Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Структура списка изделий кондитерской фабрики

Студент: гр. 321702 Казаченко В.А.

Руководитель: Семижон Е.А.

Минск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc166757283)

[СТРУКТУРЫ И ФАЙЛЫ 4](#_Toc166757284)

[**Классификация структур данных** 4](#_Toc166757285)

[**Основные функции структур данных** 5](#_Toc166757286)

[АЛГОРИТМЫ СОРТИРОВКИ 6](#_Toc166757287)

[**Сортировка методом "пузырька" (простого обмена)** 6](#_Toc166757288)

[**Сортировка методом простого выбора (SearchZnuch)** 7](#_Toc166757289)

[**Сортировка методом Quicksort** 10](#_Toc166757291)

[АЛГОРИТМЫ ПОИСКА 12](#_Toc166757292)

[**Последовательный поиск** 12](#_Toc166757293)

[**Поиск делением пополам** 14](#_Toc166757294)

[ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ: 15](#_Toc166757296)

[**1.** **Функция для поиска выбором searchZnuch().** 16](#_Toc166757297)

[**2.** **Функция создания(сохранения в) файла saveDatatoFile().** 18](#_Toc166757298)

[**3.** **Функция для загрузки данных readData ().** 19](#_Toc166757299)

[**4.** **Функция добавления данных в файл ADD** **().** 20](#_Toc166757300)

[**5.** **Функция для просмотра файла printInfo().** 21](#_Toc166757301)

[**6.** **Функция проверки на дату getValidDate().** 22](#_Toc166757302)

[**7.** **Функция линейного поиска searchDiapaz ().** 23](#_Toc166757303)

[**8.** **Функция для проверки на високосный год isLeapYear().** 24](#_Toc166757304)

[**9.** **Функция сортировки (quicksort) sortVozrastanie().** 24](#_Toc166757305)

[**10. Функция сортировки методом прямого выбора sortUb().** 26](#_Toc166757306)

[**11.** **Функция для расчета стажа сотрудников calculateExperience().** 27](#_Toc166757307)

[**12.** **Функция удаления элементов Remove\_BY\_INDEX().** 28](#_Toc166757308)

[**13.** **Функция проверки даты isValidDate().** 27](#_Toc166757307)

[**14.** **Основная функция main().** 28](#_Toc166757308)

[ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ 29](#_Toc166757309)

[ВЫВОД: 30](#_Toc166757310)

[ПРИЛОЖЕНИЕ А: 31](#_Toc166757311)

[ПРИЛОЖЕНИЕ В: 39](#_Toc166757312)

[СПИСОК ЛИТЕРАТУРЫ: 40](#_Toc166757313)

# 

# 

# **ВВЕДЕНИЕ**

В современном мире информация играет ключевую роль как стратегический ресурс. Данные стали ценным активом, и умение правильно работать с ними стало необходимым для успешного развития бизнеса, науки, образования и других сфер жизни. Современные технологии позволяют создавать, хранить и обрабатывать большие объемы информации быстро и эффективно. Анализ данных, машинное обучение и искусственный интеллект помогают извлечь ценные знания и принимать обоснованные решения.

Информатизация, высокий уровень использования информации и технологий, способствуют конкурентоспособности предприятий и эффективному управлению в различных сферах общественной жизни. Автоматизация процессов, улучшение коммуникации, повышение качества услуг и товаров, а также оптимизация ресурсов становятся возможными благодаря информатизации.

Выбор интегрированной среды разработки (IDE) также играет важную роль в разработке программного обеспечения. Примером такой IDE является CLion, разрабатываемая компанией JetBrains. CLion позволяет создавать проекты и писать код на различных языках программирования, таких как C, C++, Python и Assembly. Она обеспечивает отладку прошивок и внутрипроцессорных программ, а также предоставляет возможность совместной работы и ревью кода в реальном времени. CLion обладает функциями, которые помогают понять сложные фрагменты кода, показывая выведенные типы, сигнатуры функций и документацию. Она также обеспечивает удобную навигацию по коду и поиск использований символов в проектах любого размера.

# 

# **СТРУКТУРЫ И ФАЙЛЫ**

В информатике структура данных — программная единица, позволяющая хранить и обрабатывать данные, а также обеспечивающая их эффективное использование. Данные при этом должны быть однотипными или логически связанными.

Правильный выбор структур данных играет ключевую роль в разработке программного обеспечения, влияя на сложность реализации и качество работы программ. Различные структуры данных могут требовать различного объема памяти для хранения одних и тех же данных, а алгоритмы, использующие разные структуры данных, могут иметь разную эффективность. Подбор наиболее подходящей структуры данных для конкретной задачи позволяет выполнять множество операций, используя минимальные ресурсы.

В современном программировании структуры данных являются неотъемлемой частью профессионального подхода, и поэтому они широко присутствуют в стандартных библиотеках языков программирования, таких как STL для C++. Структура данных представляет собой совокупность данных, связей между ними и функций или операций, которые могут быть применены к этим данным.

**Классификация структур данных:**

**Линейные:**

* массив;
* список;
* связанный список;
* стек;
* очередь;
* хэш-таблица.

**Иерархические:**

* двоичные деревья;
* n-арные деревья;
* иерархический список.

**Сетевые:**

* неориентированный граф;
* ориентированный граф.

**Табличные:**

* таблица реляционной базы данных;
* двумерный массив.

**Основные функции структур данных:**

1. **Хранение данных:**

* Одной из основных функций структур данных является хранение данных. Структуры данных позволяют организовать данные таким образом, чтобы они занимали минимальное количество памяти и были легко доступны для обработки.

1. **Упорядочение данных:**

* Некоторые структуры данных предназначены для упорядочения данных в определенном порядке, что упрощает поиск, сортировку и обработку данных.

1. **Быстрый доступ к данным:**

* Структуры данных должны обеспечивать быстрый доступ к данным, чтобы операции чтения и записи были эффективными.

1. **Эффективность использования ресурсов:**

* Хорошая структура данных должна быть эффективной в использовании ресурсов, таких как память и процессорное время.

1. **Поддержка операций:**

* Структуры данных должны поддерживать различные операции, такие как добавление, удаление, поиск, сортировка и обновление данных.

# **АЛГОРИТМЫ СОРТИРОВКИ****.**

# **Сортировка -** это процесс упорядочивания заданного множества объектов в определенном порядке. Она позволяет облегчить поиск элементов в упорядоченном множестве и является фундаментальной задачей в программировании.

## При сортировке данных обычно используется только часть информации в качестве ключа сортировки. Ключ определяет порядок элементов, и при обмене элементов происходит перемещение всей структуры данных. Например, при сортировке списка почтовой рассылки ключом может быть почтовый индекс, но сортируются все данные, включая адреса.

## Существует множество алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки. При оценке алгоритмов учитываются следующие критерии:

## - Эффективность алгоритма, которая определяется количеством сравнений и обменов элементов. Обмены требуют больше времени, чем сравнения. Некоторые алгоритмы имеют экспоненциальную зависимость времени работы, в то время как у других время работы логарифмически зависит от количества элементов.

## - Время работы в лучшем и худшем случаях. Этот критерий важен для оценки производительности алгоритма в различных сценариях. Некоторые алгоритмы могут иметь хорошее среднее время работы, но работать очень медленно в худшем случае.

## - Поведение алгоритма сортировки. Хороший алгоритм должен иметь естественное поведение, то есть время сортировки должно быть минимальным для уже упорядоченного списка элементов, увеличиваться по мере неупорядоченности списка и достигать максимума, когда элементы расположены в обратном порядке.

## Существуют различные методы сортировки массивов, от простых до более сложных. Простые методы требуют квадратичное количество сравнений, в то время как более сложные методы имеют линейно-логарифмическую сложность. Для небольших массивов простые методы могут быть более эффективными, так как они просты в понимании и имеют меньшую сложность операций.

## **Сортировка методом "пузырька" (простого обмена)**

Самый известный алгоритм – *пузырьковая сортировка* (*bubble sort*, сортировка методом пузырька или просто сортировка пузырьком). Его популярность объясняется интересным названием и простотой самого алгоритма. Алгоритм состоит из следующих шагов:

1. Проходы по массиву: Алгоритм выполняет повторяющиеся проходы по сортируемому массиву. На каждом проходе сравниваются попарно элементы и, если порядок в паре неверный, выполняется обмен элементов.

2. Проверка наличия обменов: После каждого прохода проверяется, были ли выполнены обмены элементов. Если на очередном проходе не было выполнено ни одного обмена, значит, массив уже отсортирован, и алгоритм завершает свою работу.\

3. Упорядочивание элементов: При каждом проходе алгоритма наибольший элемент "всплывает" к концу массива, а наименьший элемент "опускается" к началу массива. Таким образом, по мере проходов элементы упорядочиваются, и на каждом проходе нужно обрабатывать на один элемент меньше, чем на предыдущем проходе.

Алгоритм продолжает повторять проходы по массиву до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, и массив становится отсортированным.



**Описание функции сортировки методом пузырька**

**void** **BubbleSort** (**int** k,**int** x[max])

{

**int** i,j,buf;

**for** (i=k-**1**;i>**0**;i--)

**for** (j=**0**;j<i;j++)

**if** (x[j]>x[j+**1**])

{

buf=x[j];

x[j]=x[j+**1**];

x[j+**1**]=buf;} }

## **Сортировка методом простого выбора (Selection sort)**

Это метод при котором проходим по массиву и на каждой итерации ищем минимальное значение и перемещаем его в начало массива. Затем снова начинаем искать минимальный элемент среди оставшихся элементов и продолжаем этот процесс до тех пор, пока весь массив не будет отсортирован.

Шаги алгоритма:

1. Находим минимальное значение в текущей части массива;

2. Производим обмен этого значения со значением на первой неотсортированной позиции;

3. Далее сортируем хвост массива, исключив из рассмотрения уже отсортированные элементы.



**Описание функции сортировки методом простого выбора**

**int** **p\_dv**(**int** a[], **int** n, **int** x)

{

**int** i=**0**, j=n-**1**, m;

**while**(i<j)

{

m=(i+j)/**2**;

**if** (x > a[m])

i=m+**1**;

**else** j=m;

}

**if** (a[i]==x) **return** i;

**else** **return** -**1**;}

## 

## **Сортировка методом Quicksort**

Быстрая сортировка (англ. quicksort) – это метод сортировки значений в списке в последовательные списки с помощью повторяющейся процедуры.

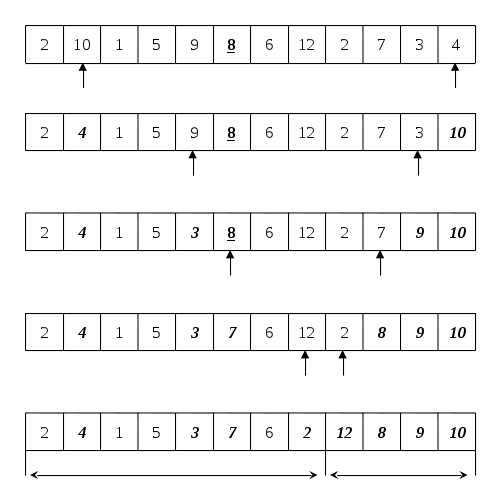
В методе быстрой сортировки выбирается значение из основного списка, которое называется опорным значением. Остальные значения разделяются на два списка:

Первый список содержит значения, которые меньше либо равны опорному значению. Эти значения располагаются слева от опорного значения.

Второй список содержит значения, которые больше опорного значения. Эти значения располагаются справа от опорного значения.

Метод быстрой сортировки повторяется для всех результирующих списков, пока не останется только одно значение или пустой список значений.

После этого вы выбираете последнее одиночное значение, и если значение располагается слева от опорного значения, оно остается таким, пока вы не дойдете до первого опорного значения вверху.



**Описание функции сортировки Quicksort:**

**void** **qsort**(**int** l, **int** r)

{

**int** w,x,i,j;

i=l;

j=r;

x=a[(l+r)/**2**];

**while** (i<=j)

{

**while** ( a[i]<x) i++;

**while** (x<a[j]) j--;

**if** (i<=j)

{

w=a[i]; a[i]=a[j]; a[j]=w;

i++; j--;

}

}

**if** (l<j) qsort(l,j);

**if** (i<r) qsort(i,r);

}

# **АЛГОРИТМЫ ПОИСКА**

Поиск является важной процедурой обработки структурированной информации и привлекает внимание ученых и программистов с самого начала компьютерной эры. Начиная с 1950-х годов, были разработаны алгоритмы для решения задачи поиска элементов с определенными свойствами в заданном множестве.

У каждого алгоритма есть свои преимущества и недостатки. Поэтому важно выбрать тот алгоритм, который лучше всего подходит для решения конкретной задачи.

При выборе алгоритма поиска необходимо учитывать такие факторы, как размер данных, требования к скорости выполнения, доступность предварительно сгенерированных индексов (если таковые имеются), а также особенности самих данных и типов операций поиска, которые нужно выполнить.

## **Последовательный поиск**

**Последовательный (линейный) поиск** – это простой метод поиска в массиве, который последовательно проверяет каждый элемент массива на соответствие заданному значению.

Идея этого метода заключается в следующем. Множество элементов просматривается последовательно в некотором порядке, гарантирующем, что будут просмотрены все элементы множества (например, слева направо). Если в ходе просмотра множества будет найден искомый элемент, просмотр прекращается с положительным результатом; если же будет просмотрено все множество, а элемент не будет найден, алгоритм должен выдать отрицательный результат.

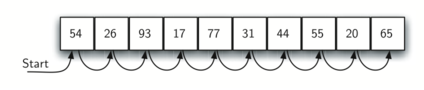
Алгоритм последовательного поиска:

Шаг 1. Полагаем, что значение переменной цикла i=0.

Шаг 2. Если значение элемента массива x[i] равно значению ключа key, то возвращаем значение, равное номеру искомого элемента, и алгоритм завершает работу. В противном случае значение переменной цикла увеличивается на единицу i=i+1.

Шаг 3. Если i<k, где k – число элементов массива x, то выполняется Шаг 2, в противном случае – работа алгоритма завершена и возвращается значение равное -1.

При наличии в массиве нескольких элементов со значением key данный алгоритм находит только первый из них (с наименьшим индексом).



**Описание функции линейного (последовательного поиска):**

**int** **p\_lin1**(**int** a[],**int** n, **int** x)

{

**for**(**int** i=**0**; i < n; i++)

**if** (a[i]==x) **return** i;

**return** -**1**;

}

**int** **p\_lin2**(**int** a[],**int** n, **int** x)

{

a[n]=x;

**int** i=**0**;

**while** (a[i]!=x) i++;

**if** (i==n) **return** -**1**;

**else** **return** i;

}

## 

## **Поиск делением пополам**

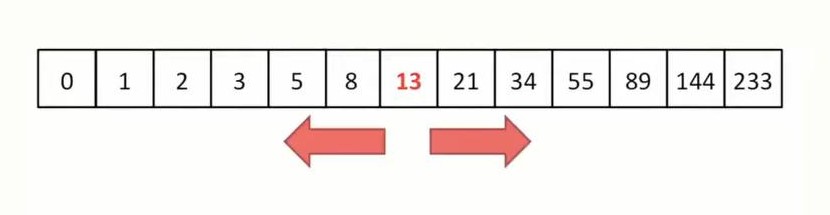
**Двоичный(бинарный) поиск —** это алгоритм поиска, который находит позицию заданного значения в отсортированном массиве. За каждый «шаг» этого алгоритма половина массива устраняется, что делает его очень эффективным для поиска в больших отсортированных данных.

Алгоритм:

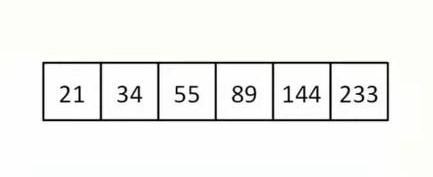
1. Сортируем массив
2. Делим массив на две части и определяем элемент в середине
3. Если элемент, который мы ищем, равен элементу в середине массива, нужный индекс найден.
4. Если элемент, который мы ищем, больше элемента в середине массива, продолжаем поиск в правой части массива.
5. Если элемент, который мы ищем, меньше элемента в середине массива, продолжаем поиск в левой части массива.
6. Повторяем эти действия, пока не найдем искомый элемент или пока не останется только один элемент в подмассиве.

Допустим на данном примера нам нужно найти число 144.

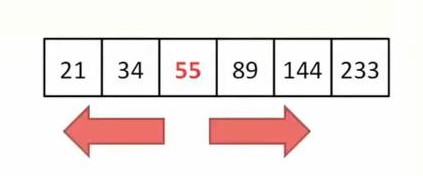
1. Выберем число в середине – 13.

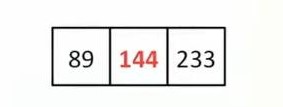


1. Смотрим 144 больше или меньше 13. Так как 144 больше 13 то мы откидываем все числа которые меньше 13.



1. Выбираем любое число по середине - 55.



1. Опять смотри , что 144 больше 55 и отбрасываем левую часть**.** 
2. Выбираем число по середине и находим наше число 144.

**Описание функции бинарного поиска**

**int** **p\_dv**(**int** a[], **int** n, **int** x)

{

**int** i=**0**, j=n-**1**, m;

**while**(i<j)

{

m=(i+j)/**2**;

**if** (x > a[m])

i=m+**1**;

**else** j=m;

}

**if** (a[i]==x) **return** i;

**else** **return** -**1**;}

# **ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ:**

В моём курсовом проекте были разработаны пользовательские функции работы с файлами, записи и редактирования данных, а также функции с применением вышеупомянутых алгоритмов сортировки и поиска. Однако программа начинает работать с вывода меню, где пользователь может выбрать то, что его интересует.

## **Функция для поиска выбором (двоичный поиск) SearchZnuch().**

Эта функция принимает вектор **sotrVector**, содержащий объекты типа **Sotrudn**, и значение **x**, которое представляет собой стаж работника, которое нужно найти. Функция выполняет линейный поиск вектора **sortVector** и выводит на экран название работника и его данные, если сотрудник с заданной стажем найден. Функция начинает поиск, перебирая элементы вектора **sotrVector** с помощью цикла  **while**. Для каждого сотрудника в векторе проверяется, равен ли стаж сотружника **x** стажу, переданной в качестве аргумента. Если найден сотрудник с нужным стажем, выводится сообщение о том, что сотрудник найден, и его данные. **firstIndex, lastIndex**: Эти переменные используются для поиска диапазона сотрудников с одинаковым стажем.  **firstIndex** хранит индекс первого сотрудника с требуемым стажем, а **lastIndex** - индекс последнего сотрудника с таким же стажем.

Эта функция реализует простой линейный поиск вектора **sotrVector** на основе стажа и выводит результат поиска на экран.

void searchZnuch(vector<Sotrudn>& sotrVector) {

sortVozrastanie(sotrVector, 0, (sotrVector.size() - 1));

int x;

cout << "Стаж работы: ";

cin >> x;

int i = 0, j = (sotrVector.size() - 1), m;

int startIdx = -1, endIdx = -1; // Индексы начала и конца диапазона

while (i <= j) {

m = (i + j) / 2;

if (calculateExperience(sotrVector[m]) < x)

i = m + 1;

else if (calculateExperience(sotrVector[m]) > x)

j = m - 1;

else {

// Найден сотрудник с требуемым стажем, ищем начало и конец диапазона

startIdx = m;

endIdx = m;

while (startIdx > 0 && calculateExperience(sotrVector[startIdx - 1]) == x)

startIdx--;

while (endIdx < sotrVector.size() - 1 && calculateExperience(sotrVector[endIdx + 1]) == x)

endIdx++;

break;

}

}

if (startIdx != -1 && endIdx != -1) {

cout << "----------------------------------------------------------------------" << endl;

for (int k = startIdx; k <= endIdx; k++) {

cout << sotrVector[k].name << ", " << sotrVector[k].dolshnost << ", " << sotrVector[k].departament << endl;

}

cout << "----------------------------------------------------------------------" << endl;

}

else {

cout << "\033[1;33mСотрудник не найден.\033[0m" << endl;

}

}

## **Функция создания(сохранения в) файла saveDatatoFile().**

Эта функция принимает строку **filename** в качестве аргумента и создает файл с указанным именем. Она использует объект **ofstream**, который предоставляет функциональность для записи данных в файл. Сначала функция пытается открыть файл с указанным именем для записи. Если файл не удается открыть, она выводит сообщение об ошибке в поток стандартного вывода ошибок **cerr** и завершает выполнение функции, не создавая файл. Если файл успешно открыт для записи, выводится сообщение об успешном создании файла в поток стандартного вывода **cout**. Затем файл закрывается вызовом метода **close** объекта **file**.

void saveDataToFile(const vector<Sotrudn>& sotrVector) {

string filename;

cout << "Введите имя файла для сохранения данных: ";

cin >> filename;

ofstream file(filename);

if (!file) {

cout << "Ошибка открытия файла.\n";

return;

}

for (const Sotrudn& sotrudn : sotrVector) {

file << "ФИО: " << sotrudn.name << endl;

file << "Департамент: " << sotrudn.departament << endl;

file << "Должность: " << sotrudn.dolshnost << endl;

file << "Дата начала работы: " << sotrudn.date.day << "." << sotrudn.date.month << "." << sotrudn.date.year << endl;

file << endl;

}

file.close();

cout << "\033[1;32mДанные сохранены в файл.\033[0m" << endl;

}

## **Функция для загрузки данных из файла readData ().**

Эта функция принимает строку **r** в качестве аргумента и открывает файл с указанным именем для чтения. Она использует объект ifstream, который предоставляет функциональность для чтения данных из файла. Если файл не удается открыть для чтения, функция выводит сообщение об ошибке в поток стандартного вывода ошибок **cerr** и завершает выполнение функции. Если файл успешно открыт для чтения, функция начинает считывать строки из файла поочередно с помощью функции **getline**. Каждая считанная строка предполагается содержащей информацию о сотруднике в формате *"ФИО отдел должность дата начала работы".* Для разбора каждой строки используется методы **substr**() и **find**() для извлечения необходимой информации из строки. Полученные данные записываются в объект типа **Sotrudn**. Далее, после успешного разбора строки, информация о сотруднике  добавляются в вектор **sotrVector** с помощью **push\_back**(). Это позволяет сохранить информацию о всех сотрудниках в векторе для дальнейшего использования. После завершения чтения содержимого файла, файл закрывается вызовом метода **close** объекта **file**.

Эта функция позволяет загрузить содержимое из файла.

void readData(const string& , vector<Sotrudn>& sotrVector) {

string r;

cout << "Введите название файла: ";

cin >> r;

ifstream file(r);

if (!file) {

cerr << "\033[1;31mОшибка открытия файла\033[0m" << endl;

return;

}

string line;

while (getline(file, line)) {

if (line.find("ФИО:") != string::npos) {

Sotrudn sotrudn;

sotrudn.name = line.substr(5);

getline(file, line); // Считываем следующую строку

size\_t pos = line.find(":");

sotrudn.departament = stoi(line.substr(pos + 1));

getline(file, line); // Считываем следующую строку

pos = line.find(":");

sotrudn.dolshnost = line.substr(pos + 1);

getline(file, line); // Считываем следующую строку

pos = line.find(":");

string dateStr = line.substr(pos + 1);

// Разбор даты

size\_t dot1 = dateStr.find('.');

size\_t dot2 = dateStr.rfind('.');

sotrudn.date.day = stoi(dateStr.substr(0, dot1));

sotrudn.date.month = stoi(dateStr.substr(dot1 + 1, dot2 - dot1 - 1));

sotrudn.date.year = stoi(dateStr.substr(dot2 + 1));

getline(file, line); // Считываем пустую строку

sotrVector.push\_back(sotrudn);

}} file.close();}

## **Функция добавления данных в файл ADD** **().**

Эта функция принимает ссылку на вектор **sotrVector** в качестве аргумента. Внутри функции создается новый объект **Sotrudn** с именем **newSotrudn**. Этот объект будет заполняться данными, которые вводит пользователь. Затем функция запрашивает у пользователя данные сотруднике: ФИО, отдел, должность,дату начала работы. Для ввода названия продукта используется функция **getline**, чтобы обеспечить возможность ввода названия, содержащего пробелы. После получения всех данных о продукте, после ввода всех данных новый объект **newSotrudn** добавляется в конец вектора **sotrVector** с помощью **sotrVector.push\_back(newSotrudn)**.

Эта функция позволяет пользователю добавлять данные о сотрудниках в конец файла.

void ADD(vector<Sotrudn>& sotrVector) {

Sotrudn newSotrudn;

cout << "Введите ФИО: ";

cin.ignore();

getline(cin, newSotrudn.name);

cout << "Введите номер департамента: ";

cin >> newSotrudn.departament;

cout << "Введите должность: ";

cin.ignore();

getline(cin, newSotrudn.dolshnost);

cout << "Введите дату начала работы:\n";

newSotrudn.date = getValidDate();

cout << endl;

// Остальная часть функции ADD

sotrVector.push\_back(newSotrudn);

}

## **Функция для просмотра файла printInfo().**

1. Эта функция принимает в качестве аргумента константную ссылку на объект типа **Sotrudn** с именем **sotrudn**. Внутри функции последовательно выводятся на экран следующие данные о сотруднике:
   * ФИО: sotrudn.name
   * Номер департамента: **sotrudn.departament**
   * Должность: **sotrudn.dolshnost**
   * Дата начала работы: **sotrudn.date.day, sotrudn.date.month, sotrudn.date.year**. После вывода всех данных о сотруднике, выводится пустая строка для разделения информации о разных сотрудниках.

Эта функция позволяет просмотреть содержимое файла.

void printInfo(const Sotrudn& sotrudn) {

cout << "ФИО: " << sotrudn.name << endl;

cout << "Департамент: " << sotrudn.departament << endl;

cout << "Должность: " << sotrudn.dolshnost << endl;

cout << "Дата начала работы: " << sotrudn.date.day << "."

<< sotrudn.date.month << "." << sotrudn.date.year << endl;

cout << endl;

}

## **Функция проверки на дату getValidDate()**

## Функция создает объект **Date** с именем **date**, который будет хранить введенную пользователем дату.

1. Далее начинается бесконечный цикл while(true), который будет продолжаться до тех пор, пока пользователь не введет корректную дату.
2. Внутри цикла выводится сообщение, запрашивающее ввод даты в формате "дд.мм.гггг". Пользовательский ввод сохраняется в строковую переменную **input**.
3. Затем создается объект **stringstream ss** с содержимым из input. Используя этот объект, происходит разбор введенной даты на отдельные компоненты: день, месяц и год. Для разделения компонентов используются символы точки.
4. После разбора даты проводятся проверки
5. Если все проверки пройдены успешно, то цикл завершается, и функция возвращает объект **date**, содержащий корректную введенную дату.

Таким образом, функция **getValidDate()** обеспечивает надежный ввод даты пользователем, проверяя ее на корректность.

Date getValidDate() {

Date date;

string input;

while (true) {

cout << "Введите дату в формате дд.мм.гггг: ";

getline(cin, input);

stringstream ss(input);

char dot1, dot2;

ss >> date.day >> dot1 >> date.month >> dot2 >> date.year;

if (ss.fail() || dot1 != '.' || dot2 != '.' || !isValidDate(date.day, date.month, date.year)) {

cout << "Ошибка ввода! Повторите ввод.\n";

}

else {

break;

}

}

return date;

}

## **Функция (линейного поиска) searchDiapaz ().**

Функция запрашивает у пользователя минимальный и максимальный стаж, сохраняя их в переменные **min** и **max**. Затем инициализируется булевая переменная **found** в **false** для отслеживания, был ли найден хотя бы один подходящий сотрудник.

Далее начинается цикл **for**, который итерируется по вектору **sotrVector**, содержащему информацию о сотрудниках. Для каждого сотрудника вычисляется его стаж с помощью функции **calculateExperience**(). Если стаж сотрудника находится в заданном диапазоне, то выводится информация о нем, и **found** устанавливается в **true**.

После цикла, если ни один сотрудник не был найден (т.е. **found** осталась **false**), выводится сообщение "Сотрудник не найден".

Эта функция реализует простой линейный поиск вектора на основе стажа и выводит результат поиска на экран. void searchDiapaz(const vector<Sotrudn>& sotrVector) {

int min, max;

cout << "Минимальный стаж (в годах): ";

cin >> min;

cout << "Максимальный стаж (в годах): ";

cin >> max;

bool found = false;

cout << "----------------------------------------------------------------------" << endl;

for (const auto& sotrudn : sotrVector) {

int experience = calculateExperience(sotrudn);

if ((min <= experience) && (experience <= max)) {

cout << "ФИО: " << sotrudn.name << endl;

cout << "Отдел: " << sotrudn.departament << endl;

cout << "Должность: " << sotrudn.dolshnost << endl;

cout << "Дата: " << sotrudn.date.day << "." << sotrudn.date.month << "." << sotrudn.date.year << endl;

cout << "----------------------------------------------------------------------" << endl;

found = true;

}

}

if (!found) {

cout << "\033[1;33mСотрудник не найден.\033[0m" << endl;

}

}

## **Функция для проверки на високосный год isLeapYear().**

Эта функция проверяет год вводимым пользователем на високосный.

Данная функция проверяет дату на следующие условия

1. Год кратен 4, то есть его можно разделить на 4 без остатка.
2. Но при этом Год не кратен 100, то есть не может быть разделен на 100 без остатка.
3. Исключение составляют годы, кратные 400 - они всегда считаются високосными.

bool isLeapYear(int year) {

return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

}

## **Функция сортировки (quicksort) sortVozrastanie().**

Этот рекурсивная функция быстрой сортировки (**quickSort**), которая сортирует вектор **sotrVector** в порядке возрастания стажа работы.

1. Внутри функции инициализируются три целочисленные переменные **i**, **j** и **x**. Переменные **i** и **j** используются как указатели на левый и правый концы подмассива соответственно. Переменная x присваивается медианным значением поля **experience**
2. Объявляется переменная типа **Sotrudn** под названием **rh**, которая будет использоваться для обмена элементами.
3. Используется цикл **while**, чтобы разделить подмассив на две части. Цикл продолжается, пока **i** меньше или равно **j**. Внутри цикла:
4. После шага разделения функция рекурсивно вызывает себя для левого и правого подмассивов

void sortVozrastanie(vector<Sotrudn>& sotrVector, int l, int r) {

int i = l;

int j = r;

int x = sotrVector[(l + r) / 2].experience;

Sotrudn rh;

while (i <= j) {

while (sotrVector[i].experience < x) i++;

while (sotrVector[j].experience > x) j--;

if (i <= j) {

rh = sotrVector[i];

sotrVector[i] = sotrVector[j];

sotrVector[j] = rh;

i++;

j--;

}

} if (l < j) sortVozrastanie(sotrVector, l, j);

if (i < r) sortVozrastanie(sotrVector, i, r)

1. **Функция сортировка методом прямого выбора sortUb().**

Основные шаги алгоритма:

1. Выбирается опорный элемент **x** - значение поля **experience** из среднего элемента диапазона, который нужно отсортировать.
2. Инициализируются указатели **i** и **j** на начало и конец диапазона.
3. Происходит разбиение диапазона на две части:
   * Элементы слева от **i** имеют значение **experience** больше, чем **x**.
   * Элементы справа от **j** имеют значение **experience** меньше, чем **x**.
4. Производится обмен элементов между **i** и **j**, пока **i** не станет больше **j**.
5. Рекурсивно вызывается функция **sortUb** для левой и правой частей диапазона, чтобы отсортировать их.

void sortUb(vector<Sotrudn>& sotrVector, int l, int r) {

int i = l;

int j = r;

int x = sotrVector[(l + r) / 2].experience;

Sotrudn rh;

while (i <= j) {

while (sotrVector[i].experience > x) i++;

while (sotrVector[j].experience < x) j--;

if (i <= j) {

rh = sotrVector[i];

sotrVector[i] = sotrVector[j];

sotrVector[j] = rh;

i++;

j--;

}

}

if (l < j) sortVozrastanie(sotrVector, l, j);

if (i < r) sortVozrastanie(sotrVector, i, r);

}

## **Функция для расчета стажа сотрудников calculateExperience()**

1. Получение текущей даты:
   * Используется функция **time()** для получения текущего времени в секундах с начала эпохи (1 января 1970 года).
   * Затем с помощью функции **localtime\_s()**извлекаются год и месяц из структуры **tm**, которая представляет локальное время.
2. Получение даты начала работы:
   * Год и месяц начала работы берутся из соответствующих полей объекта **Sotrudn**.
3. Вычисление стажа работы:
   * Стаж работы в годах рассчитывается как разница между текущим годом и годом начала работы (**experience** **=** **currentYear** - **startYear**).
4. Возврат результата:
   * Функция возвращает рассчитанный стаж работы в годах.

int calculateExperience(const Sotrudn& sotrudn) {

// Получение текущей даты

time\_t currentTime;

time(&currentTime);

tm currentDate;

localtime\_s(&currentDate, &currentTime);

int currentYear = currentDate.tm\_year + 1900;

int currentMonth = currentDate.tm\_mon + 1;

// Получение даты начала работы

int startYear = sotrudn.date.year;

int startMonth = sotrudn.date.month;

// Вычисление стажа работы

int experience = currentYear - startYear;

if (currentMonth < startMonth)

experience--;

return experience;

}

## **Функция удаления элементов Remove\_BY\_INDEX().**

Эта функция удаляет сотрудников с указанным именем из файла, используя вектор **Sotrudn** длявременного хранения данных, а затем сохраняет обновленный список продуктов в файл.

1. Запрашивает у пользователя ввод индекса для удаления.
2. Проверяет, что индекс в допустимом диапазоне.
3. Если индекс допустим, удаляет элемент с этим индексом из вектора

.

1. Выводит сообщение об успешном удалении или об ошибке, если индекс недопустим.
2. Эта функция предоставляет возможность удалить из файла сотрудников с определенным именем, если они там присутствуют.

void REMOVE\_BY\_INDEX(vector<Sotrudn>& sotrVector) {

int index;

cout << "Введите номер добавления работника для удаления: ";

cin >> index;

if (index >= 1 && index <= sotrVector.size()) {

sotrVector.erase(sotrVector.begin() + index - 1);

cout << "\033[1;32mРаботник успешно удален.\033[0m" << endl;

}

else {

cout << "\033[1;33mНедопустимый номер добавления работника.\033[0m" << endl;

}

}

**13.Функция проверки даты isValidDate().**

Она выполняет следующие проверки:

1. Проверяет, что год не отрицательный, месяц находится в диапазоне от 1 до 12, а день не меньше 1.
2. Определяет максимальное количество дней в месяце, основываясь на том, является ли месяц февралем (учитывая високосный год) или одним из месяцев с 30 днями.
3. Возвращает **true**, если день не превышает максимальное количество дней для данного месяца и года, и **false** в противном случае.

Эта функция используется для валидации дат, чтобы гарантировать, что введенные пользователем даты соответствуют правильному формату и существующим календарным датам.

bool isValidDate(int day, int month, int year) {

if (year < 0 || month < 1 || month > 12 || day < 1) {

return false;

}

int maxDays = 31; // Максимальное количество дней в месяце

if (month == 2) { // Февраль

maxDays = isLeapYear(year) ? 29 : 28;

}

else if (month == 4 || month == 6 || month == 9 || month == 11) {

maxDays = 30;

}

return day <= maxDays;

}

## 

## **14.Основная функция main().**

Она представляет собой вызов меню, где пользователь может выбирать различные действия для работы с файлом. Программа выполняет создание файла, запись в файл, добавление информации, удаление, просмотр информации, линейный поиск, линейную сортировку, бинарный поиск, вывод информации о продуктах по типу и статусу, а также реализует алгоритм быстрой сортировки и функцию обратной сортировки. После выполнения каждой операции программа ожидает нажатия клавиши для продолжения.

int main() {

setlocale(LC\_ALL, "RU");

vector<Sotrudn> sotrVector;

menu(sotrVector);

return 0;

}

# 

# **ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ**

Моя программа представляет собой меню для работы с файлом, где пользователь может выполнять различные операции для управления данными в файле. Она позволяет пользователю:

1. создавать новый файл для хранения данных.

2. записывать в файл новую информацию.

3. добавлять новые данные в конец файла без удаления существующей информации.

4. удалять определенные данные из файла.

5. просматривать содержимое файла для получения информации.

6. осуществлять поиск определенного элемента в файле путем последовательного просмотра каждого элемента.

7. сортировать элементы в файле в порядке возрастания или убывания.

8. осуществлять эффективный поиск элемента в отсортированном файле путем деления области поиска пополам.

9. выполнять сортировку, которая разделяет массив на более мелкие части для последующей сортировки.

10. сортировать элементы по убыванию элемента в файле.

11. осуществлять поиск по заданному типу и статусу.

# **ВЫВОД:**

Программа предоставляет пользователю инструменты для структурирования, хранения и обработки данных в файле. Благодаря разнообразным функциям, пользователь может легко добавлять новую информацию, редактировать существующие данные, искать нужные элементы, сортировать данные по различным критериям и удалять ненужные записи.

Эти возможности помогают пользователю эффективно управлять информацией, обеспечивая быстрый доступ к необходимым данным и облегчая процесс обработки информации. Благодаря широкому спектру функций программа становится мощным инструментом для работы с данными, который помогает организовать информацию таким образом, чтобы пользователь мог эффективно выполнять свои задачи и достигать поставленных целей.

# 

# **ПРИЛОЖЕНИЕ А:**

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

#include <limits>

#include <fstream>

#include <ctime>

using namespace std;

struct Date {

int day;

int month;

int year;

};

struct Sotrudn {

string name;

int departament;

string dolshnost;

Date date;

int experience;

};

bool isLeapYear(int year) {

return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

}

bool isValidDate(int day, int month, int year) {

if (year < 0 || month < 1 || month > 12 || day < 1) {

return false;

}

int maxDays = 31; *// Максимальное количество дней в месяце*

if (month == 2) { *// Февраль*

maxDays = isLeapYear(year) ? 29 : 28;

}

else if (month == 4 || month == 6 || month == 9 || month == 11) {

maxDays = 30;

}

return day <= maxDays;

}

Date getValidDate() {

Date date;

string input;

while (true) {

cout << "Введите дату в формате дд.мм.гггг: ";

getline(cin, input);

stringstream ss(input);

char dot1, dot2;

ss >> date.day >> dot1 >> date.month >> dot2 >> date.year;

if (ss.fail() || dot1 != '.' || dot2 != '.' || !isValidDate(date.day, date.month, date.year)) {

cout << "Ошибка ввода! Повторите ввод.\n";

}

else {

break;

}

}

return date;

}

void printInfo(const Sotrudn& sotrudn) {

cout << "ФИО: " << sotrudn.name << endl;

cout << "Департамент: " << sotrudn.departament << endl;

cout << "Должность: " << sotrudn.dolshnost << endl;

cout << "Дата начала работы: " << sotrudn.date.day << "."

<< sotrudn.date.month << "." << sotrudn.date.year << endl;

cout << endl;

}

int getValidChoice() {

int choice;

while (!(cin >> choice) || choice < 1 || choice > 11) {

cout << "Некорректный выбор. Пожалуйста, выберите число от 1 до 11: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

return choice;

}

void ADD(vector<Sotrudn>& sotrVector) {

Sotrudn newSotrudn;

cout << "Введите ФИО: ";

cin.ignore();

getline(cin, newSotrudn.name);

cout << "Введите номер департамента: ";

cin >> newSotrudn.departament;

cout << "Введите должность: ";

cin.ignore();

getline(cin, newSotrudn.dolshnost);

cout << "Введите дату начала работы:\n";

newSotrudn.date = getValidDate();

cout << endl;

*// Остальная часть функции ADD*

sotrVector.push\_back(newSotrudn);

}

void saveDataToFile(const vector<Sotrudn>& sotrVector) {

string filename;

cout << "Введите имя файла для сохранения данных: ";

cin >> filename;

ofstream file(filename);

if (!file) {

cout << "Ошибка открытия файла.\n";

return;

}

for (const Sotrudn& sotrudn : sotrVector) {

file << "ФИО: " << sotrudn.name << endl;

file << "Департамент: " << sotrudn.departament << endl;

file << "Должность: " << sotrudn.dolshnost << endl;

file << "Дата начала работы: " << sotrudn.date.day << "." << sotrudn.date.month << "." << sotrudn.date.year << endl;

file << endl;

}

file.close();

cout << "\033[1;32mДанные сохранены в файл.\033[0m" << endl;

}

void readData(const string& , vector<Sotrudn>& sotrVector) {

string r;

cout << "Введите название файла: ";

cin >> r;

ifstream file(r);

if (!file) {

cerr << "\033[1;31mОшибка открытия файла\033[0m" << endl;

return;

}

string line;

while (getline(file, line)) {

if (line.find("ФИО:") != string::npos) {

Sotrudn sotrudn;

sotrudn.name = line.substr(5);

getline(file, line); *// Считываем следующую строку*

size\_t pos = line.find(":");

sotrudn.departament = stoi(line.substr(pos + 1));

getline(file, line); *// Считываем следующую строку*

pos = line.find(":");

sotrudn.dolshnost = line.substr(pos + 1);

getline(file, line); *// Считываем следующую строку*

pos = line.find(":");

string dateStr = line.substr(pos + 1);

*// Разбор даты*

size\_t dot1 = dateStr.find('.');

size\_t dot2 = dateStr.rfind('.');

sotrudn.date.day = stoi(dateStr.substr(0, dot1));

sotrudn.date.month = stoi(dateStr.substr(dot1 + 1, dot2 - dot1 - 1));

sotrudn.date.year = stoi(dateStr.substr(dot2 + 1));

getline(file, line); *// Считываем пустую строку*

sotrVector.push\_back(sotrudn);

}

}

file.close();

}

int calculateExperience(const Sotrudn& sotrudn) {

*// Получение текущей даты*

time\_t currentTime;

time(&currentTime);

tm currentDate;

localtime\_s(&currentDate, &currentTime);

int currentYear = currentDate.tm\_year + 1900;

int currentMonth = currentDate.tm\_mon + 1;

*// Получение даты начала работы*

int startYear = sotrudn.date.year;

int startMonth = sotrudn.date.month;

*// Вычисление стажа работы*

int experience = currentYear - startYear;

if (currentMonth < startMonth)

experience--;

return experience;

}

*// сортировки:*

*// линейный поиск*

void searchDiapaz(const vector<Sotrudn>& sotrVector) {

int min, max;

cout << "Минимальный стаж (в годах): ";

cin >> min;

cout << "Максимальный стаж (в годах): ";

cin >> max;

bool found = false;

cout << "----------------------------------------------------------------------" << endl;

for (const auto& sotrudn : sotrVector) {

int experience = calculateExperience(sotrudn);

if ((min <= experience) && (experience <= max)) {

cout << "ФИО: " << sotrudn.name << endl;

cout << "Отдел: " << sotrudn.departament << endl;

cout << "Должность: " << sotrudn.dolshnost << endl;

cout << "Дата: " << sotrudn.date.day << "." << sotrudn.date.month << "." << sotrudn.date.year << endl;

cout << "----------------------------------------------------------------------" << endl;

found = true;

}

}

if (!found) {

cout << "\033[1;33mСотрудник не найден.\033[0m" << endl;

}

}

*//QuickSort*

void sortVozrastanie(vector<Sotrudn>& sotrVector, int l, int r) {

int i = l;

int j = r;

int x = sotrVector[(l + r) / 2].experience;

Sotrudn rh;

while (i <= j) {

while (sotrVector[i].experience < x) i++;

while (sotrVector[j].experience > x) j--;

if (i <= j) {

rh = sotrVector[i];

sotrVector[i] = sotrVector[j];

sotrVector[j] = rh;

i++;

j--;

}

}

if (l < j) sortVozrastanie(sotrVector, l, j);

if (i < r) sortVozrastanie(sotrVector, i, r);

}

*//метод прямого выбора*

void sortUb(vector<Sotrudn>& sotrVector, int l, int r) {

int i = l;

int j = r;

int x = sotrVector[(l + r) / 2].experience;

Sotrudn rh;

while (i <= j) {

while (sotrVector[i].experience > x) i++;

while (sotrVector[j].experience < x) j--;

if (i <= j) {

rh = sotrVector[i];

sotrVector[i] = sotrVector[j];

sotrVector[j] = rh;

i++;

j--;

}

}

if (l < j) sortVozrastanie(sotrVector, l, j);

if (i < r) sortVozrastanie(sotrVector, i, r);

}

*//двоичный поиск*

void searchZnuch(vector<Sotrudn>& sotrVector) {

sortVozrastanie(sotrVector, 0, (sotrVector.size() - 1));

int x;

cout << "Стаж работы: ";

cin >> x;

int i = 0, j = (sotrVector.size() - 1), m;

int startIdx = -1, endIdx = -1; *// Индексы начала и конца диапазона*

while (i <= j) {

m = (i + j) / 2;

if (calculateExperience(sotrVector[m]) < x)

i = m + 1;

else if (calculateExperience(sotrVector[m]) > x)

j = m - 1;

else {

*// Найден сотрудник с требуемым стажем, ищем начало и конец диапазона*

startIdx = m;

endIdx = m;

while (startIdx > 0 && calculateExperience(sotrVector[startIdx - 1]) == x)

startIdx--;

while (endIdx < sotrVector.size() - 1 && calculateExperience(sotrVector[endIdx + 1]) == x)

endIdx++;

break;

}

}

if (startIdx != -1 && endIdx != -1) {

cout << "----------------------------------------------------------------------" << endl;

for (int k = startIdx; k <= endIdx; k++) {

cout << sotrVector[k].name << ", " << sotrVector[k].dolshnost << ", " << sotrVector[k].departament << endl;

}

cout << "----------------------------------------------------------------------" << endl;

}

else {

cout << "\033[1;33mСотрудник не найден.\033[0m" << endl;

}

}

void REMOVE\_BY\_INDEX(vector<Sotrudn>& sotrVector) {

int index;

cout << "Введите номер добавления работника для удаления: ";

cin >> index;

if (index >= 1 && index <= sotrVector.size()) {

sotrVector.erase(sotrVector.begin() + index - 1);

cout << "\033[1;32mРаботник успешно удален.\033[0m" << endl;

}

else {

cout << "\033[1;33mНедопустимый номер добавления работника.\033[0m" << endl;

}

}

void menu(vector<Sotrudn>& sotrVector) {

int choice = 0;

Sotrudn sotrudn;

while (choice != 11) {

cout << "Меню:\n";

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

<< "1. Добавить запись\n";

cout << "2. Просмотреть записи\n";

cout << "3. Расчет стажа работы(в годах)\n";

cout << "4. Сохранить в файл \n";

cout << "5. Загрузить данные из файла\n";

cout << "6. Сортировка структуры ( По диапозону)\n";

cout << "7. Сортировка по возрастанию\n";

cout << "8. Сортировка по убыванию \n";

cout << "9. Найти по опред значению \n";

cout << "10. Удаление из списка \n";

cout << "11 Выход\n"

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

cout << "Выберите пункт меню: ";

choice = getValidChoice();

Sotrudn sotrudn;

switch (choice) {

case 1:

ADD(sotrVector);

break;

case 2:

for (const Sotrudn& sotrudn : sotrVector) {

printInfo(sotrudn);

}

break;

case 3:

for (const Sotrudn& sotrudn : sotrVector) {

printInfo(sotrudn);

int experience = calculateExperience(sotrudn);

cout << "Стаж работы: " << experience << " лет" << endl;

}

break;

case 4:

saveDataToFile(sotrVector);

break;

case 5:

readData("имя\_файла.txt", sotrVector);

*// Вывод информации*

for (const auto& sotrudn : sotrVector) {

cout << "ФИО: " << sotrudn.name << endl;

cout << "Отдел: " << sotrudn.departament << endl;

cout << "Должность: " << sotrudn.dolshnost << endl;

cout << "Дата: " << sotrudn.date.day << "." << sotrudn.date.month << "." << sotrudn.date.year << endl;

cout << endl;

}

break;

case 6:

searchDiapaz(sotrVector);

break;

case 7:

sortVozrastanie(sotrVector, 0, sotrVector.size() - 1);

for (const auto& sotrudn : sotrVector) {

cout << "Имя: " << sotrudn.name << ", Стаж работы: " << calculateExperience(sotrudn) << endl;

}

break;

case 8:

sortUb(sotrVector, 0, sotrVector.size() - 1);

for (const auto& sotrudn : sotrVector) {

cout << "Имя: " << sotrudn.name << ", Стаж работы: " << calculateExperience(sotrudn) << endl;

}

break;

case 9:

searchZnuch(sotrVector);

break;

case 10:

REMOVE\_BY\_INDEX(sotrVector);

break;

case 11:

cout << "the end";

break;

default:

cout << "Некорректный выбор. Повторите ввод.\n";

break;

}

}

}

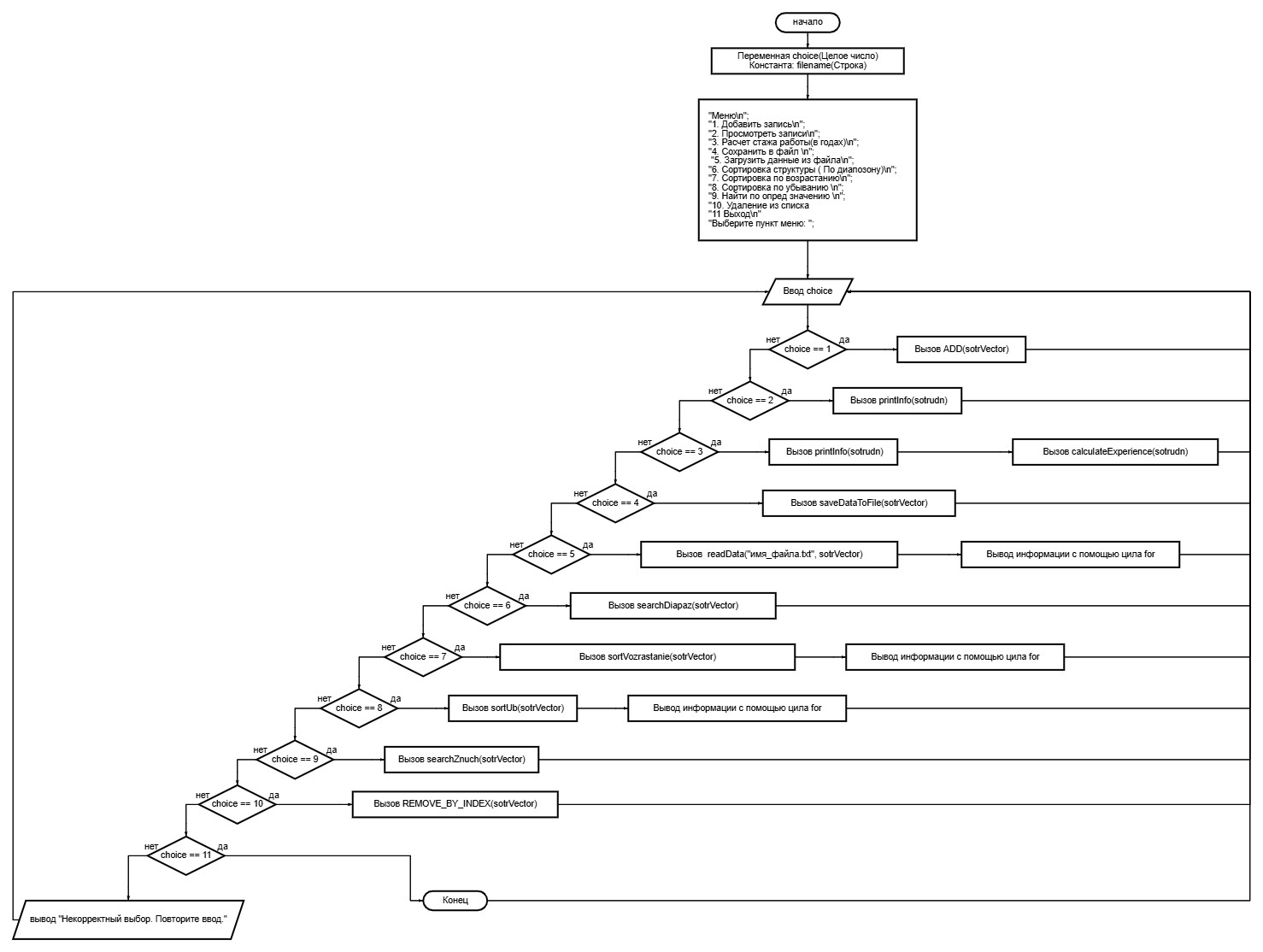
int main() {

setlocale(LC\_ALL, "RU");

vector<Sotrudn> sotrVector;

menu(sotrVector);

return 0;}

**ПРИЛОЖЕНИЕ В:**

# **СПИСОК ЛИТЕРАТУРЫ:**

1. Электронный сборник статей по материалам XLVIII студенческой

международной заочной научно-практической конференции. – Москва: Изд. «МЦНО». – 2017. – № 8 (48) / [Электронный ресурс] – Режим доступа. – URL: http://www.nauchforum.ru/archive/MNF\_tech/8(48).pdf

1. Информационные технологии в управлении. Учебное пособие. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2008. – 336с.
2. Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. — М. : Издательский дом "Вильямс", 2007. - 400 с. : ил. — Парал. тит. англ.
3. Адельсон-Вельский Г. М., Ландис Е. М. (1962). Один алгоритм организации информации. Докл. АН СССР, 146, с. 263-266.
4. ﻿﻿﻿Aho, A. V., J. E. Hopcroft, and J. D. Ullman (1974). The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. (Русский перевод: Ахо A., Хоркрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — М., "Map", 1979.)
5. Зуев.К.И Структуры данных / Зуев.К.И [Электронный ресурс] // Профильное обучение. Информатика 10 класс: [сайт]. — URL: http://profil.adu.by/mod/book/tool/print/index.php?id=3573 (дата обращения: 10.05.2024).
6. НАУЧНЫЕ ИССЛЕДОВАНИЯ МОЛОДЫХ УЧЁНЫХ: сборник статей XII Международной научно-практической конференции. – Пенза: МЦНС «Наука и Просвещение». – 2021. – 252 с.
7. Б. Керниган, Д. Ритчи Язык программирования Си. – AT&T Bell Labs, 1978. – 343 c.
8. Таненбаум Э. Остин Т. Cho Архитектура компьютера. – Питер – 2013. – 320 c.
9. Г.В. Ваныкина, Т.О. Сундукова Структуры и алгоритмы компьютерной обработки данных [Текст] / Г.В. Ваныкина, Т.О. Сундукова — 1. — Новосибирск: ИНТИУИТ, 2011 — 219 c