это самая простая в реализации структура данных.

- Обход этого двусвязного списка возможен в двунаправленной реализации, что невозможно в односвязном списке.

- Удалять узлы проще по сравнению с односвязным списком. Для удаления односвязного списка требуется указатель на узел и предыдущий узел, подлежащий удалению, но в двусвязном списке требуется только указатель, который подлежит удалению.

- Двусвязные списки имеют низкие накладные расходы по сравнению с другими структурами данных, такими как массивы.

Входными данными в программе является информационное поле записи, описание которого имеет вид:

struct my\_struct{

int number\_of\_factory; // номер завода

int number\_of\_branch; // номер филиала

char surname\_of\_accountable[15]; // фамилия Ответственного

float availability\_of\_values; // наличие ценностей

float amount\_of\_received; // прибыло на сумму

float amount\_of\_retired; // выбыло на сумму

};

Первые два поля имеют тип целого числа int, потому что подразумевают номера завода и филиала соответственно, которые могут быть только трёхзначными.

Поле фамилии ответственного является С-строкой – массивом символов, потому что в нём нужно хранить конкретное слово.

Остальные поля имеют типы float, подразумевая, что целая часть означает тысяча, а десятичная - сотни, десятки и единицы.

И сама структура, используемая для связки в списке:

struct my\_list{

struct my\_struct data; // информационное поле

struct my\_list \*prev, // указатели на предыдущий и следующий

\*next; // записи соответственно

};

**2.3 Обоснование выбора языка и среды программирования**

Для данной курсовой работы был выбран язык программирования С, поскольку именно на этом языке зачастую реализуется метод структурного программирования, а также именно этому языку была посвящена работа студентов на первом курсе обучения в учебном заведении. Средой разработки выбрана программа DevC++, предоставляющая возможности программирования на языках С и С++, и обусловлен выбор тем, что она проста в установке и хорошо подходит для использования начинающим программистам.

**2.4 Описание алгоритмов функционирования программы**

2.4.1)Алгоритм функции main приведён на рисунках 2.1.-2.2. Поблочное описание функции main: блок 1

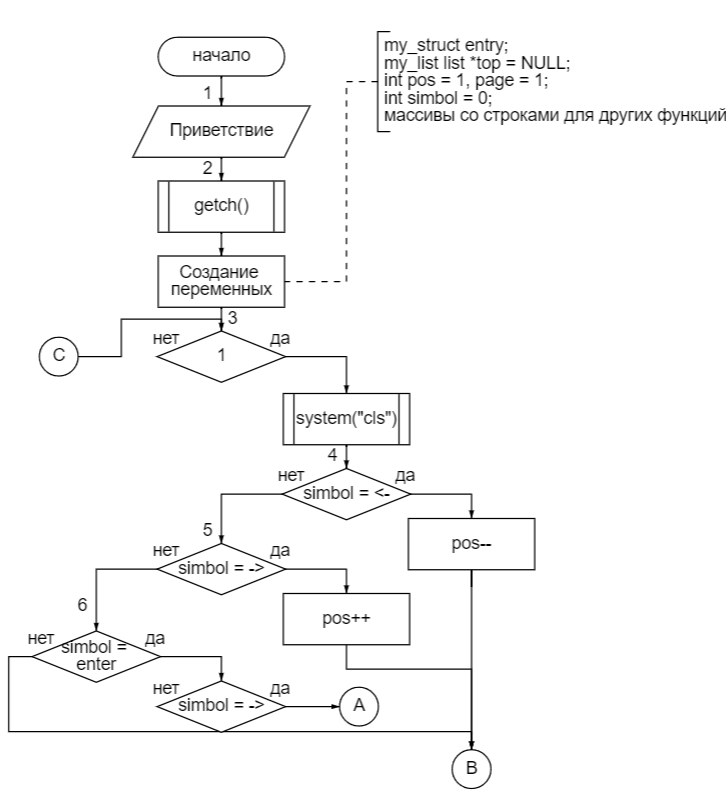


Рисунок 2.1– Схема главной функции main ветка начало

Итак, работа главной функции main начинается с приветствия. Далее в (блок 2) после нажатия любой клавиши создаются необходимые для дальнейшей работы переменные (структура для заполнения её пользователем (entry) при создании записей, указатель на список (top), позиция текущего выбора (pos), переменная для сохранения нажатой клавиши (simbol) и перечень массивов строк для их вывода в различных функциях). После подготовки переменных программа входит в (блок 3) - бесконечный цикл while – в нём мы и будем находиться до конца работы с программой. Происходит очистка экрана от предыдущего сообщения, и с помощью функции print\_menu (см. п. 2.4.20) выводится список доступных для пользователя действий (блок 8).

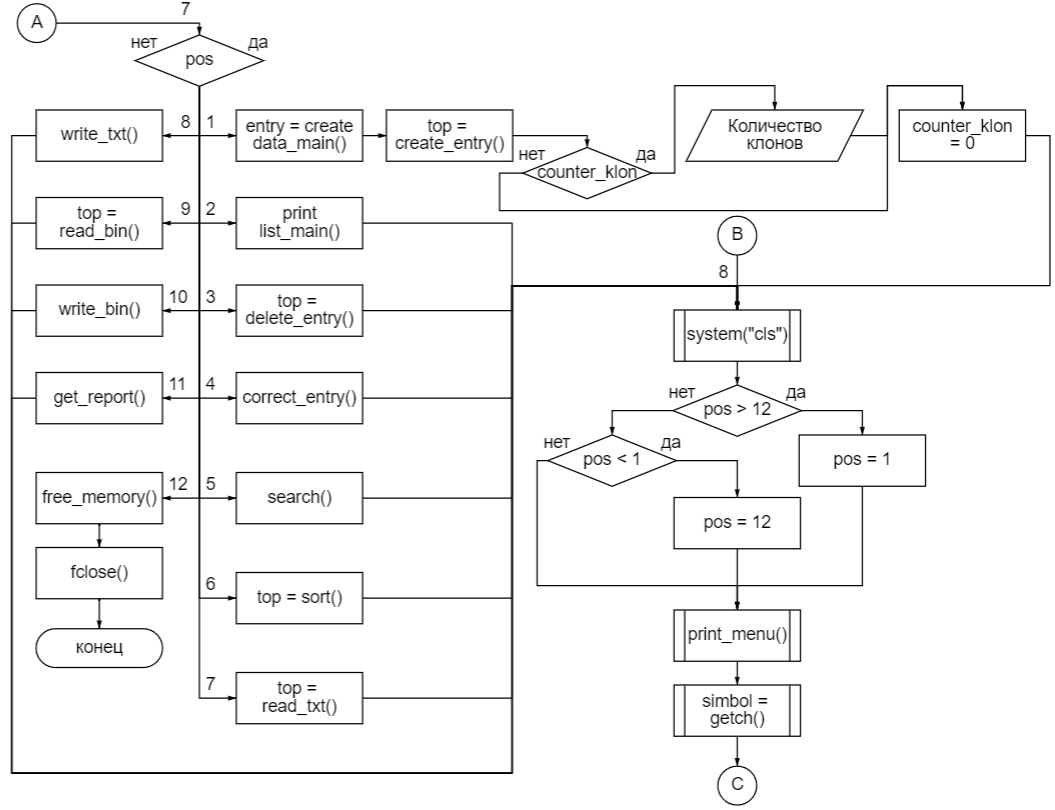


Рисунок 2.2– Схема главной функции main ветка A и B

Основной способ работы с программой – перемещение «выбора» функции посредством нажатия стрелок на клавиатуре. Заключается эта работа в том, что пользователь, к примеру, нажимает стрелку «вниз», после чего происходит итерация цикла и проверка того, что нажал пользователь: стрелку «вверх», стрелку «вниз», клавишу «ввод» (блоки 3,4 и 5 соотвественно) или же любую другую. В зависимости от этого может измениться либо переменная pos, отвечающая за то, какой по номеру строка из массива «mass[]» будет подсвечена в последующем вызове функции print\_menu (см. п. 2.4.20), либо, если нажата клавиша «ввод» произведётся переход в оператор выбора switch (блок 7), в котором по той же переменной pos будет осуществляться вызов соответствующей функции из перечня предложенных. Так и будет бесконечно очищаться экран, проверяться последнее нажатие символа, вызов функции и/или вывод на экран списка действий с подсвеченным текущим значением до того, пока не будет выбран вариант «12) End the work program», в котором будет вызвана функция освобождения памяти, занимаемой списком, и последующим вызовом «return», который и закончит выполнение программы.

Заметим, что при выходе значения переменной pos за границы 1 <pos <12, она(переменная) будет принимать значения кольцевым способом, то есть возвращаться в начало списка или же в конец (блок 8).

2.4.2) Алгоритм функции create\_data\_main приведён на рисунке 2.3:

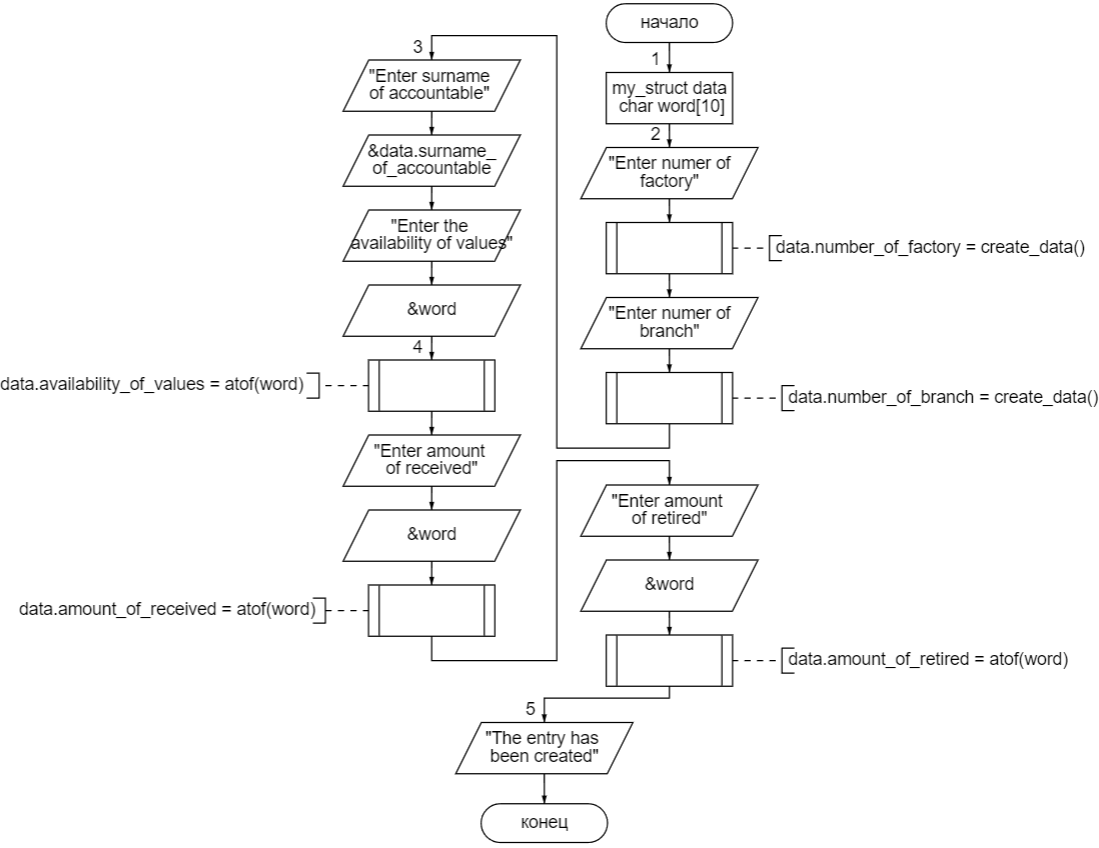


Рисунок 2.3– Схема функции create\_data\_main

Данная функция предназначена для заполнения информационного поля перед созданием записи в списке. Возвращать она будет структуру типа my\_struct (инфо. поле), а в качестве параметра принимает небольшой массив, состоящий из двух сообщений, которые будут использоваться позже.

В начале создаётся переменная data типа информационной структуры my\_struct (блок 1), в которую и будут записываться данные для записи. Второй и последней переменной в данной функции создаётся массив из 10 символов word, который будет являться промежуточным «контейнером» для преобразования данных. Работа функции заключается в следующем: последовательно от начала и до конца функции выводятся сообщения с просьбой ввести пользователя некоторые данные, вслед за чем в первые два раза (блок 2) происходит вызов вспомогательной функции create\_data (см. п. 2.4.3), о которой несколько позже. Возвращаемые значения при её вызове помещаются в инфо поля структуры data. Далее, в (блок 3), произойдёт обычный ввод имени с клавиатуры. А при последующих вводах пользователем данных в блоке 4, они будут помещаться в созданный массив word, значение которого будет помещаться в инфо поля data после преобразования функцией atof (преобразование строки в числовое значение типа float). В (блок 5) будет выведено сообщение об удачной записи, после чего произойдёт возврат функции.

2.4.3) Алгоритм функции create\_data приведён на рисунке 2.4:

Данная функция служит для заполнения информационного поля. Принимает параметрами массив для вывода сообщения о требуемом вводе и режим вывода (0 – номер завода, 1 – номер филиала). Предназначена функция для проверки вводимых пользователем данных в поля «номер завода» и «номер филиала», которые были ограничены мною размерами не менее 100 и не более 1000, то есть для данных полей принимаются трёхзначные целые числа, не содержащие других символов.

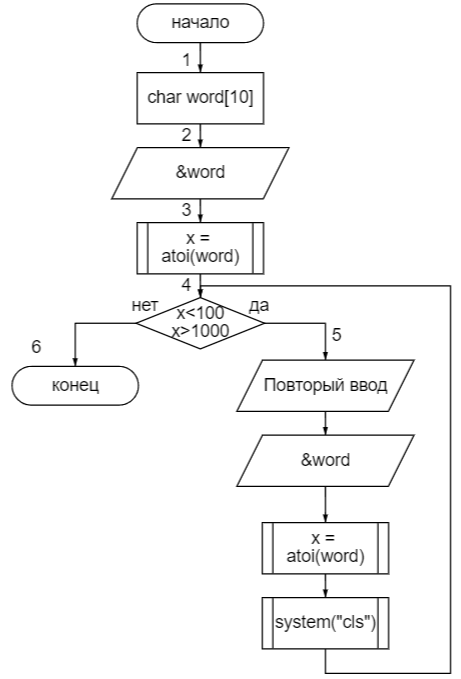


Рисунок 2.4– Схема функции create\_data

Как раз таки для проверки введённых данных мы и создаём всё ту же переменную word (блок 1), в которую считываем с клавиатуры строку (блок 2), преобразуя её функцией atoi (строка в целое int) и присваивая переменной x (блок 3), после чего проверяя на соответствие области 100 <= x < 1000 (блок 4). В случае несоответствия размера, пользователю будет выведено сообщение о предоставляемых ограничениях на ввод и просьба снова ввести данные (блок 5). Данная процедура будет продолжаться до тех пор, пока не будут введены корректные данные (блок 4).

2.4.4) Алгоритм функции create\_entry приведён на рисунках 2.5-2.8:

Функция create\_entry – наиважнейшая функция данной программы. Она служит связующим все записанные пользователем инфо структуры, именно в этой функции происходит выделение динамической памяти под место в списке и вставка инфо поля в соответствующее место списка.

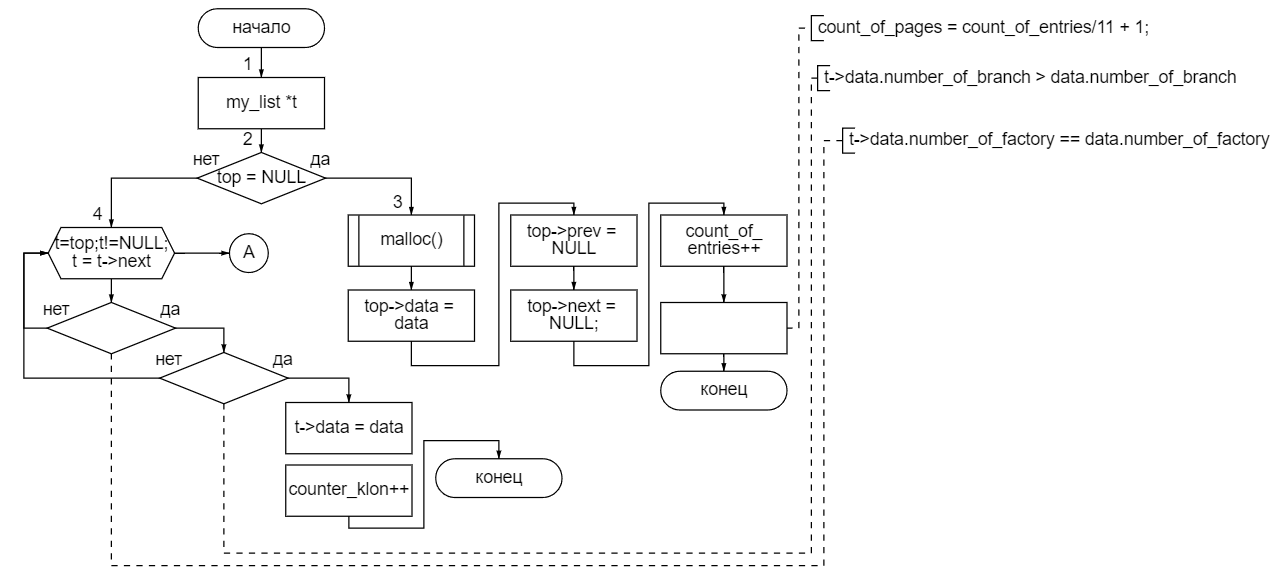


Рисунок 2.5– Схема функции create\_entry

Работа программы организуется следующим образом: создаётся указатель t, необходимый для прохождения и просмотра всего списка (блок 1). Далее происходит проверка на пустоту списка (блок 2) – в случае истинности просто выделяется память под запись, в неё заносится инфо поле, полученное параметром, и в указатели на перёд и назад записываются NULL, после чего глобальная переменная count\_of\_entries (количество записей) увеличивается на 1 и обновляется текущее глобальное значение count\_of\_pages (количество страниц) (блок 3).

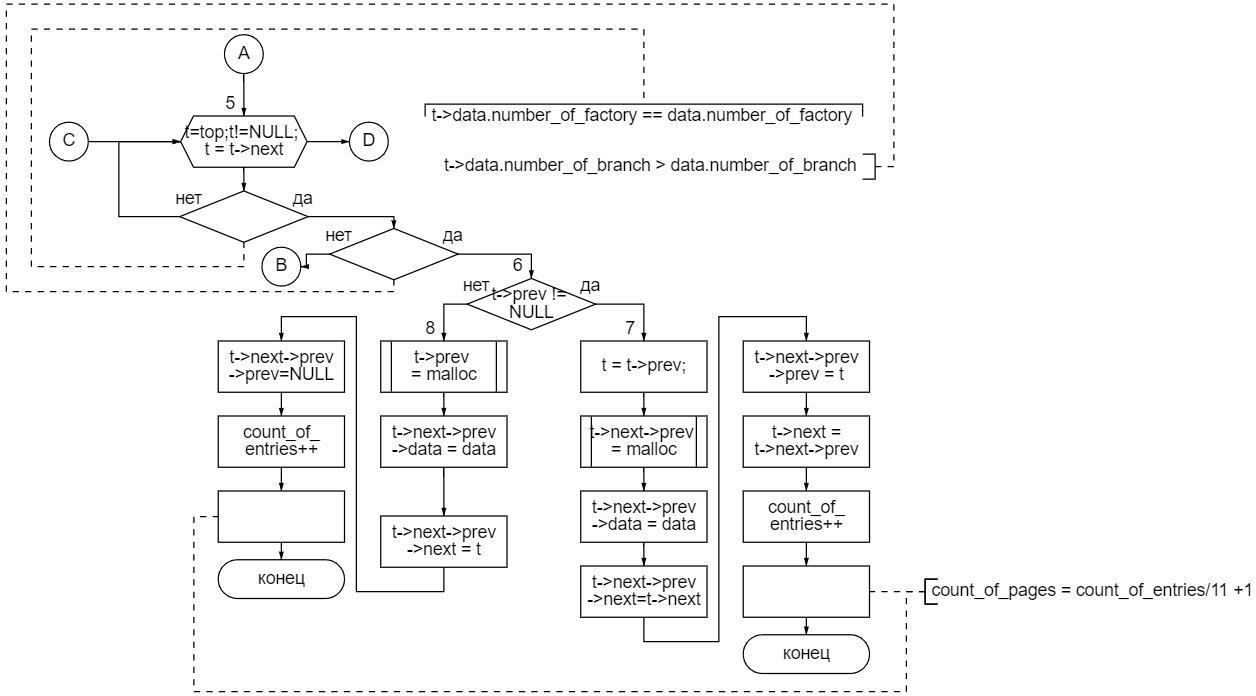


Рисунок 2.6– Схема функции create\_entry

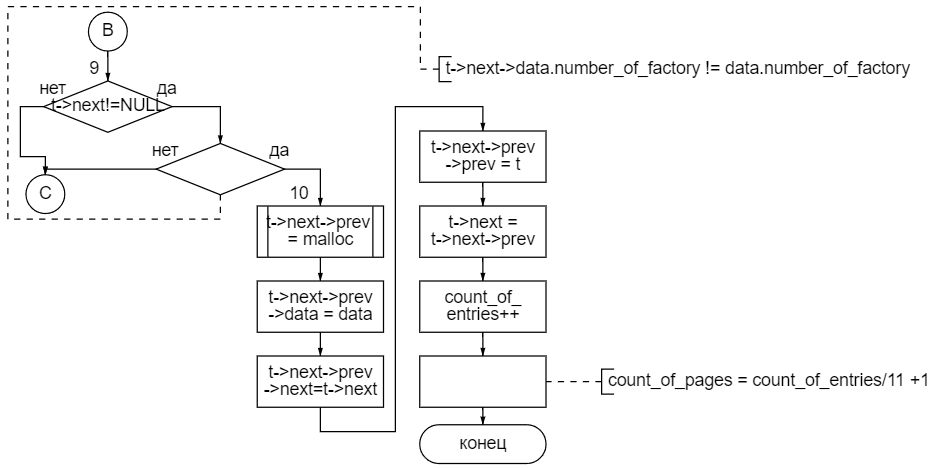


Рисунок 2.7– Схема функции create\_entry

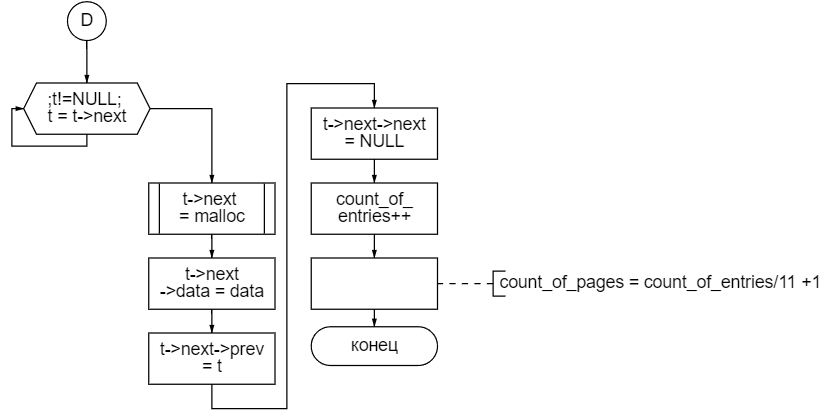


Рисунок 2.8– Схема функции create\_entry

В случае же заполненности списка происходит его полная проверка на уже наличие такой записи (блок 4), и если запись с таким же номером завода и филиала находится, то для неё просто обновляются эти данные, и поднимается счётчик counter\_klon (количество клонов). Если и клон записи не находится, то происходит поиск такого же завода (блок 5), и запись добавляется в порядке возрастания филиала (блок 6-10), если же такого завода нет, то запись добавляется в конец списка (ветка D).

2.4.5) Алгоритм функции check\_list приведён на рисунке 2.9:

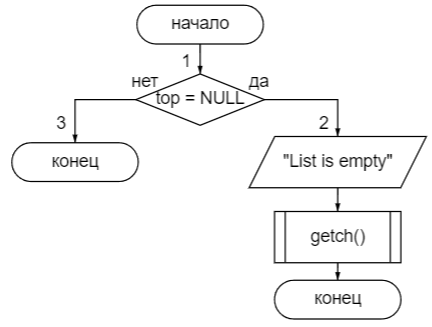


Рисунок 2.9– Схема функции check\_list

Данная функция проверяет заполненность списка (блок 1), в случае пустоты выводится соответствующее сообщение и происходит возврат в функцию, вызвавшую эту проверку (блок 2).

2.4.6) Алгоритм функции print\_list\_main приведён на рисунке 2.10:

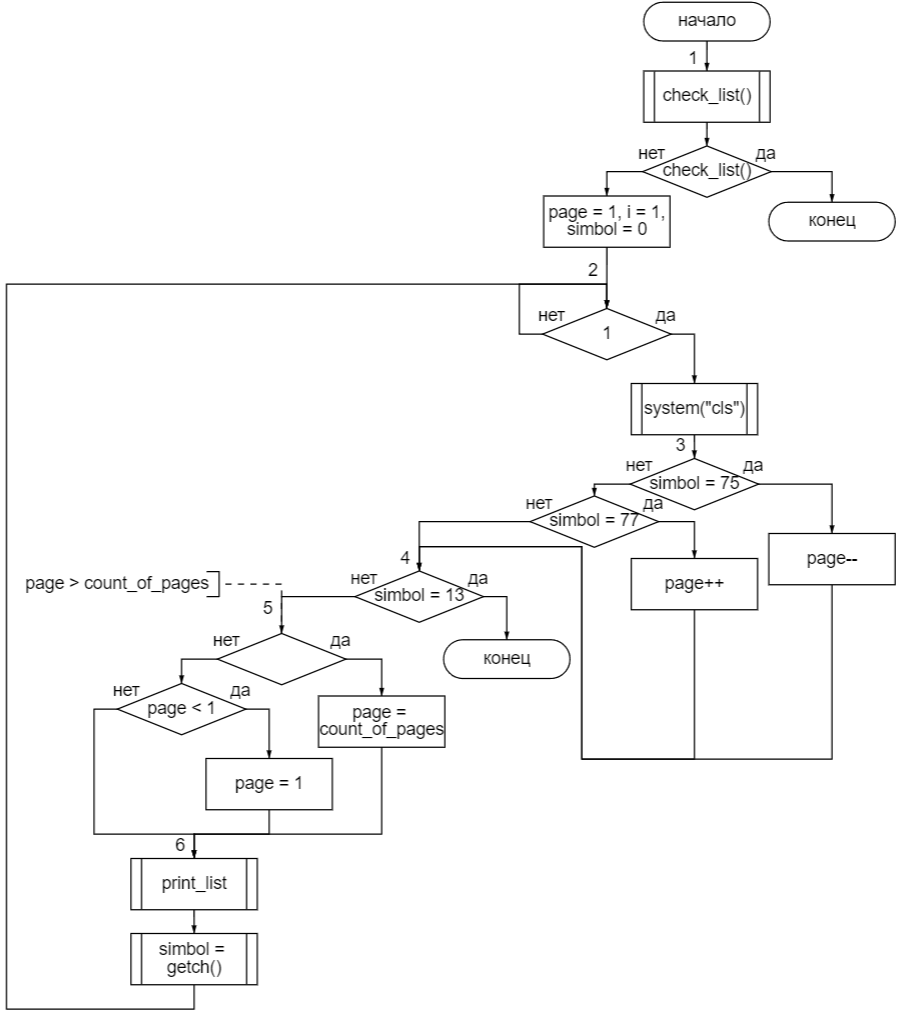


Рисунок 2.10– Схема функции print\_list\_main

Функция служит координатором вывода списка перечня доступных действий в программе. Почему только координатором? Потому что в ней происходит только управление положением текущей страницы и проверка на выход из функции, а сам же вывод списка производит общая функция print\_list (см. п. 2.4.7), используемая и другими функциями. Сначала происходит проверка пустоты списка - функцией check\_list (см. п. 2.4.5) (блок 1). Если выхода не произошло, значит есть, что вывести на экран. Далее создаются переменные: i(индекс для нумерации записей), symbol(для считывания нажатия) и page(текущая страница). Далее по аналогии с функцией main происходит вход в бесконечный цикл (блок 2), в котором также чистится экран, проверяется значение последней нажатой клавиши symbol (блок 3), и если была нажата стрелочка «вправо» или «влево», то значение текущей страницы pos соответственно увеличивается или уменьшается, при выходе за границы 1<page< count\_of\_pages (глоб. переменная), page получает значение этого края, то есть остаётся на месте, в отличие от pos в main (блок 5). Если же нажата клавиша «ввод» (блок 4), то произойдёт выход из функции и управление вернётся в функцию main. При иных нажатиях клавиш, включая стрелки, следом производится вызов печатающей список функции print\_list (2.4.7) (блок 6), которой передаётся значение текущей страницы, которую необходимо вывести пользователю.

2.4.7) Алгоритм функции print\_list приведён на рисунке 2.11:

Эта функция служит для непосредственного вывода всех записей в виде таблицы на экран пользователя. В качестве параметров она принимает указатель на начало списка (top), позицию записи (pos), которую надо подсветить, текущую страницу списка (page) и режим работы (mode) (0 – простой вывод списка, 1 – вывод списка с выделением конкретной записи (для функций удаления и корректировки)). Итак, в начале программы создаётся переменная hStdOut типа HANDLE – она будет использоваться для смены цвета заднего фона выделяемой строки.

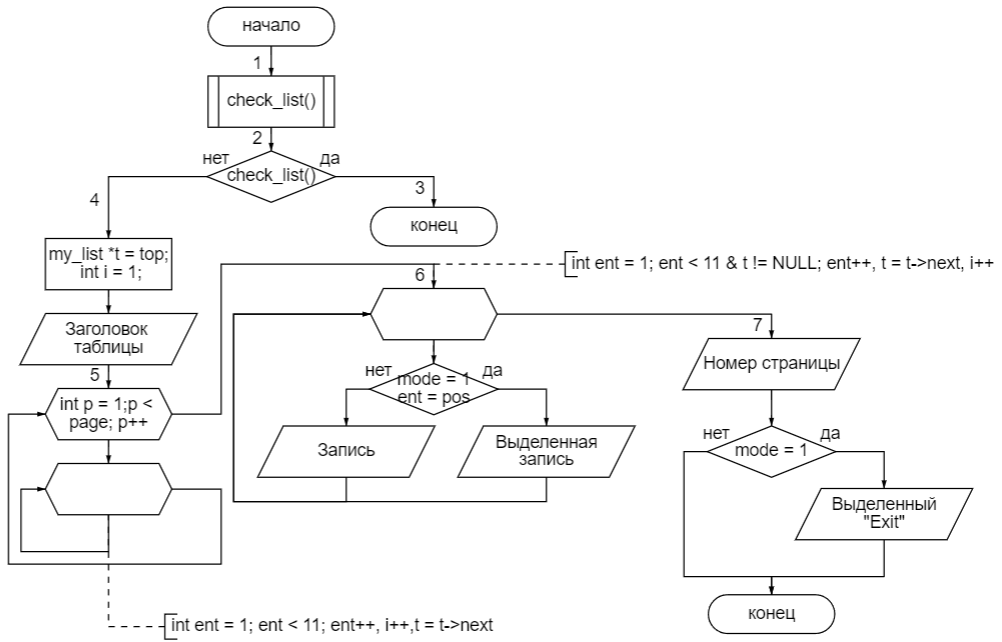


Рисунок 2.11– Схема функции print\_list

Далее вызывается стандартная проверка на пустоту списка (блок 1). Следом создаётся указатель t для перемещения по списку и индекс i для вывода номера той или иной записи (блок 4). После выводится заголовок таблицы, и происходит «перелистывание» записей (блок 5): по 10 раз в цикле с параметром p, пока p меньше заданной параметром страницы page, будет происходить присваивание к t следующего элемента списка – t->next. Таким образом, мы page-1 раз пролистнём по 10 записей (на страницу выводится 10 записей) и дойдём до нужной страницы. Далее, в блоке 6, если это не последняя страница списка (page != count\_of\_pages), на экран выведется 10 записей, если же вывести нужно последнюю страницу списка, то цикл пройдёт, пока t!=NULL. Параллельно с выводом каждой записи производится проверка: если режим вывода равен 1 (вывод с выделением записи) и номер выводимой строки равен заданной позиции pos, то сменить цвет, вывести эту запись, и сменить цвет обратно, после чего оператором continue перейти к итерации цикла и вывести остальные записи. После выхода из цикла выведется значение текущей страницы и снова проверится mode (блок 7). В случае 0 выведется выделенная строка “Exit”, как бы показывая пользователю, что доступен только выход из цикла. Работа функции вывода на этом заканчивается.

2.4.8) Алгоритм функции delete\_entry приведён на рисунке 2.12-2.16:

Данная функция, как видно из её названия, предназначена для удаления записи в списке. Своими параметрами она принимает только ссылку на начало списка (top). Возвращать она будет такую же ссылку на верх, ведь эта самая ссылка в процессе удаления может меняться.

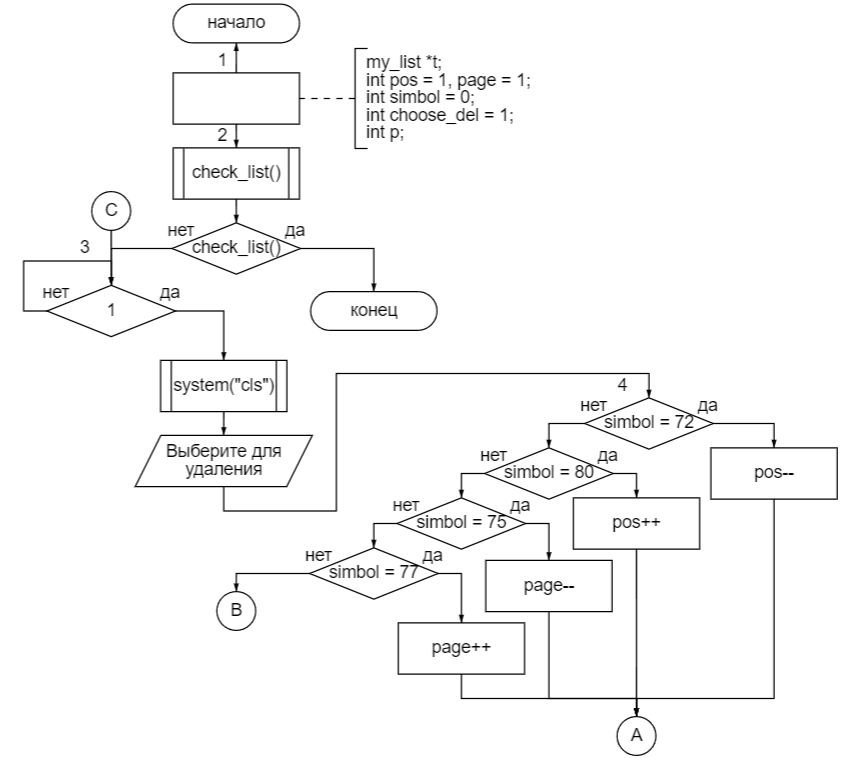


Рисунок 2.12– Схема функции delete\_entry ветка начало

В подготовительном этапе происходит создание переменных (блок 1): t (указатель для перемещения по списку), pos (счётчик выделения), page (страница вывода), simbol (приёмник нажатия клавиши), choose\_del (счётчик для подтверждения удаления), p ( счётчик для перемещения к заданной странице).

Далее вызывается уже стандартная проверка пустоты файла (блок 2).

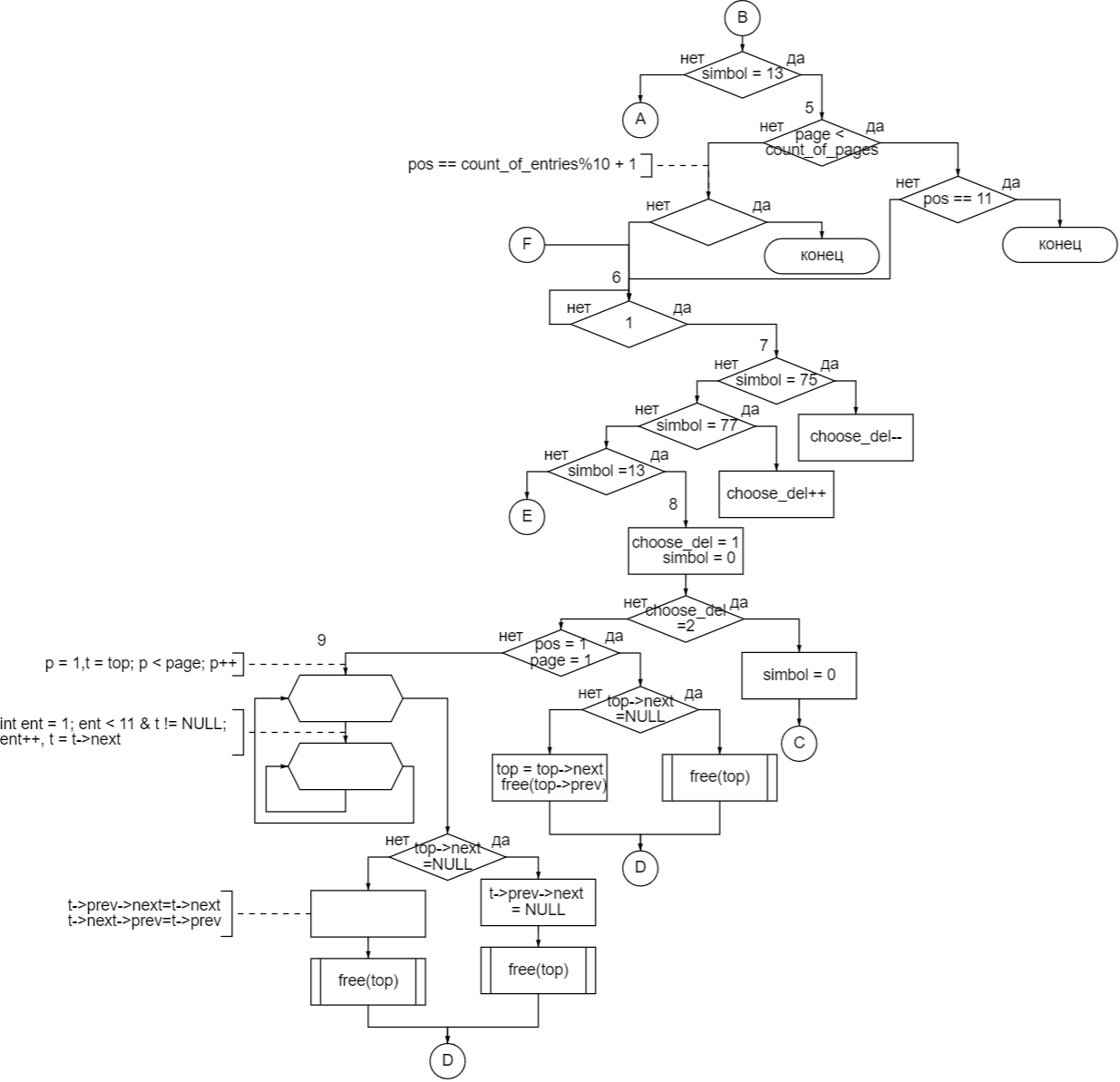


Рисунок 2.13– Схема функции delete\_entry ветка B

После происходит вхождение в основной блок функции (блок 3) – бесконечный цикл, в начале которого чистится экран, выводится сообщение с просьбой выбрать запись для удаления, и проверяется значение переменной simbol (по умолчанию 0) (блок 4). Из-за нулевого значения переменной произойдёт пропуск всех проверяющих блоков, и произойдёт вызов функции print\_list (печать списка) c режимом 1(выделение записи) (блок 11) (см. п. 2.4.7). Далее произойдёт считывание с клавиатуры в simbol и следующая за ней итерация цикла.

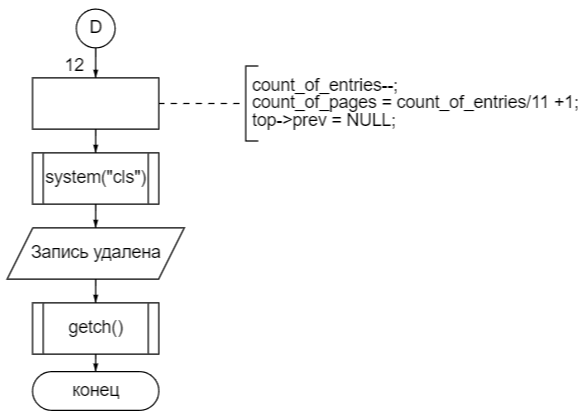


Рисунок 2.14– Схема функции delete\_entry ветка D

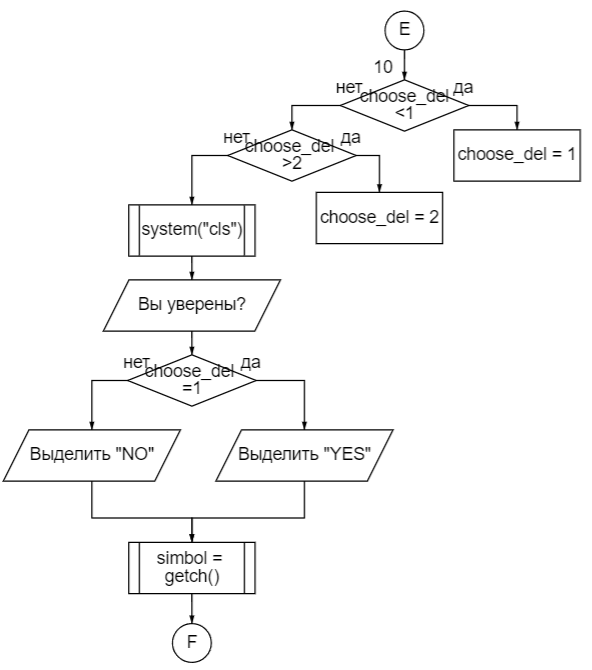


Рисунок 2.15– Схема функции delete\_entry ветка E

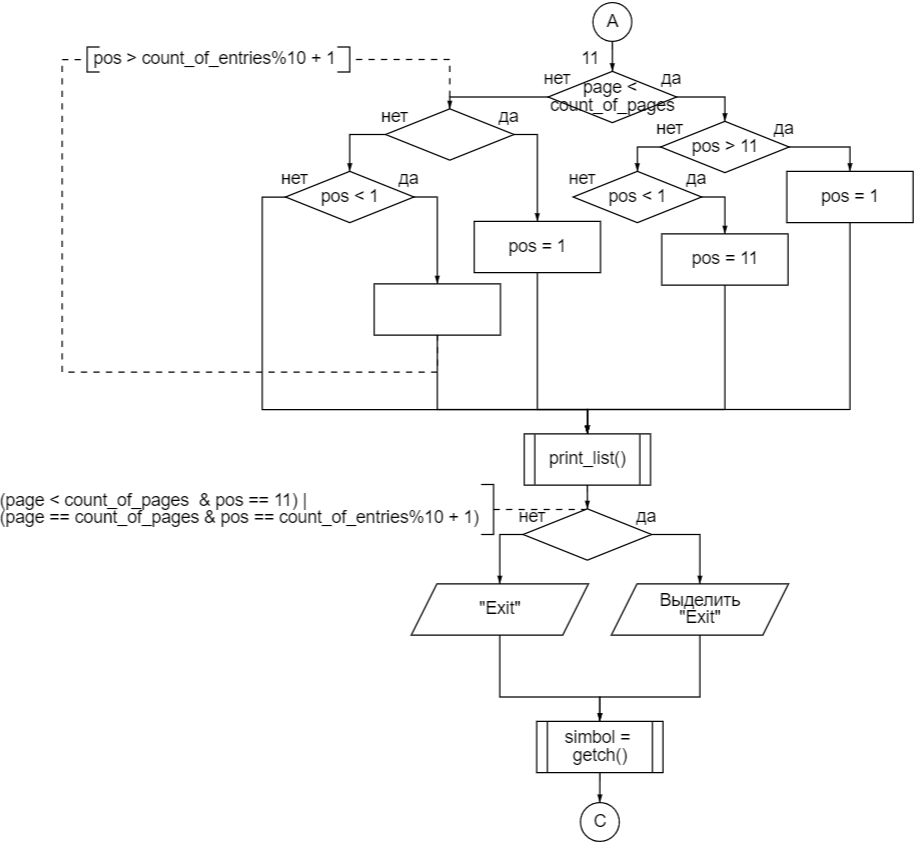


Рисунок 2.16– Схема функции delete\_entry ветка A

Далее, в (блок 3) произойдёт проверка переменной, из которой следуют уже привычные действия:

а) Стрелка вверх – уменьшить переменную позиционирования pos,

б) Стрелка вниз – увеличить переменную позиционирования pos,

в) Стрелка влево – уменьшить переменную выводимой страницы page,

г) Стрелка влево – увеличить переменную выводимой страницы page,

д) Enter – перейти к (блок 5) для дальнейших действий.

Нажимая, клавишу Ввод, мы подтверждаем выбор, которым может являться либо одна из выведенных записей, либо выбор «Exit», который выйдет из функции.

Если же выбрана запись, то произойдёт переход к (блок 6) – новой бесконечной функции, в которой будет выведено сообщение с вопросом «Вы уверены, что хотите удалить запись?», и будут предложены два варианта действий: «Да» и «Нет» (блок 10). Цикл бесконечный, и работает он по аналогии со всеми циклами выбора в программе: проверка переменной ввода с клавиатуры simbol (блок 7), и при нажатии стрелок «Вправо» или «Влево» будет происходить манипуляция с переменной choose\_del, которая будет использоваться для выделения выбора удаления. При нажатии «Ввод» будет происходить либо переход в (блок 8), в котором произведётся удаление записи, либо возврат к списку путём перехода на метку в начало цикла (блок 6).

После удаления записи производится переход на метку end (блок 12), в котором происходит чистка экрана и выводится сообщение о том, что запись удалена.

2.4.9) Алгоритм функции correct\_entry приведён на рисунках 2.17-2.21:

Функция реализует возможность изменения (корректировки) уже существующей записи без её замены в списке. Возможно изменение любого поля записи.

Принимает функция в параметр только указатель на начало списка, а возвращать ей ничего не требуется.

В (блок 1) создаются переменные, необходимые для функционирования программы: t(указатель для перемещения по списку),j (вспомогательный указатель для прохода по списку) , data (информационная структура для хранения промежуточных данных), corrected (строка для хранения введённых пользователем данных), pos (текущая выбранная запись в списке), page (текущая страница для вывода), pos\_entry (для хранения позиции текущего выбора поля корректировки), simbol (для считывания нажатой клавиши), simbol\_entry(для считывания нажатой клавиши при выборе поля корректировки), flag (флаг для проверки существования похожей записи), counter(счётчик для проверки существования похожей записи), counter2 (такой же счётчик).

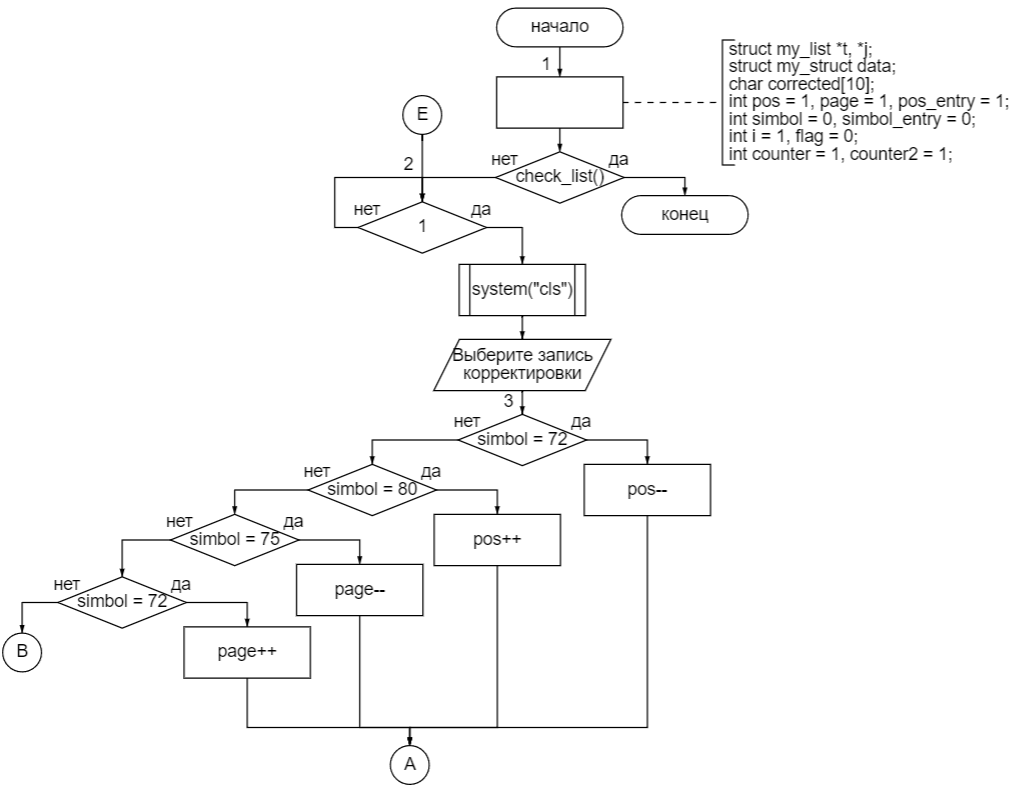


Рисунок 2.17– Схема функции correct\_entry ветка начало

Далее проверяется пустота списка, и происходит вход в бесконечный цикл (блок 2) – в нём выводятся записи списка, и пользователь путём нажатия клавиш «стрелки» выбирает запись, которую хочет отредактировать.

Проверка нажатия происходит в (блок 3). Если пользователь нажал клавишу «Enter», функция переходит в ветку B (См. далее).

Если нажимается какая-либо другая клавиша, то в Ветке А происходит контроль переменной pos на выход за границы, и вызывается функция вывода таблицы print\_list (см. п. 2.4.7).

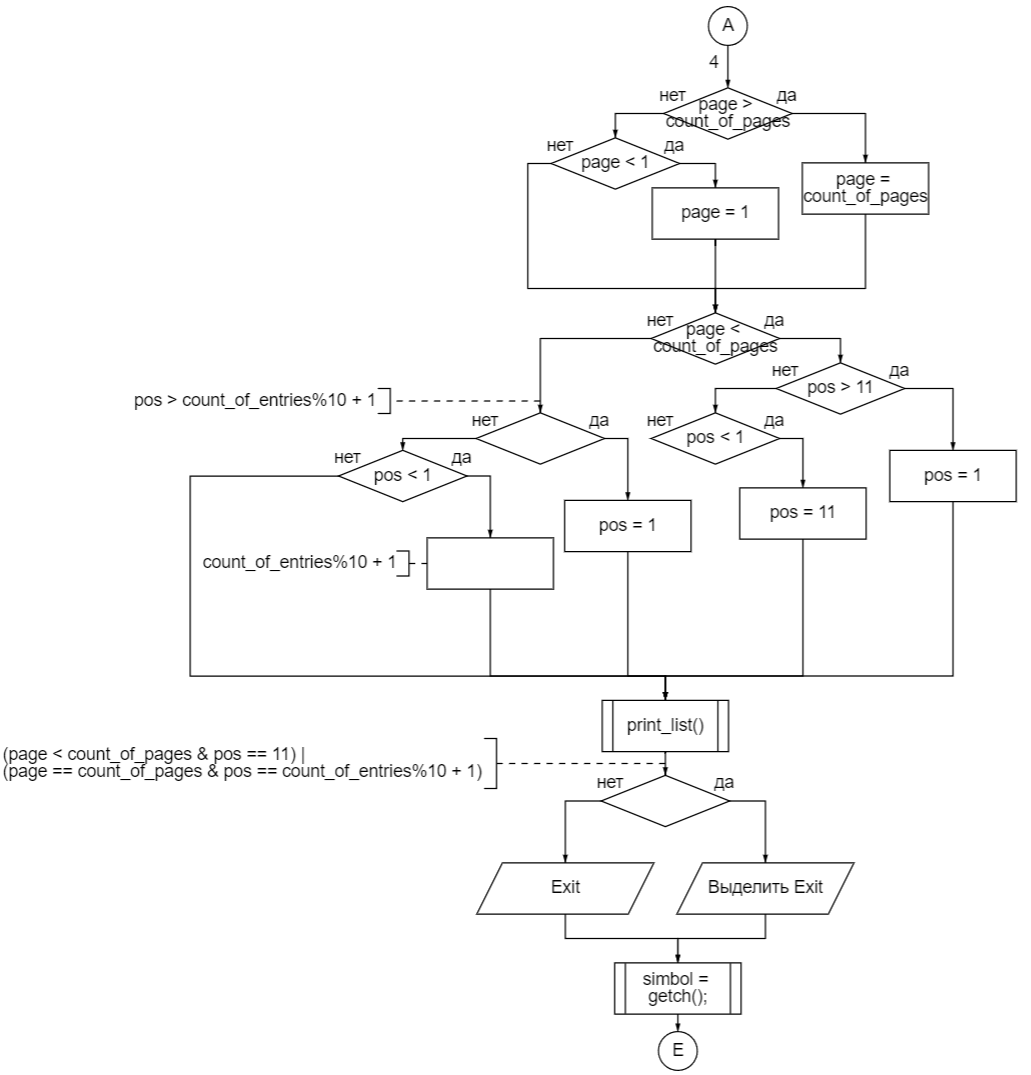


Рисунок 2.18– Схема функции correct\_entry ветка A

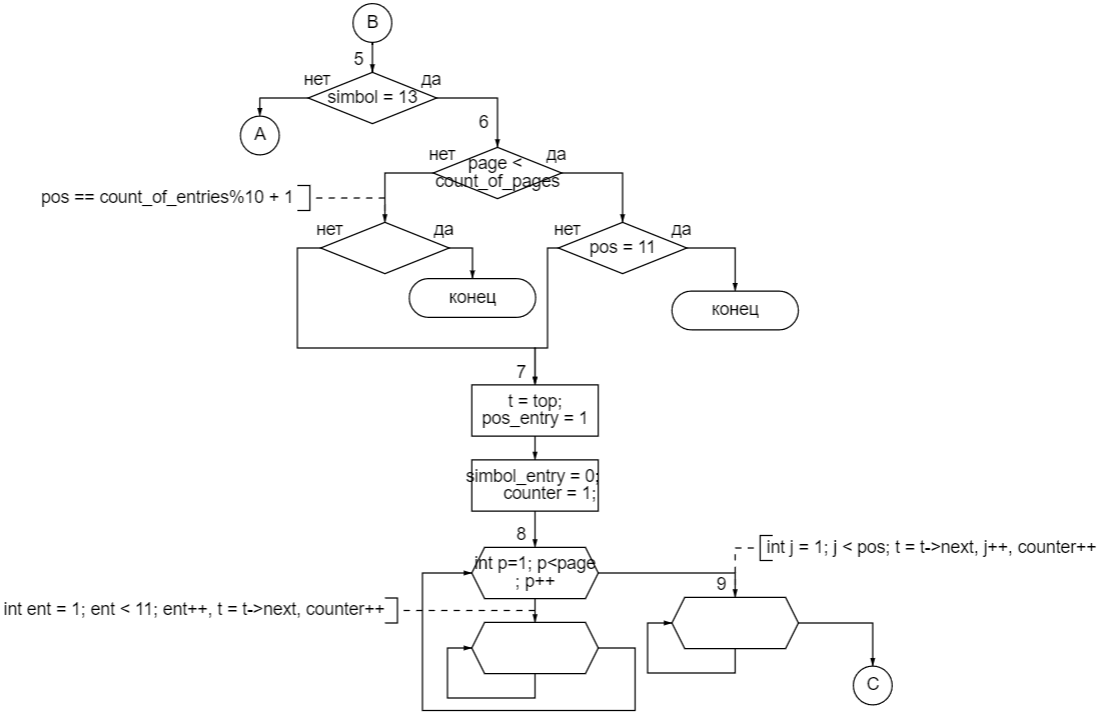


Рисунок 2.19– Схема функции correct\_entry ветка B

В начале ветки проверяется, не выбрал ли пользователь вариант выхода из функции «Exit» (блок 6).

Если не выбрал, то в циклах с параметром (блок 8 и 9) происходит переход к выбранной на редактирование записи. Дальнейшие действия продолжаются в ветке С (см. далее).

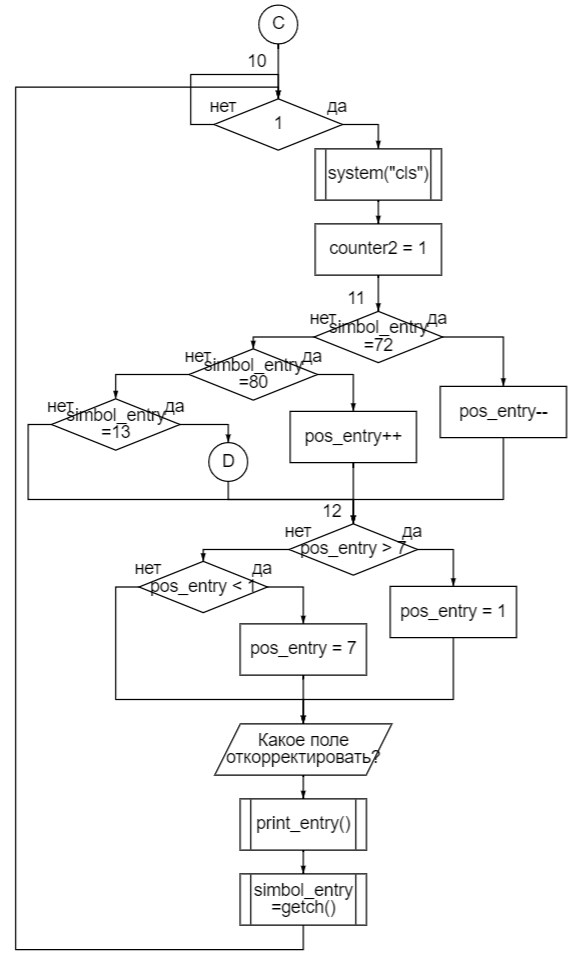
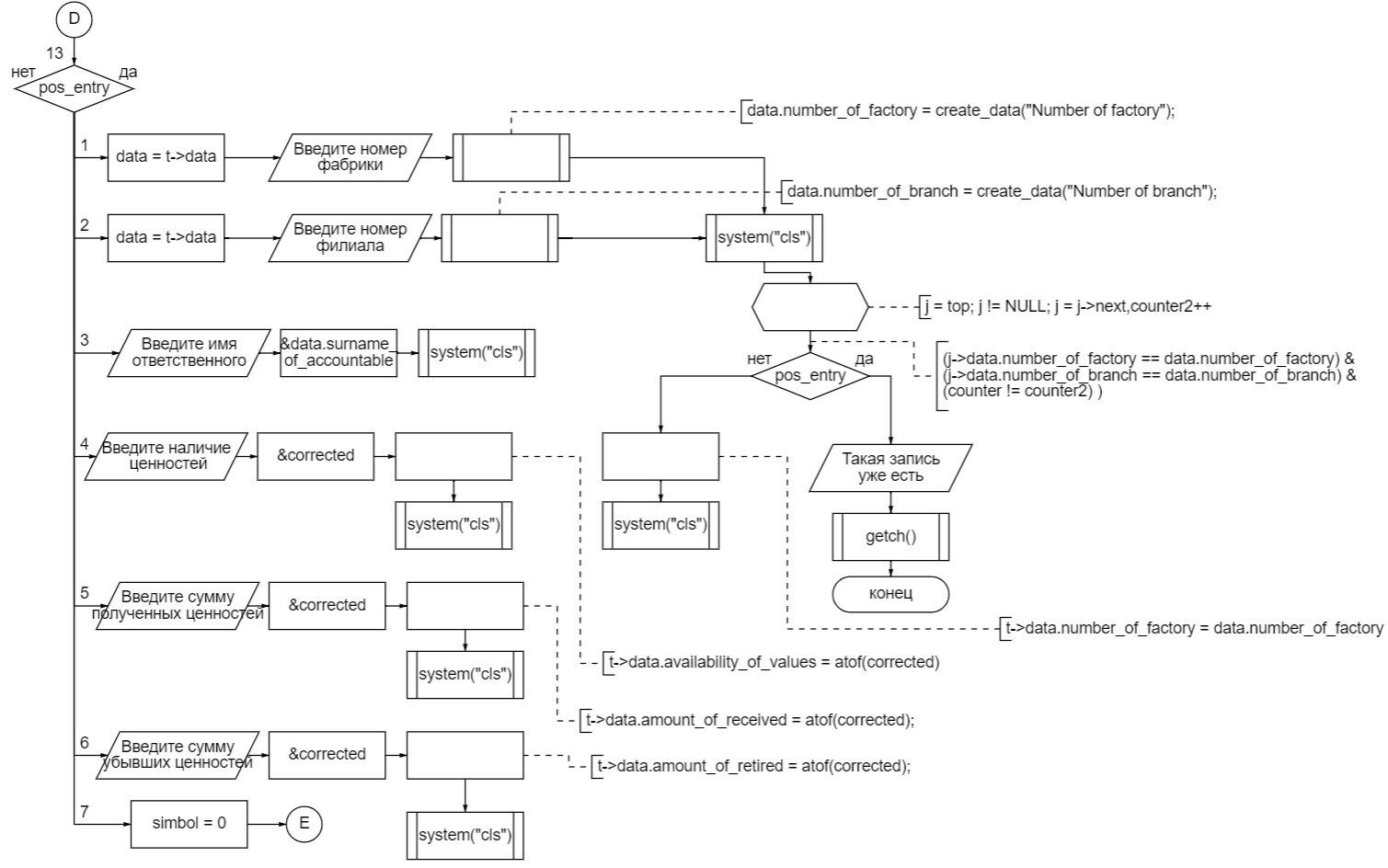


Рисунок 2.20– Схема функции correct\_entry ветка C

Здесь происходит вывод на экран полей записи с выделением той, которая выбрана в данных момент для корректировки. Также есть возможность выйти обратно к таблице записей.

В бесконечном цикле (блок 10) происходит очистка экрана и дальнейшая проверка нажатия клавиши (блок 11). Если пользователь нажал «Enter», происходит переход к самому редактированию указанного поля в ветке D (см. далее). Иначе, в (блок 12) происходит проверка переменной pos\_entry на выход за границы и последующий вывод предлагаемых для редактирования полей.

 Рисунок 2.21– Схема функции correct\_entry ветка D

Это основная ветка программы. Здесь с помощью оператора выбора (блок 13) по переменной pos\_entry происходит переход к редактированию одного поля записи. Программа просит пользователя ввести данные. После чего проверяет корректность их ввода, а также существование уже существующей записи во избежание клонирования записей. Если такая запись есть (с похожим заводом и филиалом) происходит выход из функции.

2.4.10) Алгоритм функции sort приведён на рисунках 2.22-2.25:

Функция реализует сортировку списка по заданному полю в заданном направлении. Принимает в качестве параметра указатель на начало списка (top). Ничего не возвращает.

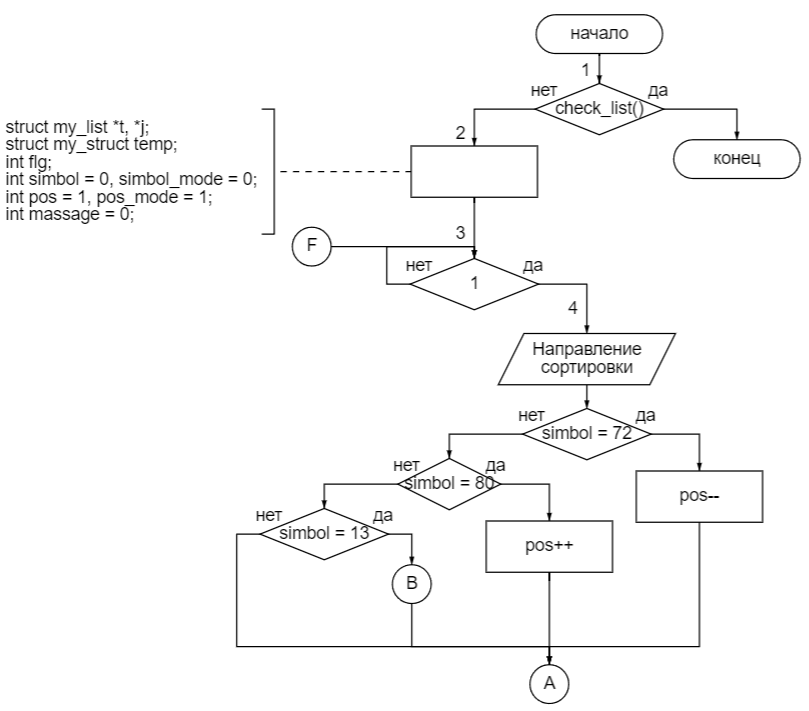


Рисунок 2.22– Схема функции sort ветка начало

Сначала функцией check\_list (см. п. 2.4.5) проверяется пустота списка. Если он не пуст, создаются необходимые для работы функции переменные:

t и j (указатели для перемещения по списку и его сортировки), temp (инфо структура для хранения данных записи), flg (флаг для проверки условия обмена записей), simbol (для считывания нажатия клавиши в меню выбора направления сортировки), simbol\_mode (для считывания клавиши в меню выбора поля сортировки), pos (позиция для выделения строки в первом меню), pos\_mode (позиция выделения строки во втором меню), massage (флаг для проверки вывода сообщения о сортировке).

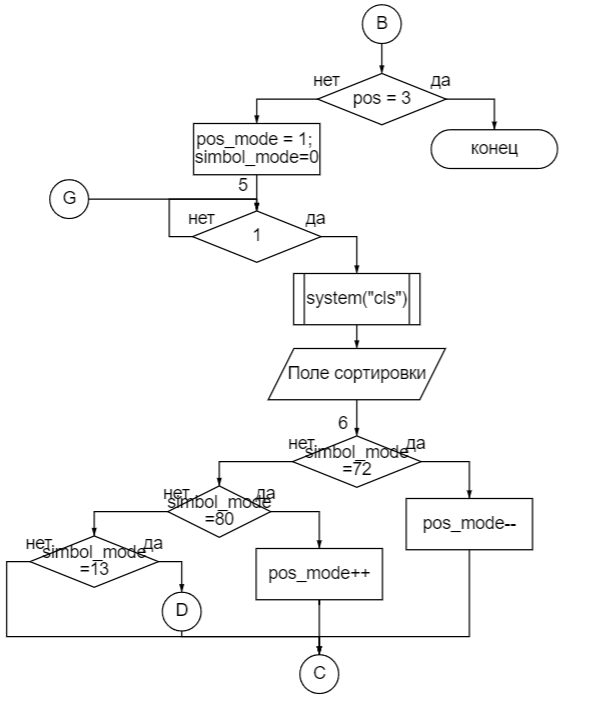


Рисунок 2.23– Схема функции sort ветка B

В ветке происходит выбор поля, по которому произведётся сортировка. За сам выбор и выделение строки выбора отвечает переменная pos\_mode, которая изменяется путём нажатия стрелок «вверх» и «вниз» (блок 6). Нажатием клавиши «enter» пользователь подтверждает свой выбор поля и производится переход в ветку сортировки D (см. далее).

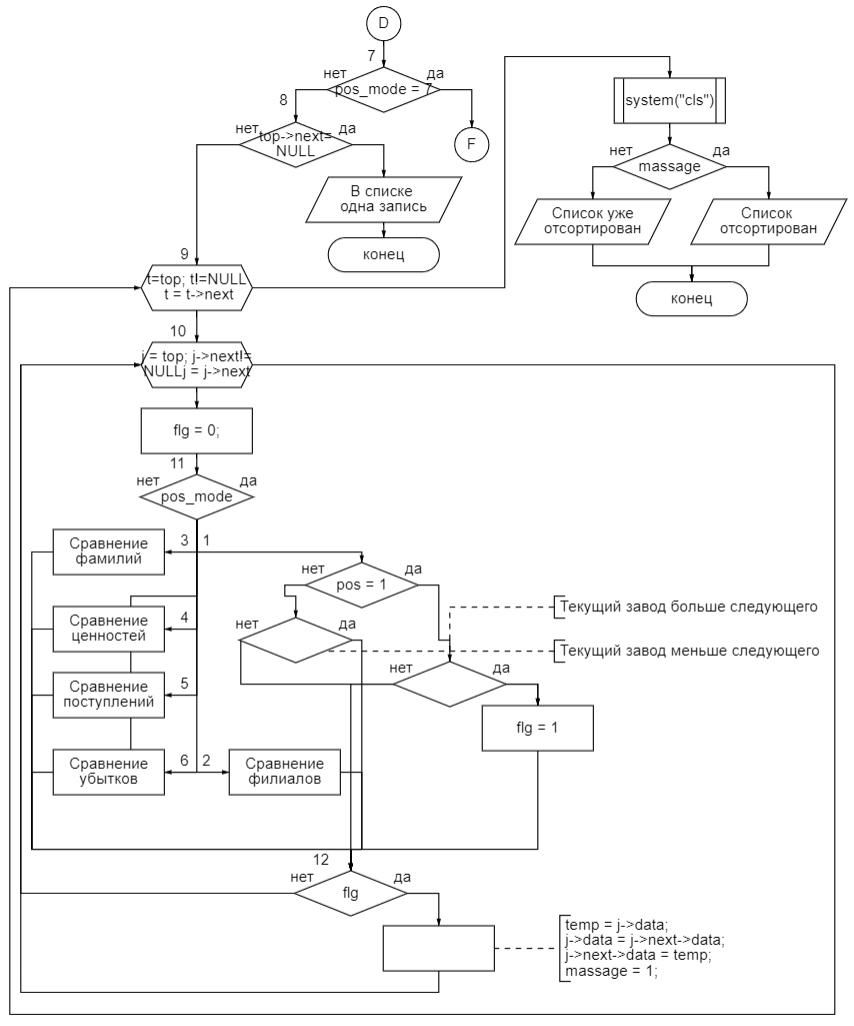


Рисунок 2.24– Схема функции sort ветка D

Ветка реализует непосредственную сортировку списка. Сначала проверяется, не выбрал ли пользователь вариант «Exit», если выбрал, происходит выход в меню выбора направления поиска. В ином случае происходит сама сортировка методом пузырька (блоки 9-10). Если же в списке всего одна запись, пользователя уведомят об этом, и программа завершит свою работу.

После сортировки выводится сообщение о том, что список был или не был отсортирован.

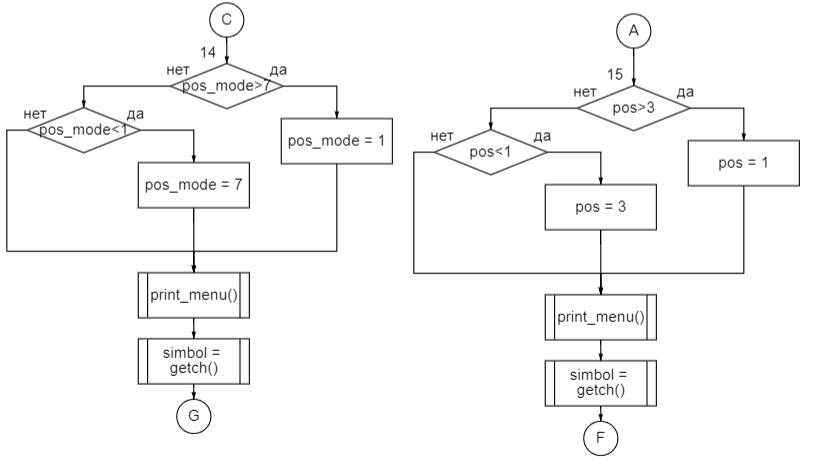


Рисунок 2.25– Схема функции sort ветки A и C

Здесь происходит контроль выхода переменных позиции за свои границы и последующий вывод либо предлагаемых полей поиска (ветка С), либо направление поиска (ветка А). После чего считывается нажатие с клавиши и происходит переход в ветки G и F соответственно.

2.4.11) Алгоритм функции read\_txt приведён на рисунке 2.25:

Функция служит для создания списка с помощью текстового файла.

В качестве параметра она принимает указатель на начало списка, в который будет добавлять новые записи.

В (блок 1) создаются переменные: entry (структура-посредник для считывания из файла), n(для хранения возвращаемого функцией чтения значения), flg (флаг для проверки корректности поданных данных), count (для количества записей-браков), word1,word2,word3,word4,word5 (массивы для считывания и преобразования значений), skobka (переменная-заглушка для считывания номера-записи).

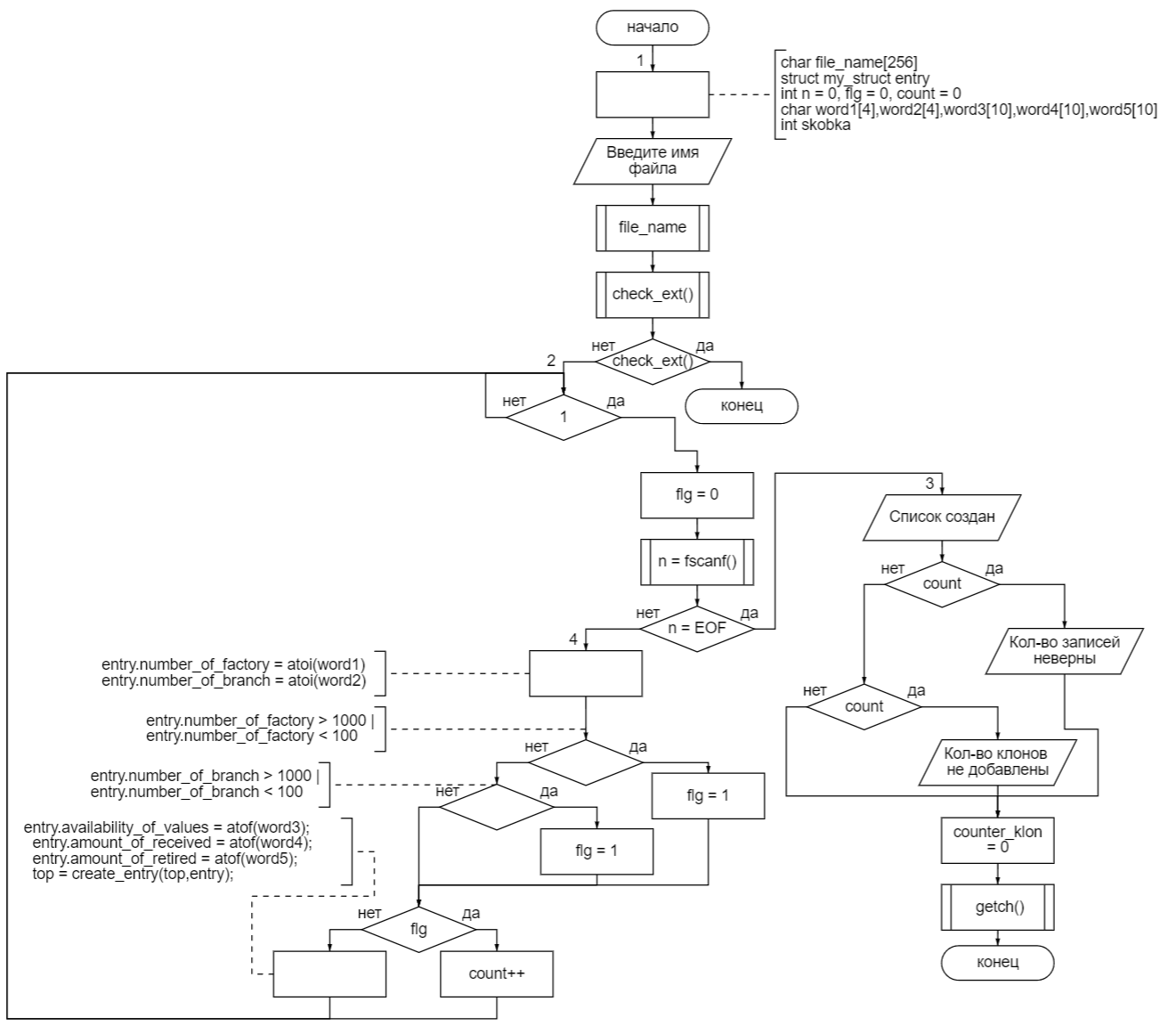


Рисунок 2.26– Схема функции read\_txt

Далее пользователь вводит название файла для чтения, и вызывается функция проверки расширения файла check\_ext с режимом работы 1(чтение из текстового файла) (см. п. 2.4.15), в случае отрицательного результата происходит выход из функции. В ином же случае происходит вход в бесконечный цикл (блок 2), в котором и происходит последовательное считывание переменных из файла. Условием выхода из цикла является переменная n, равная значению EOF (конец файла). В случае, когда условие не выполняется, происходит переход к (блок 4), в котором происходит преобразование переменных-строк к числовым типам, которые помещаются в структуру-посредник entry, После этого происходит проверка некоторых переменных на выход за границы(100<=значение<1000). Если происходит выход, то запись помечается как брак, и поднимается флаг flg, после проверки которого определяется, добавить ли значение count и совершить итерацию цикла, либо преобразовать остальные переменные и вызвать функцию create\_entry (см. п. 2.4.4), которая создаст запись в списке. При n = EOF происходит переход к блоку 3, в котором пользователя осведомляют о создании списка, при наличии выводится количество браков и клонов и происходит выход из функции.

2.4.12) Алгоритм функции write\_txt приведён на рисунке 2.27:

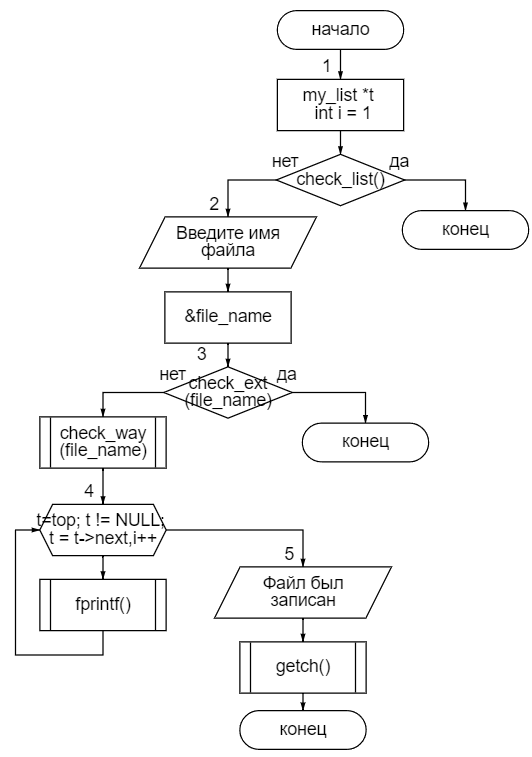


Рисунок 2.27– Схема функции write\_txt

Функция осуществляет непосредственное сохранение списка в текстовый файл. Параметром она принимает лишь указатель на начало списка(top), а указатель на файловый поток является в программе глобальным, и его открытие происходит в функции check\_ext (см. п. 2.4.15).

В (блок 1) создаются: указатель t (для перемещения по списку) и счётчик вывода i (для нумерации записей в файле).

Далее происходит стандартная проверка на пустоту файла, и в случае наполненности пользователь вводит имя файла (блок 2), после чего происходит вызов функции check\_ext с режимом работы 2 (запись в текстовый файл) (см. п. 2.4.15) для проверки корректности введённого расширения файла. Если расширение удовлетворяет заданному, то вызывается функция проверки пути check\_way (см. п. 2.4.16). После в (блок 4) происходит основная работа функции – проход по списку и последовательная запись структур в файл функцией fprintf. Когда указатель t доходит до конца списка, цикл завершается и в завершающем (блок 5) выводится сообщение об успешной записи файла, и функция завершает свою работу.

2.4.13) Алгоритм функции read\_bin приведён на рисунке 2.28:

Функция создаёт список из типизированного(бинарного) файла. Параметром принимает указатель на начало списка (top), и возвращает такой же указатель.

В (блок 1) создаются переменные: entry (структура-посредник для считывания из файла), n (для хранения возвращаемого функцией чтения значения), flg (флаг для проверки создания списка), count (для количества записей-браков).

Далее происходит вызов функции check\_ext с режимом работы 3 (чтение из бинарного файла) (см. п. 2.4.15), после которого, если не произошёл выход из функции, происходит вход в бесконечный цикл (блок 2), в котором последовательно считываются структуры из файла (функцией fread), а возвращаемое ею значение помещается в n. Работает цикл до тех пор, пока n>0 ( кол-во считанный функций больше 0).

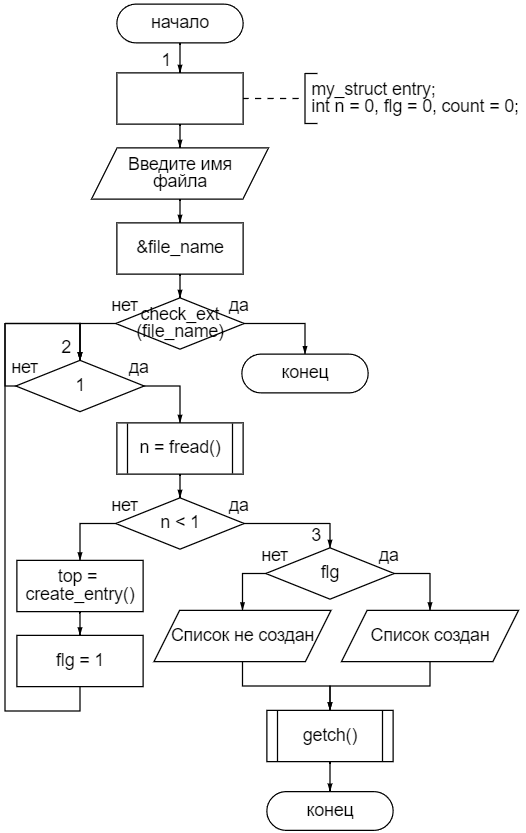


Рисунок 2.28– Схема функции read\_bin()

После чтения из файла структура-посредник entry отправляется параметром в функцию create\_entry для создания записи (см. п. 2.4.4), и поднимается флаг flg. Когда в файле структур не остаётся, происходит выход из цикла и в (блок 3) при положительном значении флага flg пользователя уведомляют об успешном создании списка, после чего работа функции завершается.

2.4.14) Алгоритм функции write\_bin приведён на рисунке 2.29:

Функция осуществляет непосредственное сохранение списка в типизированный файл. Параметром она принимает лишь указатель на начало списка(top), а указатель на файловый поток является в программе глобальным, и его открытие происходит в функции check\_ext (см. п. 2.4.15).

В (блок 1) создаётся только указатель t (для перемещения по списку).

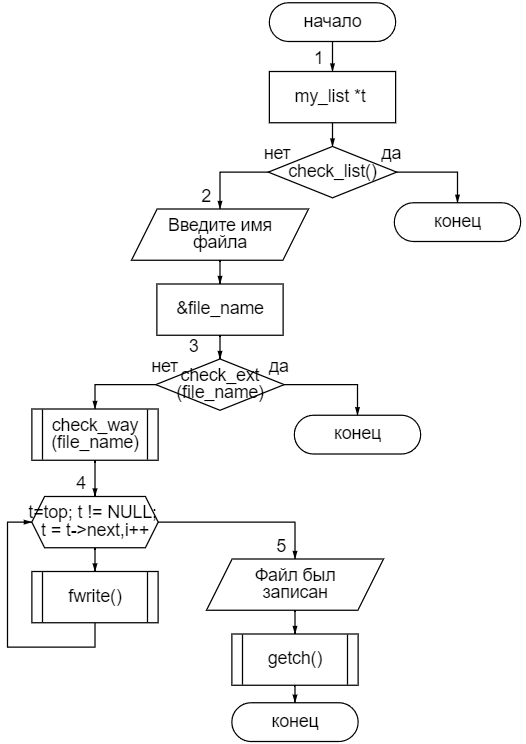


Рисунок 2.29– Схема функции write\_bin

Далее происходит стандартная проверка на пустоту файла, и в случае наполненности пользователь вводит имя файла (блок 2), после чего происходит вызов функции check\_ext с режимом работы 4 (запись в бинарный файл) (см. п. 2.4.15) для проверки корректности введённого расширения файла. Если расширение удовлетворяет заданному, то вызывается функция проверки пути check\_way (см. п. 2.4.16). После в (блок 4) происходит основная работа функции – проход по списку и последовательная запись структур в файл функцией fwrite. Когда указатель t доходит до конца списка, цикл завершается и в завершающем (блок 5) выводится сообщение об успешной записи файла, и функция завершает свою работу.

2.4.15) Алгоритм функции check\_ext приведён на рисунке 2.30:

Функция предназначена для проверки корректности введённого пользователем названия файла и непосредственного открытия файлового потока ввода/вывода.

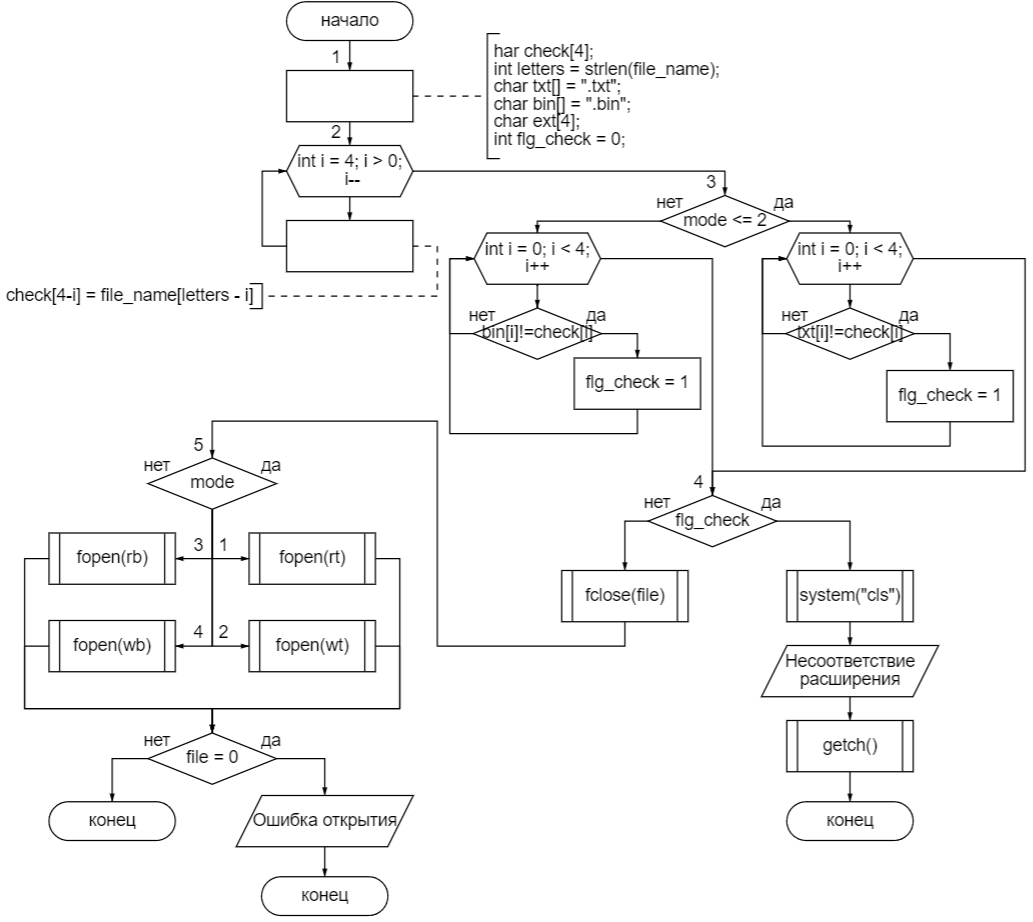


Рисунок 2.30– Схема функции check\_ext

Параметрами функция принимает только номер режима открытия файла (1 – чтение из текстового файла, 2 – запись в текстовый файл, 3 – чтение из типизированного файла, 4 – запись в типизированный файл). Возвращает она код 0 (успешное открытие) либо 1 (ошибка).

В (блок 1) создаются переменные: check (С-строка для хранения считанного расширения названия файла), letters (количество символов в названии файла), txt (С-строка для хранения эталона расширения «.txt»), bin (С-строка для хранения эталона расширения «.bin»), flg\_check (флаг для проверки совпадения расширения с эталоном).

Далее (блок 2) происходить вычленение последний 4 символов названия файла, предполагая, что эти символы и есть расширение. После, в зависимости от режима работы функции, в (блок 3) производится посимвольное сравнение расширения-эталона с расширением указанного файла. Если хотя бы один из символов не совпадёт, то поднимется флаг flg\_check и в последующей проверка этого флага (блок 4) произойдёт выход из функции с соответствующим сообщением. Если же флаг остался не поднятым, функция перейдёт в (блок 5), в котором путём использования оператора выбора по переменной mode выберется режим открытия файла. Если при этом ошибки не возникло, функция завершит свою работу.

2.4.16) Алгоритм функции check\_way приведён на рисунке 2.31:

Функция предназначена для проверки наличия указания диска, ведь если функция fwrite или fread не найдёт указанный файл, она создаст свой файл в указанном каталоге, если тот существует, либо в корневом каталоге, содержащем саму программу.

Исходя из этого можно заключить, что функция просто проверит наличие диска в названии файла (блок 2), и если он есть, то опустит проверяющий флаг flg\_first, перейдёт в проверку второго символа на наличие символа двоеточия «:» (блок 3), и в случае наличия которого опустит второй флаг flg\_second, а после всего проверит состояние обоих флагов, и в случае поднятия хотя бы одного из них программа уведомит пользователя о том, что файл создан в корневом каталоге с программой, после чего в обоих случаях вернёт управление вызвавшей её функции.

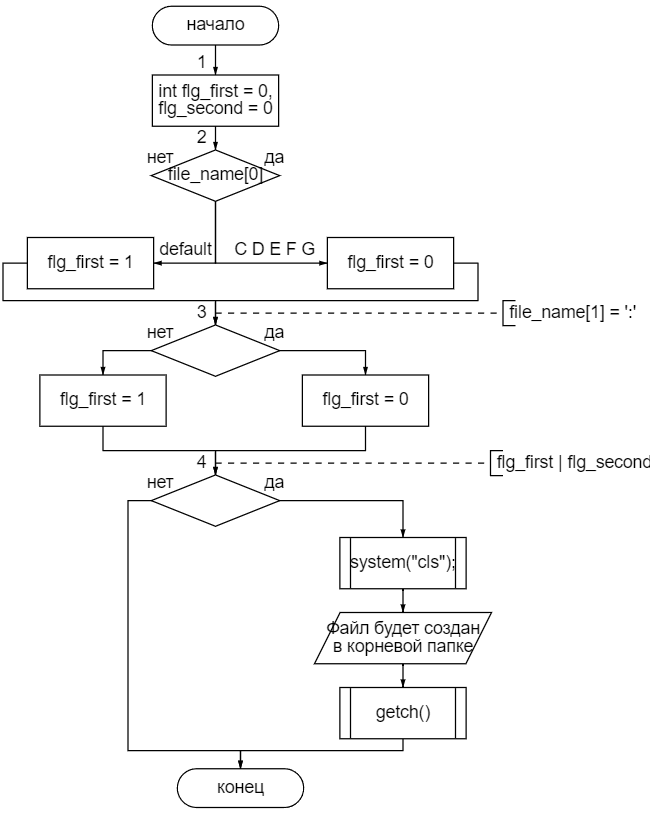


Рисунок 2.31– Схема функции check\_way()

2.4.17) Алгоритм функции free\_memory приведён на рисунке 2.32:

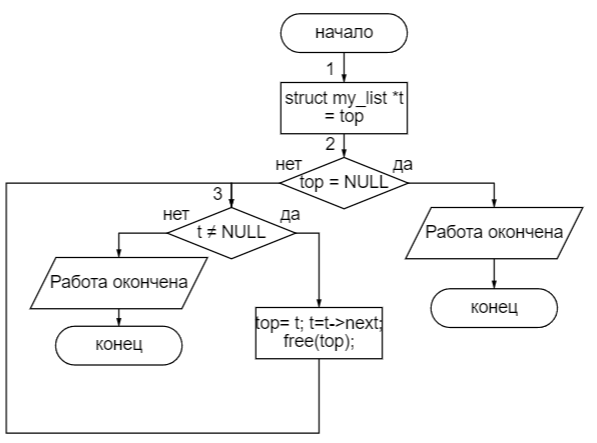


Рисунок 2.32– Схема функции free\_memory

Функция, как видно из названия, очищает динамическую память, используемую списком. Параметром она принимает указатель на вершину списка(top), а возвращать ничего не будет.

В (блок 1) создаётся вспомогательный указатель t для перемещения по списку.

Далее, в (блок 2) происходит проверка на пустоту списка, и если он пуст, то пользователя уведомляют о завершении работы с программой, и на этом она, собственно, и заканчивается. Если же список полон, то происходит переход в (блок 3), где в цикле, пока t не будет являться концом списка, будет происходить освобождение памяти функцией free. По выходе из цикла пользователя также уведомят о завершении работы с программой.

2.4.18) Алгоритм функции search приведён на рисунках 2.33-2.36:

Функция организует поиск записей в списке. Предоставляет возможность осуществить поиск по трём полям: номер завода, номер филиала и имя ответственного. Параметром принимает указатель на начало списка (top). Ничего не возвращает.

В (блок 1) происходи проверка списка функцией check\_list (см. п. 2.4.5), которая в случае его пустоты заканчивает выполнение функции.

Далее, в (блок 2) создаются необходимые переменные: t (указатель для перемещения по списку), pos (текущая позиция для выделения поля для поиска), page (текущая страница для вывода найденных записей), pages (всего страниц найденных записей), simbol (для хранения нажатия клавиши), i (для нумерации вывода записей), p\_count (счётчик для подсчёта записей при пролистывании страниц), choose (переменная поиска), choose\_n (С-строка поиска), p (счётчик для прохода к странице вывода), found (количество найденных записей), flg (флаг для проверки условия вывода записи).

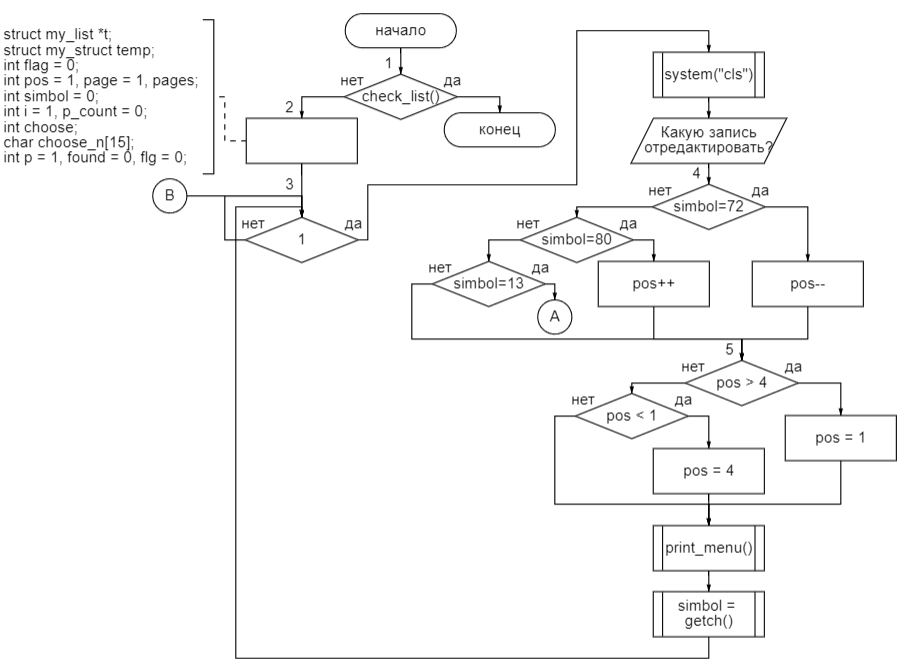


Рисунок 2.33– Схема функции search ветка начало

Далее происходит вход в бесконечный цикл для выбора поля поиска (блок 3), в котором будут выводится предложенные варианты поиска функцией print\_menu (см. п. 2.4.20), а также считываться нажатие клавиши, в зависимости от которого будет меняться переменная pos (блок 4), которая определяет, какое поле в данный момент нужно выделить (зависит от нажатия стрелок вверх-вниз). После проверки нажатия контролируется выход переменной pos за свои границы (блок 5).

Если была нажата клавиша enter, произойдёт переход в ветку А (см. далее).

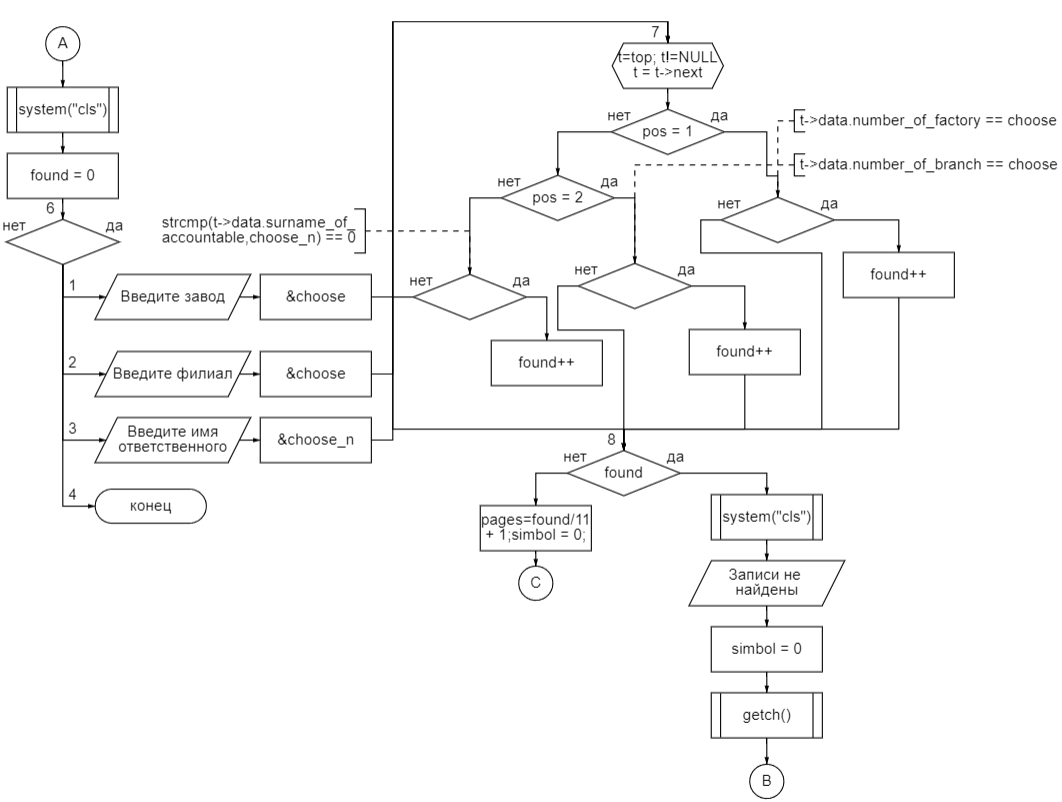


Рисунок 2.34– Схема функции search ветка A

Ветка реализует считывание параметра поиска операторе выбора (блок 6), после чего пробегается по списку на предмет наличия соответствующих записей (блок 7), и подсчитывает их количество в переменной found. Если записи не нашлись, на экран выводится соответствующее сообщение, и происходит возврат в меню выбора поля поиска (см. ветку B). В ином случае определяется количество страниц для вывода, и происходит переход в ветку С (см. далее).

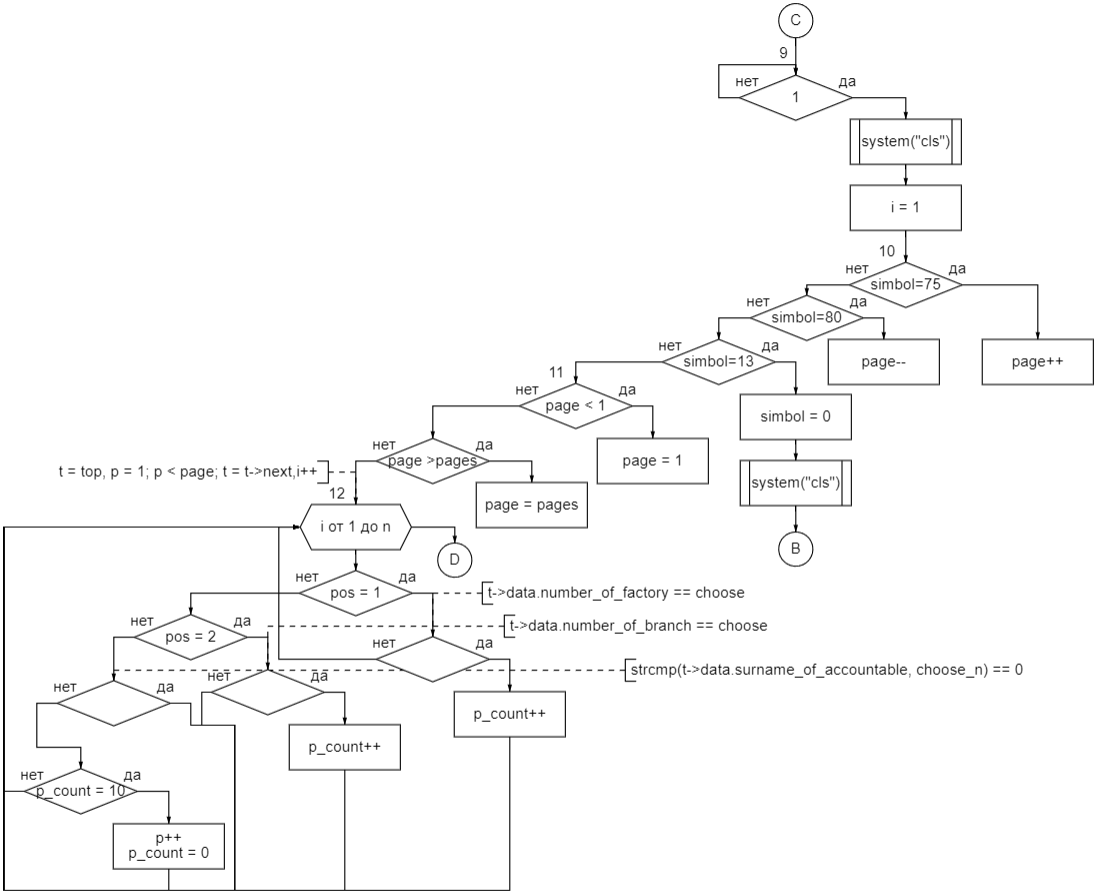


Рисунок 2.35– Схема функции search ветка C

В ветке происходит переход к записи, которая будет началом «текущей страницы» вывода.

Определяется эта страница переменной page, которая изменяется при нажатии стрелок «влево» и «вправо» (блок 10). После изменения этой переменной происходит контроль её выхода за границы (блок 11).

Далее, в (блок 12), производится сам переход к началу страницы вывода – по 10 раз в page циклов происходит шаг в списке. Счётчиком до 10 является переменная p\_count, которая увеличивается только тогда, когда текущая запись соответствует условию поиска.

Если же пользователь нажал «enter», произойдёт выход в меню вывода полей поиска (строка «Exit» выделяется, показывая, что выбор пользователя стоит на ней).

После прохождения к записи вывода происходит переход в ветку D (см. далее).

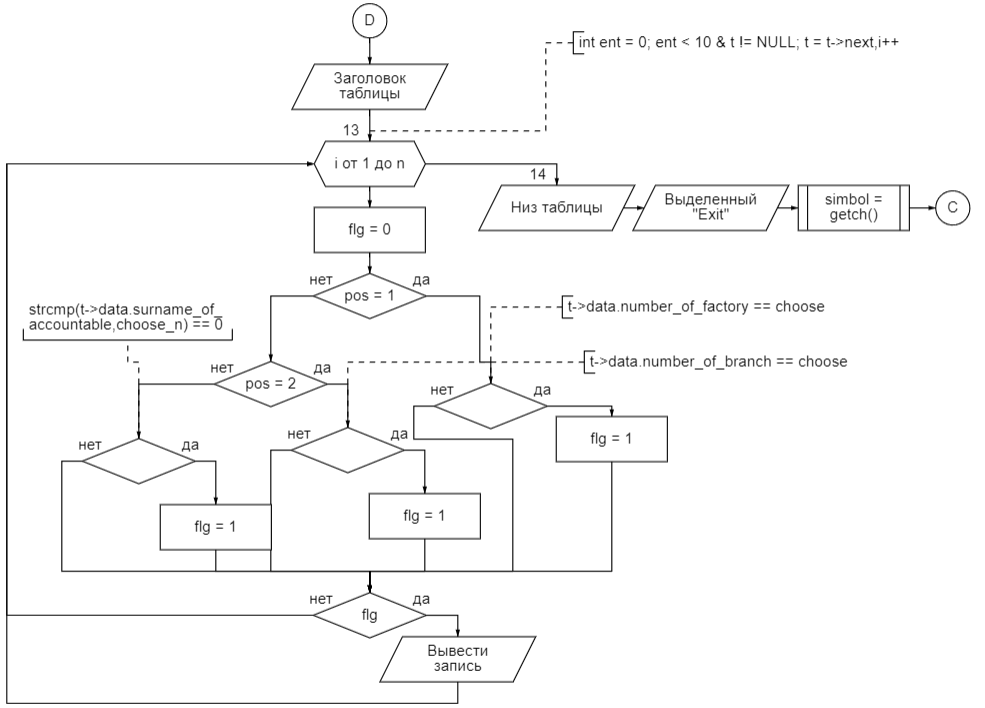


Рисунок 2.36– Схема функции search ветка D

Ветка реализует постраничный вывод на экран найденных в списке записей.

В (блок 13) производится пробег по списку с места, которое является началом выводимой страницы. Но выводятся не все записи подряд, а только те, которые удовлетворяют заданному условию. Так выводится 10 найденных записей, или, если это последняя страница, оставшиеся записи.

После нажатия «enter» происходит переход в начало программы, которая выведет поля для поиска записей.

2.4.19) Алгоритм функции get\_report приведён на рисунках 2.37-2.40:

Функция осуществляет формирование отчёта заданного типа. Типа отчёта есть два: вывод всех филиалов данного завода и подсчёт итоговых цифр (суммирование всех числовых полей) филиалов заданного завода.

Параметром передаётся только указатель на начало списка (top). Сама функция ничего не возвращает.

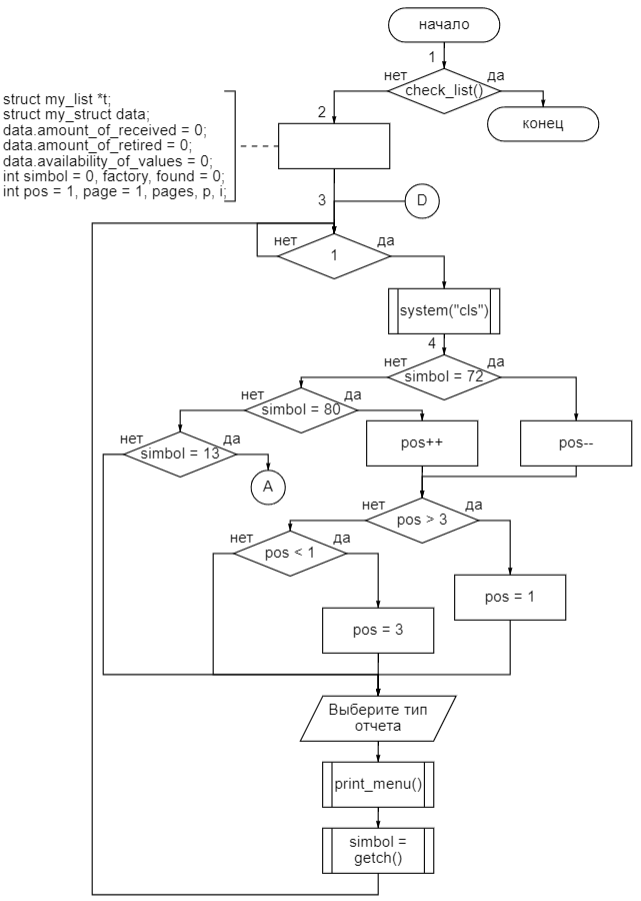


Рисунок 2.37– Схема функции get\_report ветка начало

Работа функции начинается с традиционной проверки списка на пустоту (блок 1) функцией check\_list (см. п. 2.4.4). Если список не пуст, работа функции продолжается.

Далее, в (блок 2), создаются необходимые переменные: t (указатель для перемещения по списку), data (информационная структура для формирования итоговых цифр), simbol (для считывания нажатой клавиши), factory (для задания завода поиска), found (для количества найденных записей), pos (текущая позиция выбора типа отчёта), page (текущая страница вывода найденных записей), pages (всего страниц найденных записей), p (счётчик для перехода к странице записей вывода), i (счётчик для нумерации записей вывода).

Далее происходит проверка нажатия клавиши (блок 4), где при нажатии стрелочек «вверх» и «вниз» меняется переменная pos, которая потом определит, какой тип отчета выбрал пользователь или какую строку нужно выделить при выводе. Если пользователь нажал клавишу «enter», происходит переход в ветку А (см. далее).

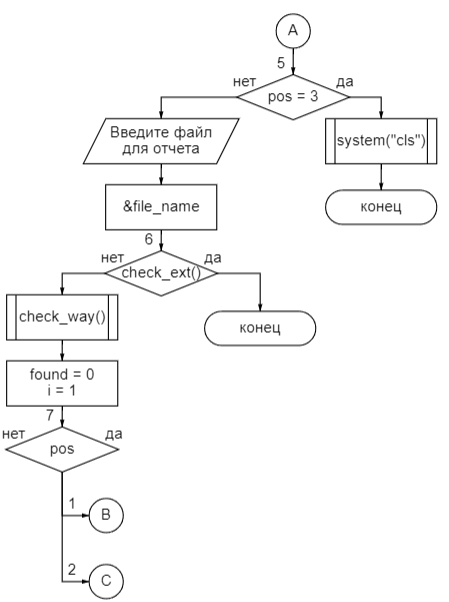


Рисунок 2.38– Схема функции get\_report ветка A

Ветка является началом формирования отчета. Сначала, (блок 5), проверяется, не выбрал ли пользователь вариант «Exit» (выход).

Если выбран другой вариант, пользователя просят ввести имя файла, в который будет выведен результат формирования отчёта. После ввода проверяется корректность ввода расширения файла – txt функцией check\_ext (см. п. 15). Далее проверяется наличие указания диска функцией check\_way (см. п. 16). И после, в операторе выбора (блок 7) в зависимости от того, какой тип отчёта выбран, происходит переход в ветки B (все филиалы) или C (итоговые цифры).

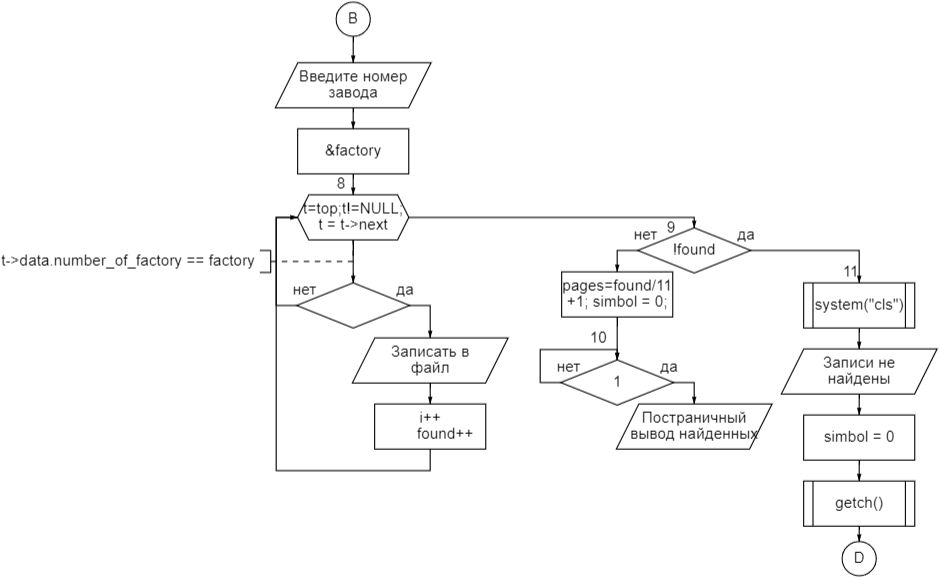


Рисунок 2.39– Схема функции get\_report ветка B

Ветка производит формирование отчёта первого типа – вывод всех филиалов данного завода. Сначала пользователь вводит имя завода, по которому произведётся поиск, а потом, в (блок 8), произведётся проход по списку с подсчётом и сразу выводом в файл отчёта подходящих записей. Далее проверяется переменная found, и, если она равна нулю, происходит выход с функции, потому что записи не найдены, в ином же случае происходит определение количества страниц для вывода найденных записей, и сам вывод происходит в бесконечном цикле (блок 10) по аналогии с предыдущими выводами.

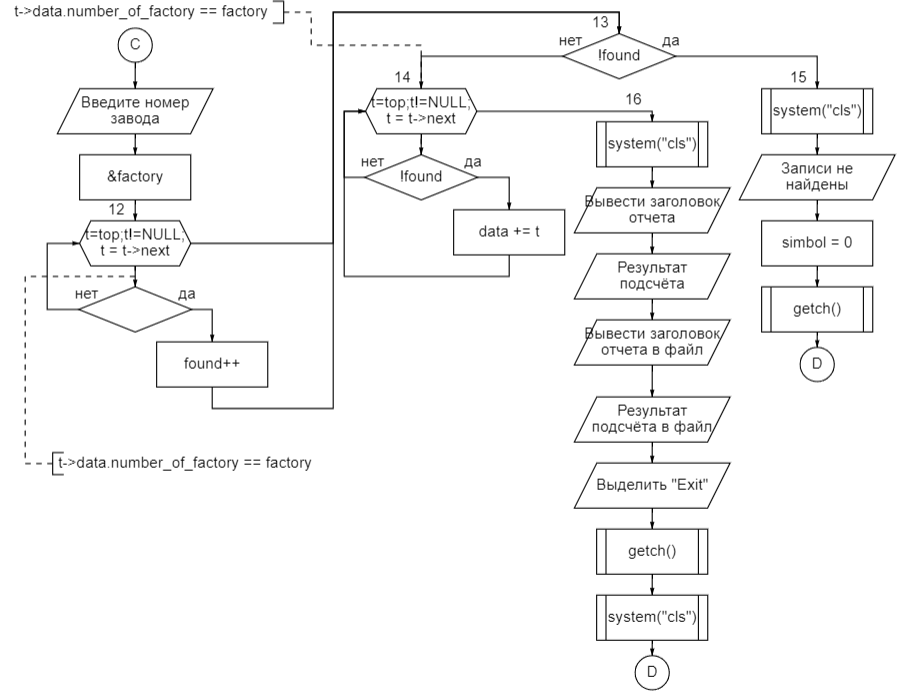


Рисунок 2.40– Схема функции get\_report ветка C

Ветка реализует формирование отчёта второго типа – подсчёт итоговых цифр по заданному заводу.

Пользователь вводит имя завода, и происходит проход по списку (блок 12) в поисках подходящей записи, если найдена хотя бы одна запись, цикл завершается и происходит новый проход по циклу с суммированием полей подходящих записей в структуру data.

Далее просто выводится табличка с полученными цифрами (блок 16), и работа функции заканчивается.

2.4.20) Алгоритм функции print\_menu приведён на рисунке 2.41:

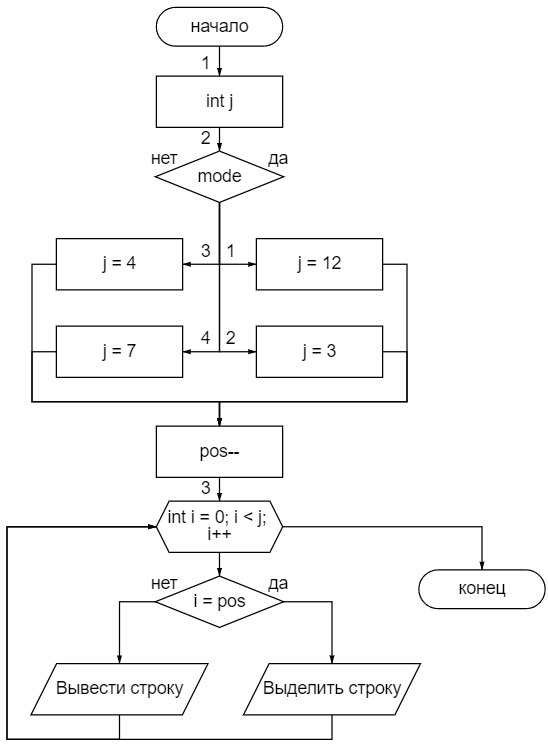


Рисунок 2.41– Схема функции print\_menu

Функция предназначена для вывода строк меню, выделяя одну, на которой стоит выбор пользователя. Принимает она массив команд на вывод (mass), позицию строки выделения (pos), а также режим работы (mode) – он определяет количество выводимых строк (в функцию передаются разные наборы строк).

Сначала в функции создаётся переменная j (количество выводимых строк).

После этого происходит выбор – сколько строк будет нужно вывести на экран, и это определяется в операторе выбора (блок 2) по параметру mode. Конечно, количество строк можно было бы передавать сразу параметром в функцию, но дабы сохранить читаемость кода и понимание того, что функция предназначена для нескольких режимов работы, эти режимы задаются номерами.

После этого происходит понижение переменной pos, т.к. нумерация в массиве осуществляется с 0, и в (блок 3) производится непосредственный вывод всех строк. Если номер очередной строки совпал с номером pos, то эта строка выделяется. После цикла функция завершает свою работу.

2.4.21) Алгоритм функции print\_entry приведён на рисунке 2.42:

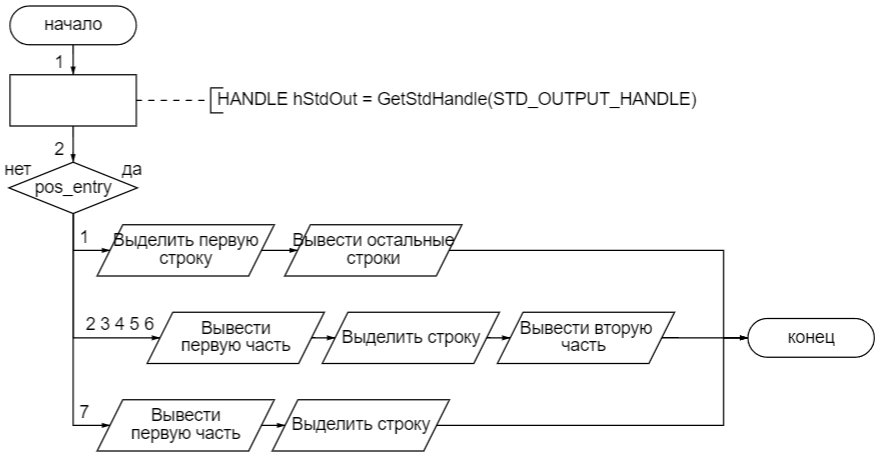


Рисунок 2.42– Схема функции print\_entry

Функция служит для выделения полей записи в функции корректировки correct (см.п. 2.4.9). Параметрами она принимает информационную структуру (data) и номер (pos\_entry) строки, а если быть точнее, то в контексте данной функции вариант вывода строк, который определяет, какую строку выделить. Функция ничего не возвращает.

В (блок 1) создаётся переменная для управления цветом консоли. Далее, с помощью оператора выбора по позиции вывода(pos\_entry) производится вывод полей записи, выделяя конкретное поле. Более функция ничего не делает.

2.4.22) Алгоритм функции print\_line приведён на рисунке 2.43:

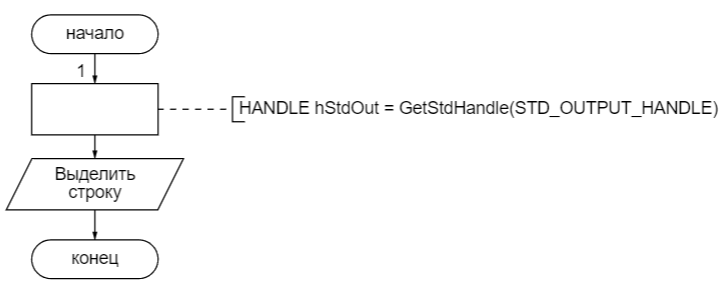


Рисунок 2.43– Схема функции print\_line

Функция принимает параметром строку, и выделяя, выводит её.

# **3 ВЫПОЛНЕНИЕ ПРОГРАММЫ**

**3.1 Условия выполнения программы**

**Аппаратные требования:**

Компьютер: Доступ к компьютеру с операционной системой,

способной выполнить программу на языке C++.

Клавиатура и мышь: Для ввода данных и взаимодействия с

программой.

**Программные требования:**

Компилятор C++: Необходим компилятор, способный компилировать

программы на языке C++ (Например: Microsoft Visual Studio)

Стандартная библиотека C++: Должна быть доступна для

использования стандартных функций и классов языка C++.

Операционная система: Программа должна быть совместима с

операционной системой, на которой она запускается (например, Windows).

Ресурсы компьютера (память, процессор) должны быть достаточными

для обработки данных, хранящихся в списке студентов.

Также, для корректного выполнения программы, необходимо, чтобы

введенные данные (например, файлы) соответствовали ожидаемому формату

программы.

**3.2 Загрузка и запуск программы**

Работа программы начинается с приветствия (рис. 3.1):

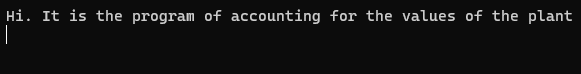


Рисунок 3.1 – Приветствие программы

И после нажатия любой клавиши выводится меню (рис. 3.2), в котором пользователь будет находиться всё время работы программы (до тех пор, пока не выберет вариант 12 – завершение работы с программой).

Взаимодействие с программой происходит посредством нажатия стрелок на клавиатуре с целью выбрать необходимое действие программы.

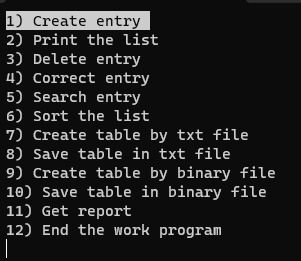


Рисунок 3.2 – Главное меню

Сначала требуется создать запись в списке (в ином случае нет смысла использовать другие функции), либо можно создать список с текстового/бинарного файла (рис. 3.3):

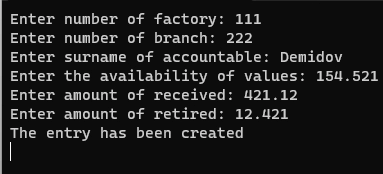


Рисунок 3.3 – Создание записи

Вывод созданной записи (рис. 3.4):

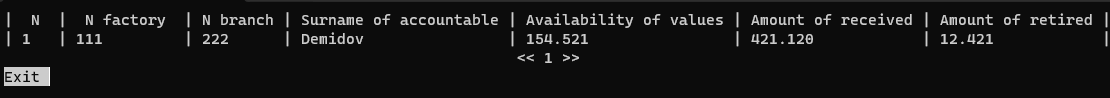


Рисунок 3.4 – Вывод записи

При выборе варианта 7 или 9 в случае, если список уже имеет записи, пользователя спросят, что делать с предыдущим списком – удалить, дополнить или выйти в меню (рис. 3.5):

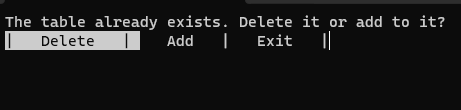


Рисунок 3.5 – Выбор варианта создания списка из файла

Далее вводится имя файла, и производится создание/дополнение списка (рис. 3.6):

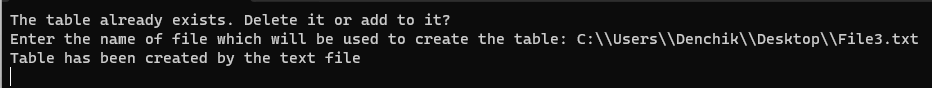


Рисунок 3.6 – Указание пути файла

Вывод на экран списка, созданного из файла (рис. 3.7):

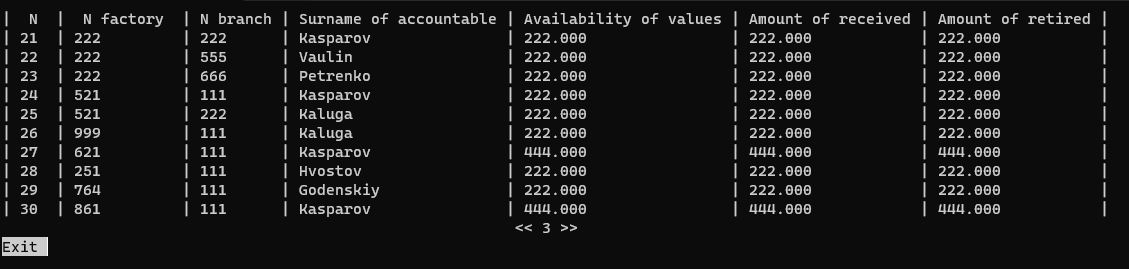


Рисунок 3.7 – Вывод списка из файла

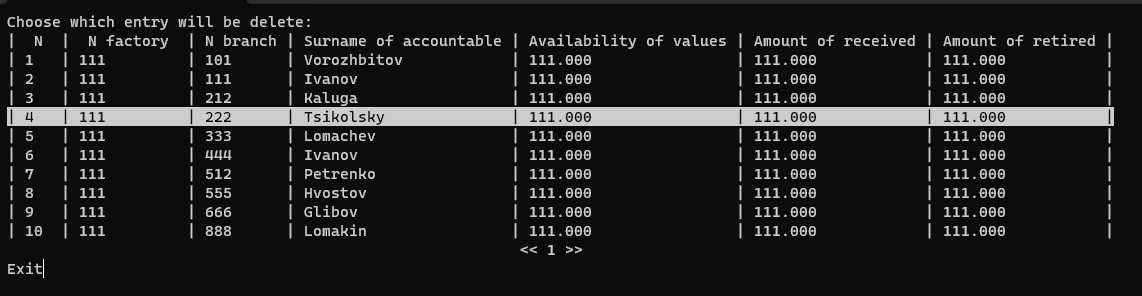
****

Рисунок 3.8 – Выбор записи на удаление

Удаление записи реализовано также посредством выбора записи на удаление, а после выбора требуется подтверждение на удаление (рис. 3.8-3.9):

****

Рисунок 3.9 – Подтверждение удаления

Корректировка точно так же производится с помощью выбора записи, после чего выводятся все поля записи с таким же выбором поля на корректировку (рис. 3.10):

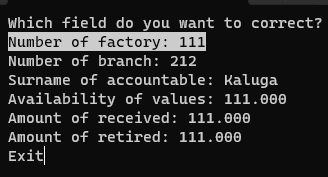
****

Рисунок 3.10 – Корректировка записи

В случае, если откорректированная запись является клоном уже существующей (имеет такой же номер завода и филиала) пользователя об этом уведомляют и отменяют корректировку (рис. 3.11):

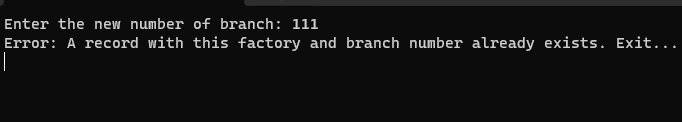
****

Рисунок 3.11 – Уведомление о существовании похожей записи

Поиск предлагается по нескольким полям (номер завода, филиала, и фамилия ответственного) (рис. 3.12-3.13):

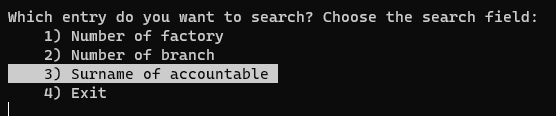
****

Рисунок 3.12 – Меню поиска

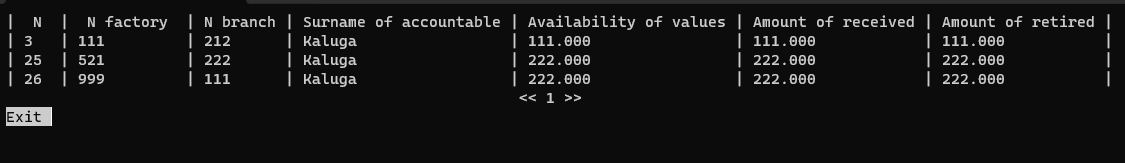
****

Рисунок 3.13 – Результат поиска по фамилии

Сортировка реализована с выбором обоих направлений и всех полей записей (рис. 3.14-3.15):

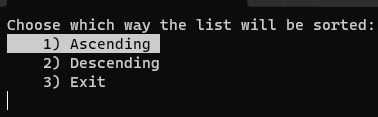
****

Рисунок 3.14 – Выбор направления сортировки

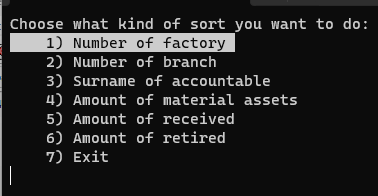
****

Рисунок 3.15 – Выбор поля сортировки

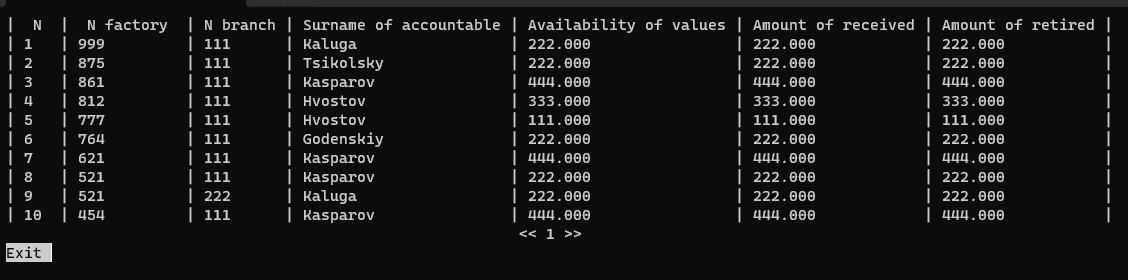
****

Рисунок 3.16 – Результат сортировки по убыванию номера завода

Сохранение списка возможно в два типа файла: текстовый(txt) и типизированный(bin). Пользователь просто вводит имя создаваемого или открываемого файла, и программа записывает туда весь список (рис. 3.17):

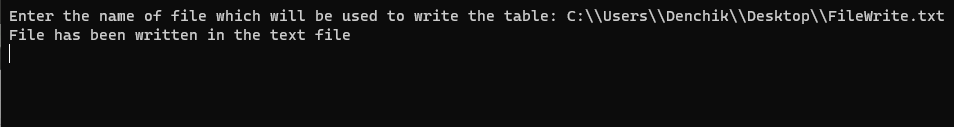
****

Рисунок 3.17 – Указание файла для сохранения списка

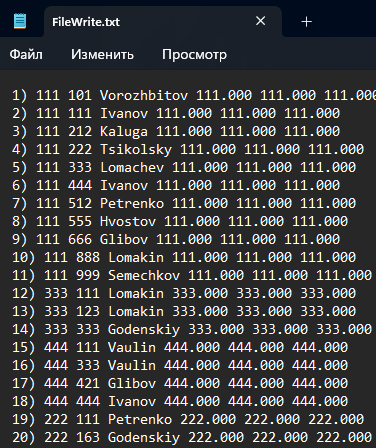
****

Рисунок 3.18 – Файл с записанным списком

Отчёт предлагается двух типов: цифры по всем филиалам указанного завода и итоговые цифры указанного завода (просуммированные поля всех его филиалов). Сам отчёт, кроме вывода на экран, записывается в прежде указанный пользователем файл (рис. 3.19-3.21):

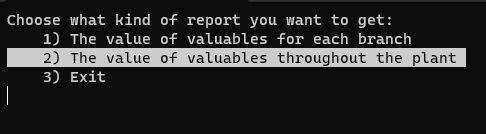
****

Рисунок 3.19 – Выбор типа отчёта

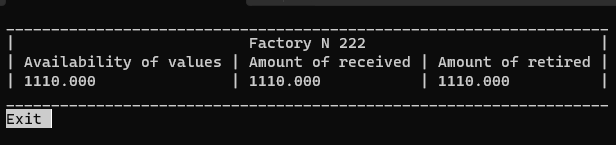
****

Рисунок 3.20 – Результат формирования отчёта по итоговым цифрам

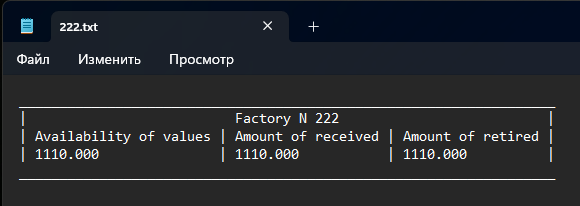
****

Рисунок 3.21 – Отчёт, записанный в файл

При вводе названий файла проверяется корректность введённого расширения (txt и bin) (рис. 3.22), а также наличие указания диска (если он не указан, пользователю уведомят о том, что файл создастся в корневой папке с программой) (рис. 3.23):

****

Рисунок 3.22 – Ошибка указания расширения файла

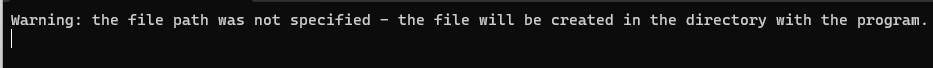
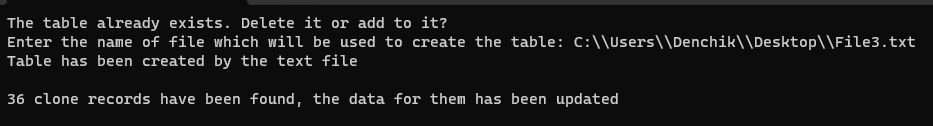
****

Рисунок 3.23 – Предупреждение не указания диска файла

Также создание записей производится с проверкой на наличие клонов в списке – если создаваемая запись совпадает номером завода и филиала с существующей записью, то данные для второй заменяются новой. При создании списка из файла работает тот же механизм, и пользователя уведомят о количестве найденных клонов (рис. 3.24):

****Рисунок 3.24 – Сообщение о заменённых клонах

При завершении работы с программой выводится прощальное сообщение (рис. 3.25):

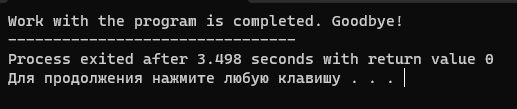
****

Рисунок 3.25 – Сообщение при завершении работы с программой

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данного проекта мы успешно справились с

задачами, направленными на создание программы для эффективного

управления реестром студентов. Реализованная функциональность включает

в себя широкий спектр действий, таких как добавление, исключение,

коррекция и поиск записей в списке студентов. Заметным достижением стало

внедрение возможности сохранения и извлечения данных из файла,

придавшее программе дополнительную гибкость.

Основным механизмом для работы с данными был выбран двусвязный

список, обеспечивающий высокую эффективность при выполнении операций

добавления, удаления и коррекции записей. Кроме того, этот список

обеспечивает двунаправленный доступ к данным, повышая тем самым

удобство использования.

Проект успешно завершен, и его функциональность проверена на

соответствие поставленным задачам. Основная цель проекта, а именно

создание программы для эффективного управления реестром студентов, была достигнута. Рекомендуем внедрить разработанный продукт в образовательных учреждениях для улучшения учета данных о студентах. Возможны доработки и расширения функционала в будущем, с возможным масштабированием в различных учебных заведениях России.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 - Солдатенко, И. С. Практическое введение в язык программирования Си : учебное пособие / И. С. Солдатенко, И. В. Попов. — Санкт-Петербург (дата обращения: 10.12.2022).

2 - Солдатенко, И. С. Практическое введение в язык программирования Си : учебное пособие для спо / И. С. Солдатенко, И. В. Попов. — Санкт-Петербург: Лань, 2021. — 132 с. — ISBN 978-5-8114-6925-3. (дата обращения: 10.12.2022).

3 - Перцев, И. В. Программирование на языке Си : учебно-методическое пособие / И. В. Перцев ; RU. — Новосибирск : СибГУТИ, 2022. — 106 с. — (дата обращения: 10.12.2022).

4 - Рацеев, С. М. Программирование на языке Си : учебное пособие для спо / С. М. Рацеев. — Санкт-Петербург : Лань, 2022. — 332 с. — ISBN 978-5-8114-8584-0.  (дата обращения: 10.12.2022).

5 - Гуркова, М. А. Программирование на языке Си: Практикум : учебное пособие / М. А. Гуркова, Э. Р. Резникова. — Москва : РУТ (МИИТ), 2020. — 70 с.

**ПРИЛОЖЕНИЕ A**

Листинг 1 – Код программы:

#include <windows.h>

#include <stdio.h>

#include <conio.h>

struct my\_struct{

int number\_of\_factory;

int number\_of\_branch;

char surname\_of\_accountable[15];

float availability\_of\_values;

float amount\_of\_received;

float amount\_of\_retired;

};

struct my\_list{

struct my\_struct data;

struct my\_list \*prev, \*next;

};

struct my\_struct create\_data\_main(); // Создание записи

int create\_data(char \*string); // вспом создание записи

struct my\_list \*create\_entry(struct my\_list \*top, struct my\_struct data); // Добавление записи

int check\_list(struct my\_list \*top); // Проверка пустоты файла

void print\_list(struct my\_list \*top, int pos, int page, int mode); // Печать списка

void print\_list\_main(struct my\_list \*top); // вспом печать списка

struct my\_list \*delete\_entry(struct my\_list \*top); // Удаление записи

void correct\_entry(struct my\_list \*top); // Корректировка записи

void sort(struct my\_list \*top, char \*mass1[], char \*mass2[]); // Сортировка списка

struct my\_list \*read\_txt(struct my\_list \*top); // Чтение с текстового файла

void write\_txt(struct my\_list \*top); // Запись в текстовый файл

struct my\_list \*read\_bin(struct my\_list \*top); // Чтение с бинарного файла

void write\_bin(struct my\_list \*top); // Запись в бинарный файл

int check\_ext(int mode); // проверка на расширение файла и его открытие

void check\_way(); // проверка наличия пути файла

void free\_memory(struct my\_list \*top); // Освобождение памяти

void search(struct my\_list \*top, char \*mass[4]); //Поиск

void get\_report(struct my\_list \*top, char \*mass\_report[3]); // Получение ведомостей

void print\_menu(char \*mass[12],int pos, int mode); // Подсвечивание меню(r = 1) и меню ведомостей(r = 2)

void print\_entry(struct my\_struct data,int pos\_entry);

void print\_line(char \*string); // Подсвечивание одной строки

int counter\_klon = 0;

int count\_of\_entries = 0;

int count\_of\_pages = 0;

char file\_name[260];

FILE \*file = NULL;

// C:\\Users\\Denchik\\Desktop\\File3.txt

int main(){

printf("Hi. It is the program of accounting for the values of the plant\n");

getch();

struct my\_struct entry;

struct my\_list \*top = NULL;

int pos = 1;

int simbol = 0;

char \*mass[] = {

"1) Create entry",

"2) Print the list",

"3) Delete entry",

"4) Correct entry",

"5) Search entry",

"6) Sort the list",

"7) Create table by txt file",

"8) Save table in txt file",

"9) Create table by binary file",

"10) Save table in binary file",

"11) Get report",

"12) End the work program"

};

char \*mass\_report[] = {

" 1) The value of valuables for each branch",

" 2) The value of valuables throughout the plant",

" 3) Exit"

};

char \*mass\_search[] = {

" 1) Number of factory",

" 2) Number of branch",

" 3) Surname of accountable",

" 4) Exit"

};

char \*mass1\_sort[] = {

" 1) Number of factory",

" 2) Number of branch",

" 3) Surname of accountable",

" 4) Amount of material assets",

" 5) Amount of received",

" 6) Amount of retired",

" 7) Exit"

};

char \*mass2\_sort[] = {

" 1) Ascending",

" 2) Descending",

" 3) Exit"

};

while(1){

system("cls");

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 13)

switch(pos){

case 1:

entry = create\_data\_main();

top = create\_entry(top,entry);

if(counter\_klon) {

printf("\n%d clone records have been found, the data for them has been updated",counter\_klon);

getch();

}

counter\_klon = 0;

break;

case 2: print\_list\_main(top); break;

case 3: top = delete\_entry(top); break;

case 4: correct\_entry(top); break;

case 5: search(top,mass\_search); break;

case 6: sort(top,mass1\_sort,mass2\_sort); break;

case 7: top = read\_txt(top); break;

case 8: write\_txt(top); break;

case 9: top = read\_bin(top); break;

case 10: write\_bin(top); break;

case 11: get\_report(top, mass\_report); break;

case 12:

free\_memory(top);

printf("Work with the program is completed. Goodbye!");

return 0;

}

system("cls");

if(pos > 12) pos = 1;

else if(pos < 1) pos = 12;

print\_menu(mass,pos,1);

simbol = getch();

}

}

struct my\_struct create\_data\_main(){

struct my\_struct data;

char word[10];

printf("Enter number of factory: ");

data.number\_of\_factory = create\_data("Number of factory");

printf("Enter number of branch: ");

data.number\_of\_branch = create\_data("Number of branch");

printf("Enter surname of accountable: ");

scanf("%s",data.surname\_of\_accountable);

printf("Enter the availability of values: ");

scanf("%s",word);

data.availability\_of\_values = atof(word);

printf("Enter amount of received: ");

scanf("%s",word);

data.amount\_of\_received = atof(word);

printf("Enter amount of retired: ");

scanf("%s",word);

data.amount\_of\_retired = atof(word);

printf("The entry has been created\n");

return data;

}

int create\_data(char \*string){

char word[10];

scanf("%s",word);

int x = atoi(word);

while(x > 1000 | x < 100){

printf("%s must be more than 100 and less than 1000 and not contain letters\n Enter again: ",string);

scanf("%s",word);

x = atoi(word);

system("cls");

}

return x;

}

struct my\_list \*create\_entry(struct my\_list \*top, struct my\_struct data){

struct my\_list \*t;

if(!top){

top = (struct my\_list\*)malloc(sizeof(struct my\_list));

top->data = data;

top->prev = NULL;

top->next = NULL;

count\_of\_entries++;

count\_of\_pages = count\_of\_entries/11 + 1;

return top;

}

for(t = top; t != NULL; t = t->next){

if(t->data.number\_of\_factory == data.number\_of\_factory){

if(t->data.number\_of\_branch == data.number\_of\_branch){

t->data = data;

counter\_klon++;

return top;

}

}

}

for(t = top; t != NULL; t = t->next){

if(t->data.number\_of\_factory == data.number\_of\_factory){

if(t->data.number\_of\_branch > data.number\_of\_branch){

if(t->prev != NULL){

t = t->prev;

t->next->prev = (struct my\_list\*)malloc(sizeof(struct my\_list));

t->next->prev->data = data;

t->next->prev->next = t->next;

t->next->prev->prev = t;

t->next = t->next->prev;

count\_of\_entries++;

count\_of\_pages = count\_of\_entries/11 +1;

return top;

}

t->prev = (struct my\_list\*)malloc(sizeof(struct my\_list));

t->prev->data = data;

t->prev->next = t;

t->prev->prev = NULL;

count\_of\_entries++;

count\_of\_pages = count\_of\_entries/11 +1;

return t->prev;

}

else if(t->next != NULL){

if( t->next->data.number\_of\_factory != data.number\_of\_factory){

t->next->prev = (struct my\_list\*)malloc(sizeof(struct my\_list));

t->next->prev->data = data;

t->next->prev->next = t->next;

t->next->prev->prev = t;

t->next = t->next->prev;

count\_of\_entries++;

count\_of\_pages = count\_of\_entries/11 +1;

return top;

}

}

}

}

for(t = top; t->next != NULL; t = t->next);

t->next = (struct my\_list\*)malloc(sizeof(struct my\_list));

t->next->data = data;

t->next->prev = t;

t->next->next = NULL;

count\_of\_entries++;

count\_of\_pages = count\_of\_entries/11 +1;

return top;

}

int check\_list(struct my\_list \*top){

if (!top){

printf("List is empty\n");

getch();

return 1;

}

else return 0;

}

void print\_list\_main(struct my\_list \*top){

if(check\_list(top)) return;

int i = 1;

int simbol = 0;

int page = 1;

while(1){

system("cls");

if(simbol == 75) page--;

else if(simbol == 77) page++;

if(simbol == 13) return;

if (page > count\_of\_pages) page = count\_of\_pages;

else if(page < 1) page = 1;

print\_list(top,0,page,0);

simbol = getch();

}

}

void print\_list(struct my\_list \*top, int pos, int page, int mode){

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

if(check\_list(top)) return;

struct my\_list \*t = top;

int i = 1;

printf("| N | N factory | N branch | Surname of accountable | Availability of values | Amount of received | Amount of retired |\n");

for(int p = 1; p < page; p++) for(int ent = 1; ent < 11; ent++, i++,t = t->next);

for(int ent = 1; ent < 11 & t != NULL; ent++, t = t->next, i++){

if(mode == 1 & ent == pos){

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("| %-3.d | %-11.d | %-8.d | %-22s | %-22.3f | %-18.3f | %-17.3f |",i,t->data.number\_of\_factory,t->data.number\_of\_branch,t->data.surname\_of\_accountable,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\n");

continue;

}

printf("| %-3.d | %-11.d | %-8.d | %-22s | %-22.3f | %-18.3f | %-17.3f |\n",i,t->data.number\_of\_factory,t->data.number\_of\_branch,t->data.surname\_of\_accountable,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

}

printf(" << %d >> \n",page);

if(!mode){

print\_line("Exit");

}

return;

}

void print\_line(char \*string){

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("%s ",string);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

}

struct my\_list \*delete\_entry(struct my\_list \*top){

struct my\_list \*t;

int pos = 1, page = 1;

int simbol = 0;

int choose\_del = 1;

int p;

if(check\_list(top)) return top;

while(1){

mark:

system("cls");

printf("Choose which entry will be delete: \n");

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 75) page--;

else if(simbol == 77) page++;

else if(simbol == 13) {

choose\_del = 1;

simbol = 0;

if(page < count\_of\_pages){

if(pos == 11) return top;

}

else{

if(pos == count\_of\_entries%10 + 1) return top;

}

while(1){

if(simbol == 75) choose\_del--;

else if(simbol == 77) choose\_del++;

else if(simbol == 13) {

if(choose\_del == 2){

simbol = 0;

goto mark;

}

if(pos == 1 & page == 1){

if(top->next == NULL){

free(top);

return NULL;

}

top = top->next;

free(top->prev);

goto end;

}

for(p = 1,t = top; p < page; p++) for(int ent = 1; ent < 11 & t != NULL; ent++, t = t->next);

for(int i = 1; i < pos; i++, t = t->next);

if(t->next == NULL){

t->prev->next = NULL;

free(t);

count\_of\_entries--;

count\_of\_pages = count\_of\_entries/11 +1;

top->prev = NULL;

system("cls");

printf("Entry has been deleted\n");

getch();

return top;

}

t->prev->next = t->next;

t->next->prev = t->prev;

free(t);

goto end;

}

if(choose\_del < 1) choose\_del = 1;

else if(choose\_del > 2) choose\_del = 2;

system("cls");

printf("Are you sure you want to delete this entry?\n");

if(choose\_del == 1){

print\_line("| YES |");

printf(" NO |");

}

else{

printf("| YES ");

print\_line("| NO |");

}

simbol = getch();

}

}

if (page > count\_of\_pages) page = count\_of\_pages;

else if(page < 1) page = 1;

if(page < count\_of\_pages){

if(pos > 11) pos = 1;

else if(pos < 1) pos = 11;

}

else {

if(pos > count\_of\_entries%10 + 1) pos = 1;

else if(pos < 1) pos = count\_of\_entries%10 + 1;

}

print\_list(top,pos,page,1);

if((page < count\_of\_pages & pos == 11) | (page == count\_of\_pages & pos == count\_of\_entries%10 + 1)){

print\_line("Exit");

}

else printf("Exit");

simbol = getch();

}

end:

count\_of\_entries--;

count\_of\_pages = count\_of\_entries/11 +1;

top->prev = NULL;

system("cls");

printf("Entry has been deleted\n");

getch();

return top;

}

void correct\_entry(struct my\_list \*top){

struct my\_list \*t, \*j;

struct my\_struct data;

char corrected[10];

int pos = 1, page = 1, pos\_entry = 1;

int simbol = 0, simbol\_entry = 0;

int flag = 0;

int counter = 1, counter2 = 1;

if(check\_list(top)) return;

while(1){

mark:

system("cls");

printf("Choose which entry will be correct: \n");

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 75) page--;

else if(simbol == 77) page++;

else if(simbol == 13) {

if(page < count\_of\_pages){

if(pos == 11) return;

}

else{

if(pos == count\_of\_entries%10 + 1) return;

}

t = top;

pos\_entry = 1;

simbol\_entry = 0;

counter = 1;

for(int p = 1; p < page; p++) for(int ent = 1; ent < 11; ent++, t = t->next, counter++);

for(int j = 1; j < pos; t = t->next, j++, counter++);

while(1){

system("cls");

counter2 = 1;

if(simbol\_entry == 72) pos\_entry--;

else if(simbol\_entry == 80) pos\_entry++;

else if(simbol\_entry == 13){

switch(pos\_entry){

case 1:

data = t->data;

printf("Enter the new number of factory: ");

data.number\_of\_factory = create\_data("Number of factory");

system("cls");

for(j = top; j != NULL; j = j->next, counter2++){

if((j->data.number\_of\_factory == data.number\_of\_factory) & (j->data.number\_of\_branch == data.number\_of\_branch) & (counter != counter2) ){

printf("Error: A record with this factory and branch number already exists. Exit...\n");

getch();

return;

}

}

t->data.number\_of\_factory = data.number\_of\_factory;

system("cls");

break;

case 2:

data = t->data;

printf("Enter the new number of branch: ");

data.number\_of\_branch = create\_data("Number of branch");

for(j = top; j != NULL; j = j->next,counter2++){

if( (j->data.number\_of\_factory == data.number\_of\_factory) & (j->data.number\_of\_branch == data.number\_of\_branch) & (counter != counter2) ){

printf("Error: A record with this factory and branch number already exists. Exit...\n");

getch();

return;

}

}

system("cls");

t->data.number\_of\_branch = data.number\_of\_branch;

break;

case 3:

printf("Enter the new surname of accountable: ");

scanf("%s",&t->data.surname\_of\_accountable);

system("cls");

break;

case 4:

printf("Enter the new availability of values: ");

scanf("%s",corrected);

t->data.availability\_of\_values = atof(corrected);

system("cls");

break;

case 5:

printf("Enter the new amount of received: ");

scanf("%s",corrected);

t->data.amount\_of\_received = atof(corrected);

system("cls");

break;

case 6:

printf("Enter the new amount of retired: ");

scanf("%s",corrected);

t->data.amount\_of\_retired = atof(corrected);

system("cls");

break;

case 7:

simbol = 0;

goto mark;

break;

}

}

if(pos\_entry > 7) pos\_entry = 1;

else if(pos\_entry < 1) pos\_entry = 7;

printf("Which field do you want to correct?\n");

print\_entry(t->data,pos\_entry);

simbol\_entry = getch();

}

}

if (page > count\_of\_pages) page = count\_of\_pages;

else if(page < 1) page = 1;

if(page < count\_of\_pages){

if(pos > 11) pos = 1;

if(pos < 1) pos = 11;

}

else {

if(pos > count\_of\_entries%10 + 1)pos = 1;

if(pos < 1) pos = count\_of\_entries%10 + 1;

}

print\_list(top,pos,page,1);

if((page < count\_of\_pages & pos == 11) | (page == count\_of\_pages & pos == count\_of\_entries%10 + 1)){

print\_line("Exit");

}

else printf("Exit");

simbol = getch();

}

}

void sort(struct my\_list \*top, char \*mass1[7], char \*mass2[]){

if(check\_list(top)) return ;

struct my\_list \*t, \*j;

struct my\_struct temp;

int flg;

int simbol = 0, simbol\_mode = 0;

int pos = 1, pos\_mode = 1;

int massage = 0;

while(1){

mark:

system("cls");

printf("Choose which way the list will be sorted: \n");

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 13) {

if(pos == 3) {

system("cls");

return ;

}

pos\_mode = 1;

simbol\_mode = 0;

while(1){

system("cls");

printf("Choose what kind of sort you want to do: \n");

if(simbol\_mode == 72) pos\_mode--;

else if(simbol\_mode == 80) pos\_mode++;

else if(simbol\_mode == 13){

if(!top->next){

printf("The list has only one entry\n");

return ;

}

for (t = top; t != NULL; t = t->next) {

for (j = top; j->next != NULL; j = j->next) {

flg = 0;

switch(pos\_mode){

case 1:

if(pos == 1) {

if(j->data.number\_of\_factory > j->next->data.number\_of\_factory) flg = 1;

}

else if(j->data.number\_of\_factory < j->next->data.number\_of\_factory) flg = 1;

break;

case 2:

if(pos == 1) {

if(j->data.number\_of\_branch > j->next->data.number\_of\_branch) flg = 1;

}

else if(j->data.number\_of\_branch < j->next->data.number\_of\_branch) flg = 1;

break;

case 3:

if(pos == 1) {

if(strcmp(j->data.surname\_of\_accountable,j->next->data.surname\_of\_accountable) > 0) flg = 1;

}

else if(strcmp(j->data.surname\_of\_accountable,j->next->data.surname\_of\_accountable) < 0) flg = 1;

break;

case 4:

if(pos == 1) {

if(j->data.availability\_of\_values > j->next->data.availability\_of\_values) flg = 1;

}

else if(j->data.availability\_of\_values < j->next->data.availability\_of\_values) flg = 1;

break;

case 5:

if(pos == 1) {

if(j->data.amount\_of\_received > j->next->data.amount\_of\_received) flg = 1;

}

else if(j->data.amount\_of\_received < j->next->data.amount\_of\_received) flg = 1;

break;

case 6:

if(pos == 1) {

if(j->data.amount\_of\_retired > j->next->data.amount\_of\_retired) flg = 1;

}

else if(j->data.amount\_of\_retired < j->next->data.amount\_of\_retired) flg = 1;

break;

case 7:

simbol = 0;

goto mark;

break;

}

if (flg) {

temp = j->data;

j->data = j->next->data;

j->next->data = temp;

massage = 1;

}

}

}

system("cls");

if(massage) printf("The list was sorted\n");

else printf("List is already sorted\n");

getch();

return ;

}

if(pos\_mode > 7) pos\_mode = 1;

else if(pos\_mode < 1) pos\_mode = 7;

print\_menu(mass1,pos\_mode,4);

simbol\_mode = getch();

}

}

if(pos > 3) pos = 1;

else if(pos < 1) pos = 3;

print\_menu(mass2,pos,2);

simbol = getch();

}

}

struct my\_list \*read\_txt(struct my\_list \*top){

struct my\_struct entry;

int n = 0, flg = 0, count = 0, flg\_read = 0;

char word1[4],word2[4],word3[10],word4[10],word5[10];

int skobka;

int pos = 1, simbol = 0;

if(top) {

while(1){

system("cls");

printf("The table already exists. Delete it or add to it?\n");

if(simbol == 75) pos--;

else if(simbol == 77) pos++;

else if(simbol == 13) {

if(pos == 1){

free\_memory(top);

top = NULL;

break;

}

else if(pos == 2){

break;

}

else {

system("cls");

return top;

}

}

if(pos > 3) pos = 3;

else if(pos < 1) pos = 1;

if(pos == 1){

print\_line("| Delete |");

printf(" Add | Exit |");

}

else if(pos == 2){

printf("| Delete ");

print\_line("| Add |");

printf(" Exit |");

}

else{

printf("| Delete | Add ");

print\_line("| Exit |");

}

simbol = getch();

system("cls");

}

}

printf("Enter the name of file which will be used to create the table: ");

scanf("%s",file\_name);

if(check\_ext(1)) return top;

while(1){

flg = 0;

n = fscanf(file,"%s %s %s %s %s %s %s", &skobka,word1, word2, entry.surname\_of\_accountable, word3, word4, word5);

if (n == EOF) break;

flg\_read = 1;

entry.number\_of\_factory = atoi(word1);

entry.number\_of\_branch = atoi(word2);

if(entry.number\_of\_factory > 1000 | entry.number\_of\_factory < 100) flg = 1;

else if(entry.number\_of\_branch > 1000 | entry.number\_of\_branch < 100) flg = 1;

if(flg) {

count++;

continue;

}

entry.availability\_of\_values = atof(word3);

entry.amount\_of\_received = atof(word4);

entry.amount\_of\_retired = atof(word5);

top = create\_entry(top,entry);

}

if(flg\_read){

printf("Table has been created by the text file\n");

if(count) printf("%d entries were not add to the list because they have incorrect data",count);

if(counter\_klon) printf("\n%d clone records have been found, the data for them has been updated",counter\_klon);

}

else printf("The file is empty\n");

counter\_klon = 0;

getch();

fclose(file);

return top;

}

void write\_txt(struct my\_list \*top){

struct my\_list \*t;

int i = 1;

if(check\_list(top)) return;

printf("Enter the name of file which will be used to write the table: ");

scanf("%s",file\_name);

if(check\_ext(2)) return;

check\_way();

for(t=top; t != NULL; t = t->next,i++){

fprintf(file, "%d) %d %d %s %.3f %.3f %.3f\n",i,t->data.number\_of\_factory,t->data.number\_of\_branch,t->data.surname\_of\_accountable,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

}

printf("File has been written in the text file\n");

//if(!(strlen(file\_name)-4)) printf(" and placed in the app folder");

getch();

fclose(file);

return;

}

struct my\_list \*read\_bin(struct my\_list \*top){

struct my\_struct entry;

int n = 0,flg\_read = 0, count = 0;

int simbol = 0, simbol\_mode = 0;

int pos = 1, pos\_mode = 1;

if(top) {

while(1){

system("cls");

printf("The table already exists. Delete it or add to it?\n");

if(simbol == 75) pos--;

else if(simbol == 77) pos++;

else if(simbol == 13) {

if(pos == 1){

free\_memory(top);

top = NULL;

break;

}

else if(pos == 2){

break;

}

else {

system("cls");

return top;

}

}

if(pos > 3) pos = 3;

else if(pos < 1) pos = 1;

if(pos == 1){

print\_line("| Delete |");

printf(" Add | Exit |");

}

else if(pos == 2){

printf("| Delete ");

print\_line("| Add |");

printf(" Exit |");

}

else{

printf("| Delete | Add ");

print\_line("| Exit |");

}

simbol = getch();

system("cls");

}

}

printf("Enter the name of file which will be used to create the table: ");

scanf("%s",file\_name);

if (check\_ext(3))return top;

while(1){

n = fread(&entry,sizeof(struct my\_struct),1,file);

if (n < 1) break;

top = create\_entry(top,entry);

flg\_read = 1;

}

if(flg\_read) printf("The table was created by binary file\n");

else printf("The file is empty\n");

getch();

fclose(file);

return top;

}

void write\_bin(struct my\_list \*top){

struct my\_list \*t;

if(check\_list(top)) return;

printf("Enter the name of file which will be used to write the table: ");

scanf("%s",file\_name);

if (check\_ext(4)) return;

check\_way();

for(t=top; t != NULL; t = t->next){

fwrite(&t->data,sizeof(struct my\_struct),1,file);

}

printf("File has been written in the binary file\n");

getch();

fclose(file);

return;

}

int check\_ext(int mode){

char check[4];

int letters = strlen(file\_name);

char txt[] = ".txt";

char bin[] = ".bin";

int flg\_check = 0;

for(int i = 4; i > 0 ; i--) check[4-i] = file\_name[letters - i];

if(mode <= 2) {

for(int i = 0; i < 4; i++){

if(txt[i] != check[i]) flg\_check = 1;

}

}

else {

for(int i = 0; i < 4; i++){

if(bin[i] != check[i]) flg\_check = 1;

}

}

if(flg\_check){

system("cls");

if(mode <= 2) printf("Error: The file extension does not match \"%.4s\"\n",txt);

else printf("Error: The file extension does not match \"%.4s\"\n",bin);

getch();

return 1;

}

fclose(file);

switch(mode){

case 1: file = fopen(file\_name,"rt"); break;

case 2: file = fopen(file\_name,"wt"); break;

case 3: file = fopen(file\_name,"rb"); break;

case 4: file = fopen(file\_name,"wb"); break;

}

if(!file){

printf("Error: File opening error\nThe system probably can't find the file.\n");

getch();

return 1;

}

return 0;

}

void check\_way(){

int flg = 0;

switch(file\_name[0]){

case 'C': case 'D': case 'E': case 'F': case 'G': break;

default: flg = 1; break;

}

if(file\_name[1] == ':') flg = 0;

else flg = 1;

if(flg){

system("cls");

printf("Warning: the file path was not specified - the file will be created in the directory with the program.\n");

getch();

return;

}

return;

}

void free\_memory(struct my\_list \*top){

struct my\_list \*t = top;

if(!top){

return;

}

while (t != NULL){

top = t;

t = t->next;

free(top);

count\_of\_entries--;

count\_of\_pages = count\_of\_entries/11 + 1;

}

return;

}

void search(struct my\_list \*top, char \*mass[4]){

struct my\_list \*t;

if(check\_list(top)) return;

int pos = 1, page = 1, pages;

int simbol = 0;

int i = 1, p\_count = 0;

int choose;

char choose\_n[15];

int p = 1, found = 0, flg = 0;

while(1){

system("cls");

mark:

printf("Which entry do you want to search? Choose the search field: \n");

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 13) {

system("cls");

found = 0;

switch(pos){

case 1:

printf("Enter the number of factory: ");

scanf("%d",&choose);

break;

case 2:

printf("Enter the number of branch: ");

scanf("%d",&choose);

break;

case 3:

printf("Enter the surname of accountable: ");

scanf("%s",choose\_n);

break;

case 4: return;

}

for(t = top; t != NULL; t = t->next) {

if(pos == 1) {

if(t->data.number\_of\_factory == choose) found++;

}

else if(pos == 2){

if(t->data.number\_of\_branch == choose) found++;

}

else {

if(strcmp(t->data.surname\_of\_accountable, choose\_n) == 0) found++;

}

}

if(!found){

system("cls");

printf("Entries were not found\n");

simbol = 0;

getch();

goto mark;

}

pages = found/11 + 1;

simbol = 0;

while(1){

system("cls");

i = 1;

if(simbol == 75) page--;

else if(simbol == 77) page++;

else if(simbol == 13){

simbol = 0;

system("cls");

goto mark;

}

if(page < 1) page = 1;

else if(page > pages) page = pages;

for(t = top, p = 1; p < page; t = t->next,i++){

if (pos == 1){

if(t->data.number\_of\_factory == choose) p\_count++;

}

else if(pos == 2){

if(t->data.number\_of\_branch == choose) p\_count++;

}

else {

if(strcmp(t->data.surname\_of\_accountable, choose\_n) == 0) p\_count++;

}

if (p\_count == 10) {

p++;

p\_count = 0;

}

}

printf("| N | N factory | N branch | Surname of accountable | Availability of values | Amount of received | Amount of retired |\n");

for(int ent = 0; ent < 10 & t != NULL; t = t->next,i++){

flg = 0;

if (pos == 1){

if(t->data.number\_of\_factory == choose) flg = 1;

}

else if(pos == 2){

if(t->data.number\_of\_branch == choose) flg = 1;

}

else {

if(strcmp(t->data.surname\_of\_accountable, choose\_n) == 0) flg = 1;

}

if(flg){

printf("| %-3.d | %-11.d | %-8.d | %-22s | %-22.3f | %-18.3f | %-17.3f |\n",i,t->data.number\_of\_factory,t->data.number\_of\_branch,t->data.surname\_of\_accountable,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

ent++;

}

}

printf(" << %d >> \n",page);

print\_line("Exit");

simbol = getch();

}

}

if(pos > 4) pos = 1;

else if(pos < 1) pos = 4;

print\_menu(mass,pos,3);

simbol = getch();

}

}

void get\_report(struct my\_list \*top, char \*mass\_report[3]){

if(check\_list(top)) return ;

struct my\_list \*t;

struct my\_struct data;

data.amount\_of\_received = 0;

data.amount\_of\_retired = 0;

data.availability\_of\_values = 0;

int simbol = 0, factory, found = 0;

int pos = 1, page = 1, pages, p; // f - флаг для определения наличия данного завода/филиала

while(1){

system("cls");

mark:

if(simbol == 72) pos--;

else if(simbol == 80) pos++;

else if(simbol == 13) {

if(pos != 3){

printf("Enter the name of the file that will be used to output the statements: ");

scanf("%s",file\_name);

int check = check\_ext(2);

//check\_way();

if(check) return;

}

found = 0;

int i = 1;

switch(pos){

case 1:

printf("Enter the number of factory: ");

scanf("%d",&factory);

fprintf(file,"| N | N branch | Availability of values | Amount of received | Amount of retired |\n");

for(t = top; t != NULL; t = t->next){

if (t->data.number\_of\_factory == factory) {

fprintf(file, "| %-3.d | %-8.d | %-22.3f | %-18.3f | %-17.3f |\n",i,t->data.number\_of\_branch,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

i++;

found++;

}

}

if(!found){

system("cls");

printf("The factory was not found\n");

getch();

simbol = 0;

goto mark;

}

pages = found/11 + 1;

simbol = 0;

while(1){

i = 1;

system("cls");

if(simbol == 75) page--;

else if(simbol == 77) page++;

else if(simbol == 13){

simbol = 0;

system("cls");

goto mark;

}

if(page < 1) page = 1;

else if(page > pages) page = pages;

int p\_ent = 0;

for(t = top, p = 1; p < page; t = t->next) {

if(t->data.number\_of\_factory == factory) p\_ent++;

if (p\_ent == 10) {

p++;

p\_ent = 0;

}

}

printf("| N | N branch | Availability of values | Amount of received | Amount of retired |\n");

for(int ent = 0; ent < 10 & t != NULL; t = t->next){

if(t->data.number\_of\_factory == factory){

printf("| %-3.d | %-8.d | %-22.3f | %-18.3f | %-17.3f |\n",i,t->data.number\_of\_branch,t->data.availability\_of\_values,t->data.amount\_of\_received,t->data.amount\_of\_retired);

ent++;

i++;

}

}

printf(" << %d >> \n",page);

print\_line("Exit");

simbol = getch();

}

break;

case 2:

printf("Enter the number of factory: ");

scanf("%d",&factory);

for(t = top; t != NULL; t = t->next){

if (t->data.number\_of\_factory == factory) {

found++;

break;

}

}

if(!found){

system("cls");

printf("The factory was not found\n");

getch();

simbol = 0;

goto mark;

return;

}

for(t = top; t != NULL;t = t->next)

if(t->data.number\_of\_factory == factory){

data.availability\_of\_values += t->data.availability\_of\_values;

data.amount\_of\_received += t->data.amount\_of\_received;

data.amount\_of\_retired += t->data.amount\_of\_retired;

}

system("cls");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| Factory N %d |\n| Availability of values | Amount of received | Amount of retired |\n",factory);

printf("| %-22.3f | %-18.3f | %-17.3f |\n",data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

fprintf(file,"| Factory N %d |\n| Availability of values | Amount of received | Amount of retired |\n",factory);

fprintf(file,"| %-22.3f | %-18.3f | %-17.3f |\n",data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

fprintf(file,"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

print\_line("Exit");

getch();

system("cls");

break;

case 3:

system("cls");

return;

}

}

if(pos > 3) pos = 1;

else if(pos < 1) pos = 3;

printf("Choose what kind of report you want to get: \n");

print\_menu(mass\_report,pos,2);

simbol = getch();

}

}

void print\_menu(char \*mass[],int pos, int mode){

int j;

switch(mode){

case 1: j = 12; break;

case 2: j = 3; break;

case 3: j = 4; break;

case 4: j = 7; break;

}

pos--;

for(int i = 0; i < j; i++){

if(i == pos) {

print\_line(mass[i]);

printf("\n");

continue;

}

printf("%s\n", mass[i]);

}

return;

}

void print\_entry(struct my\_struct data,int pos\_entry){

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

switch(pos\_entry){

case 1:

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Number of factory: %d", data.number\_of\_factory);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nNumber of branch: %d\nSurname of accountable: %s\nAvailability of values: %.3f\nAmount of received: %.3f\nAmount of retired: %.3f\nExit",data.number\_of\_branch,data.surname\_of\_accountable,data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

break;

case 2:

printf("Number of factory: %d\n", data.number\_of\_factory);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Number of branch: %d", data.number\_of\_branch);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nSurname of accountable: %s\nAvailability of values: %.3f\nAmount of received: %.3f\nAmount of retired: %.3f\nExit",data.surname\_of\_accountable,data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

break;

case 3:

printf("Number of factory: %d\nNumber of branch: %d\n", data.number\_of\_factory,data.number\_of\_branch);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Surname of accountable: %s", data.surname\_of\_accountable);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nAvailability of values: %.3f\nAmount of received: %.3f\nAmount of retired: %.3f\nExit",data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

break;

case 4:

printf("Number of factory: %d\nNumber of branch: %d\nSurname of accountable: %s\n", data.number\_of\_factory,data.number\_of\_branch,data.surname\_of\_accountable);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Availability of values: %.3f", data.availability\_of\_values);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nAmount of received: %.3f\nAmount of retired: %.3f\nExit",data.amount\_of\_received,data.amount\_of\_retired);

break;

case 5:

printf("Number of factory: %d\nNumber of branch: %d\nSurname of accountable: %s\nAvailability of values: %.3f\n", data.number\_of\_factory,data.number\_of\_branch,data.surname\_of\_accountable,data.availability\_of\_values);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Amount of received: %.3f", data.amount\_of\_received);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nAmount of retired: %.3f\nExit",data.amount\_of\_retired);

break;

case 6:

printf("Number of factory: %d\nNumber of branch: %d\nSurname of accountable: %s\nAvailability of values: %.3f\nAmount of received: %.3f\n", data.number\_of\_factory,data.number\_of\_branch,data.surname\_of\_accountable,data.availability\_of\_values,data.amount\_of\_received);

SetConsoleTextAttribute(hStdOut,BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_BLUE);

printf("Amount of retired: %.3f", data.amount\_of\_retired);

SetConsoleTextAttribute(hStdOut,FOREGROUND\_GREEN|FOREGROUND\_RED|FOREGROUND\_BLUE);

printf("\nExit");

break;

case 7:

printf("Number of factory: %d\nNumber of branch: %d\nSurname of accountable: %s\nAvailability of values: %.3f\nAmount of received: %.3f\nAmount of retired: %.3f\n", data.number\_of\_factory,data.number\_of\_branch,data.surname\_of\_accountable,data.availability\_of\_values,data.amount\_of\_received,data.amount\_of\_retired);

print\_line("Exit");

break;

}

return;

}