МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА компьютерных технологий и программной инженерии |

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ) |
| РАЗРАБОТКА ПРОГРАММЫ  «ОТДЕЛ КАДРОВ СОТРУДНИКОВ» |
| по дисциплине: основы программирования |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 | |  |  |  | Андреев Д.И. |
|  | номер группы | |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | | 2019/3781 | |  |  |  |

|  |  |
| --- | --- |
| Шифр ИНДО |  |

Санкт-Петербург 2020

**Содержание**

[**1.** **Постановка задачи** 3](#_Toc56368881)

[**2.** **Описание структур данных** 4](#_Toc56368882)

[**3.** **Описание программы и функций** 5](#_Toc56368886)

[**4.** **Результаты тестирования** 21](#_Toc56368981)

[**5.** **Заключение** 22](#_Toc56368982)

[**6.** **Список используемой литературы** 23](#_Toc56368983)

[**Приложение 1** 24](#_Toc56368984)

[Файл main.cpp 24](#_Toc56368985)

[Файл Worker.h 30](#_Toc56368986)

[Файл Worker.cpp 31](#_Toc56368987)

[Файл LinkList.h 32](#_Toc56368988)

1. **Постановка задачи**

Задачей курсового проекта является разработка программы для заданной предметной области «Отдел кадров сотрудников», которая позволяет вводить информацию, хранить еe в файле, осуществлять поиск, модификацию, сортировку и удаление данных.

Данные о сотруднике хранятся в структуре WORKER, содержащей следующие поля:

• фамилию и инициалы работника;

• название занимаемой должности;

• год поступления на работу.

Задание на поиск: найти работников, чей стаж работы в организации превышает значение, введенное с клавиатуры.

1. **Описание структур данных**

Данные хранятся в классе Worker:

class Worker

{

public:

Worker(char\* name, char\* position, const int& year);

Worker(const Worker& worker);

bool operator==(const Worker& rightVal);

bool operator>(const Worker& rightVal);

bool operator<(const Worker& rightVal);

bool operator!=(const Worker& rightVal);

Worker& operator=(const Worker& rightVal);

const char\* name() const;

const char\* position() const;

const int yearOfEmployment() const;

void setName(const char\* name);

void setPosition(const char\* position);

private:

//Максимальная длина строк в классе

static const int m\_maxStringSize = 25;

//Строки представлен как char (а не const char),

//чтобы была возможность вызывать стандартную функцию копирования строк

char m\_name[m\_maxStringSize];

char m\_position[m\_maxStringSize];

int m\_yearOfEmployment;

};

Так как изменение года поступления на работу не имеет логического смысла, стандартного метода установки значения для данного поля нет.

База данных хранится в виде текстового файла, абсолютный путь к которому указывается пользователем с клавиатуры. Формат файла также указывается пользователем при вводе пути к файлу.

Также в классе присутствуют стандартные перегруженные операторы для возможности сравнения экземпляров класса Worker между собой (используется при реализации сортировки в контейнере), конструктор копирования и оператор присвоения для исключения потери данных и утечек памяти при копировании экземпляров класса. Стандартный конструктор для данного класса отсутствует, так как создание экземпляра с пустыми полями не имеет логического смысла.

1. **Описание программы и функций**

Программа разработана на языке С++ в виде консольного приложения. В функции main() реализован основной цикл программы, содержащий меню пользователя в котором каждому действию соответствует определенная цифра

В качестве контейнера реализован линейный однонаправленный список. Также для упрощения работы с данными, а также для скрытия реализации списка реализован паттерн «Итератор».

Диаграмма классов приведена на рис. 1.

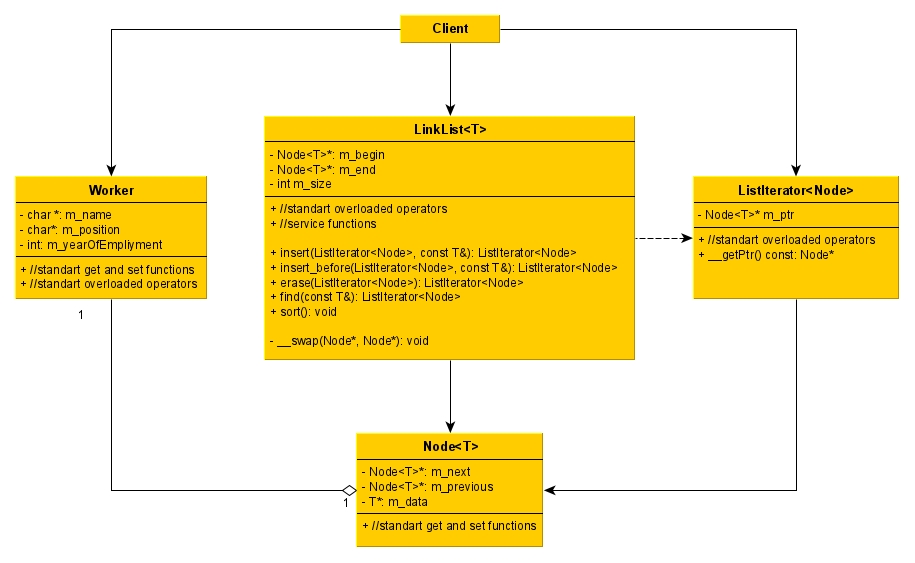


Рис. 1 Диаграмма классов разработанной программы

Описание классов:

Класс Node<T> - шаблонный класс, представляющий собой элемент данных списка. Хранит непосредственно данные, а также указатели на предыдущий и следующий за данным элементы списка.

Определенные в классе типы:

public:

using Value = typename T;

Закрытые поля класса:

node\* m\_next;

node\* m\_previous;

Value\* m\_data;

Открытые методы класса:

* Стандартные функции получения и установки значений:

Value\* data() const { return m\_data; }

node\* next() const { return m\_next; }

node\* previous() const { return m\_previous; }

void setData(Value\* data) { m\_data = data; }

void setNext(node\* next) { m\_next = next; }

void setPrevious(node\* previous) { m\_previous = previous; }

Класс LinkList<T> - шаблонный класс, реализующий методы добавления, удаления и поиска элементов в списке. Также класс реализует функцию сортировки элементов списка.

Определенные в классе типы:

private:

using Node = node<T>;

public:

using iterator = ListIterator<Node>;

Закрытые поля класса:

Node\* m\_begin;

Node\* m\_end;

int m\_size;

Открытые методы класса:

* служебные методы для получения размера списка, итератора на начало и конец списка:

int size() const { return m\_size; }

bool isEmpty() const { return !m\_size; }

iterator begin() const { return iterator(m\_begin); }

iterator end() const { return iterator(m\_end); }

* метод insert(iterator pos, const T&elem) - вставляет элемент elem, после элемента, на который указывает итератор pos. Возвращает итератор на только что вставленный элемент.

iterator insert(iterator pos, const T& elem)

{

//Вставка в пустой список

if (!pos) {

if (!isEmpty())

return pos;

m\_begin = new Node(new T(elem), nullptr, nullptr);

m\_end = new Node(nullptr, nullptr, m\_begin);

m\_begin->setNext(m\_end);

++m\_size;

return begin();

}

Node\* posPtr = pos.\_\_getPtr();

//Вставка в конец списка

if (posPtr->next() == m\_end) {

posPtr->setNext(new Node(new T(elem), m\_end, posPtr));

m\_end->setPrevious(posPtr->next());

++m\_size;

return end();

}

//Вставка нового элемена

posPtr->setNext(new Node(new T(elem), posPtr->next(), posPtr));

++m\_size;

return iterator(posPtr->next());

}

* метод insert\_before(iterator pos, const T& elem) – вставляет элемент elem перед элементом на который указывает итератор pos. Возвращает итератор на только что вставленный элемент.

iterator insert\_before(iterator pos, const T& elem)

{

//Вставка в пустой список

if (!pos)

return insert(pos, elem);

if (pos != begin())

return insert(iterator(pos.\_\_getPtr()->previous()), elem);

Node \*newBegin = new Node(new T(elem), m\_begin, nullptr);

m\_begin->setPrevious(newBegin);

m\_begin = newBegin;

return iterator(newBegin);

}

* метод erase(iterator pos) – удаляет элемент на который указывает итератор pos. Возвращает итератор на следующий элемент после удаленного.

iterator erase(iterator pos)

{

if (pos == end())

return pos;

//Удаление первого элемента

if (pos == begin()) {

Node\* newBegin = m\_begin->next();

newBegin->setPrevious(nullptr);

delete m\_begin;

m\_begin = newBegin;

--m\_size;

return begin();

}

//удаление элемента

Node\* nextPtr = pos.\_\_getPtr()->next();

Node\* prevPtr = pos.\_\_getPtr()->previous();

nextPtr->setPrevious(prevPtr);

prevPtr->setNext(nextPtr);

delete pos.\_\_getPtr();

--m\_size;

return iterator(nextPtr);

}

* метод clear() – удаляет все элементы списка, последовательно вызывая функцию erase(…) для каждого элемента.

void clear()

{

iterator it = begin();

while (!isEmpty())

erase(it++);

}

* метод find(const T& elem) – ведет поиск элемента elem в списке. Возвращает итератор на найденный элемент. Если элемент не найден, возвращает итератор на конец списка

iterator find(const T& elem)

{

if (isEmpty())

return begin();

if (elem == \*begin())

return begin();

iterator it = begin();

while (it != end()) {

if(\*it == elem)

return it;

++it;

}

return end();

}

* метод sort() – сортирует список в порядке возрастания элементов списка. Для корректной работы метода тип элементов при этом должен быть элементарным типом данных или иметь перегруженные операторы сравнения. В данном случае выбрана сортировка перестановками, так как у линейного двунаправленного списка нет произвольного доступа к элементам. Для улучшения читаемости кода реализация обмета элементов вынесена в отдельный закрытый метод \_\_swap(…)

void sort()

{

if (isEmpty())

return;

Node\* ptr = m\_begin;

Node \*cmpPtr = m\_begin->next();

while (ptr != m\_end) {

while (cmpPtr != m\_end) {

if (\*ptr->data() > \*cmpPtr->data()) {

\_\_swap(ptr, cmpPtr);

ptr = cmpPtr;

}

cmpPtr = cmpPtr->next();

}

ptr = ptr->next();

cmpPtr = ptr->next();

}

}

Закрытые методы класса:

* метод \_\_swap(Node<T> \*lPtr, Node<T> \*rPtr) – обменивает элементы списка lPtr и rPtr. Для исключения лишнего кода в функцию передаются непосредственно указатели на данные списка. Для обмена двух элементов списка, данный метод изменяет значения указателей соответствующий элементов. Т.е. при изменении порядка в списке элементы не изменяют своего физического адреса. Также в методе учтена разница в методе обмена рядом стоящих элементов (т.е. имеющих указатели друг на друга) и элементов рядом не стоящих.

void \_\_swap(Node\* lPtr, Node\* rPtr)

{

if (lPtr->next() == rPtr) {

Node\* tempNext = rPtr->next();

Node\* tempPrev = lPtr->previous();

tempNext->setPrevious(lPtr);

if (tempPrev)

tempPrev->setNext(rPtr);

else

m\_begin = rPtr;

rPtr->setNext(lPtr);

rPtr->setPrevious(tempPrev);

lPtr->setPrevious(rPtr);

lPtr->setNext(tempNext);

return;

}

Node\* tempNext = rPtr->next();

Node\* tempPrev = rPtr->previous();

tempNext->setPrevious(lPtr);

tempPrev->setNext(lPtr);

lPtr->next()->setPrevious(rPtr);

if (lPtr->previous())

lPtr->previous()->setNext(rPtr);

else

m\_begin = rPtr;

rPtr->setNext(lPtr->next());

rPtr->setPrevious(lPtr->previous());

lPtr->setNext(tempNext);

lPtr->setPrevious(tempPrev);

}

Класс LinkList<Node<T>> - шаблонный класс, реализующий паттерн итератор. Класс спроектирован таким образом, чтобы по своей функциональности быть максимально похожим на обычный указатель. Также в классе реализовано преобразование в шаблонный тип данных T, чтобы пользователь класса мог не беспокоиться об извлечении данных из итератора.

Определенные в классе типы:

private:

using ValueType = typename Node::Value;

Закрытые поля класса:

Node\* m\_ptr;

Открытые методы класса:

* стандартные перегруженные операторы:

ValueType& operator\*() { return \*m\_ptr->data(); }

ValueType\* operator->() { return m\_ptr->data(); }

bool operator==(const ListIterator& rightIter) { return m\_ptr == rightIter.m\_ptr;}

bool operator!=(const ListIterator& rightIter) { return !operator==(rightIter); }

bool operator==(Node\* rightPtr) { return m\_ptr == rightPtr; }

bool operator!=(Node\* rightPtr) { return !operator==(rightPtr); }

bool operator!() { return !m\_ptr; }

operator ValueType\* () { return m\_ptr->data(); }

ListIterator operator++(int)

{

if (!m\_ptr->next())

return \*this;

ListIterator temp = \*this;

++(\*this);

return temp;

}

ListIterator operator++()

{

if (!m\_ptr->next())

return \*this;

m\_ptr = m\_ptr->next();

return \*this;

}

ListIterator operator--()

{

if (!m\_ptr)

return \*this;

m\_ptr = m\_ptr->previous();

return \*this;

}

ListIterator operator--(int)

{

if (!m\_ptr->previous())

return \*this;

ListIterator temp = \*this;

--(\*this);

return temp;

}

* метод \_\_getPtr() – реализован для возможности доступа непосредственно к классу Node<T> для манипулирования указателями в списке. Данный метод реализован как открытый, чтобы не объявлять класс как friend для класса списка, что убирает дополнительные связи между этими классами и упрощает повторное использование кода при последующей разработке.

Node\* \_\_getPtr() const { return m\_ptr; }

Функции пользовательского интерфейса:

* Вывод всех команд на экран.

Вывод на экран подсказки происходит при вызове функции showHint()

void showHint()

{

cout << "0 - Добавить запись\n";

cout << "1 - Удалить запись\n";

cout << "2 - Найти записи по году поступления\n";

cout << "3 - Сортировать по имени\n";

cout << "4 - Записать данные в файл\n";

cout << "5 - Читать из файла\n";

cout << "6 - Изменить фамилию сотрудника\n";

cout << "7 - Изменить позицию сотрудника\n";

cout << "8 - Вывести список на экран\n";

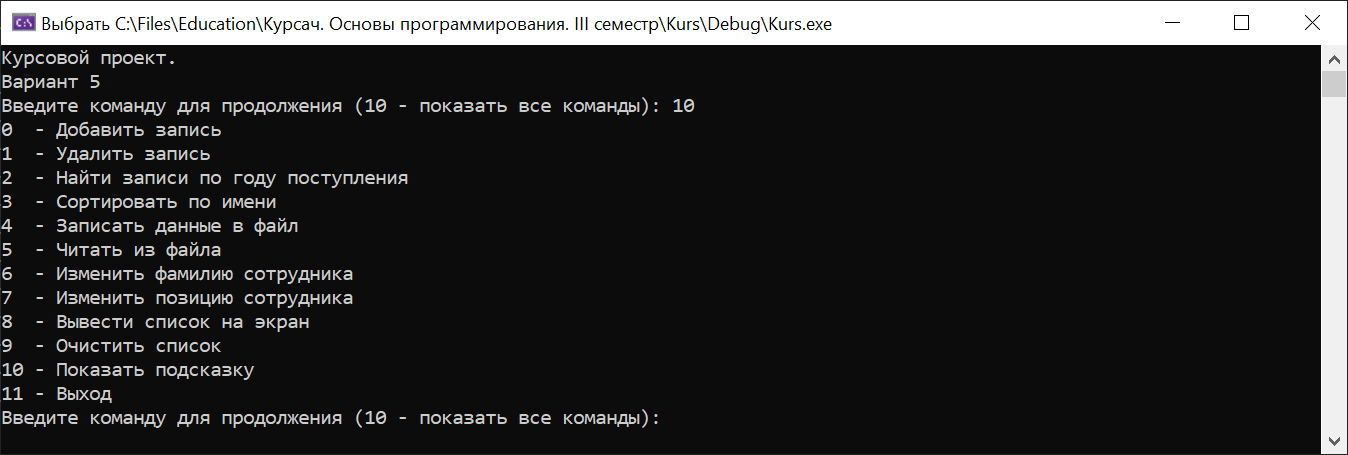
cout << "9 - Очистить список\n";

cout << "10 - Показать подсказку\n";

cout << "11 - Выход\n";

}

Пример выполнения:



* Добавление новой записи.

Добавление новой записи в список происходит вызовом функции addWorker(LinkList<Worker> \*). Функция принимает на вход указатель на список сотрудников и при выполнении приглашает пользователя ввести необходимые данные с подсказками об ограничениях ввода. В функции также реализована проверка вводимых строк на превышение длины и вводимых значений на соответствие типу.

void addWorker(LinkList<Worker>\* workers)

{

#undef max

char name[100];

char position[100];

int year = 0;

cout << "Введите имя и инициалы сотрудника (до 25 символов): ";

cin.getline(name, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

cout << "Введите позицию сотрудника (до 25 символов): ";

cin.getline(position, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(position, 25, '\n');

}

cout << "Введите год поступления на работу: ";

cin >> year;

while (!cin || cin.peek() != '\n') {

cin.clear();

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

cout << "Введено некорректное значение. Повторите ввод: ";

cin >> year;

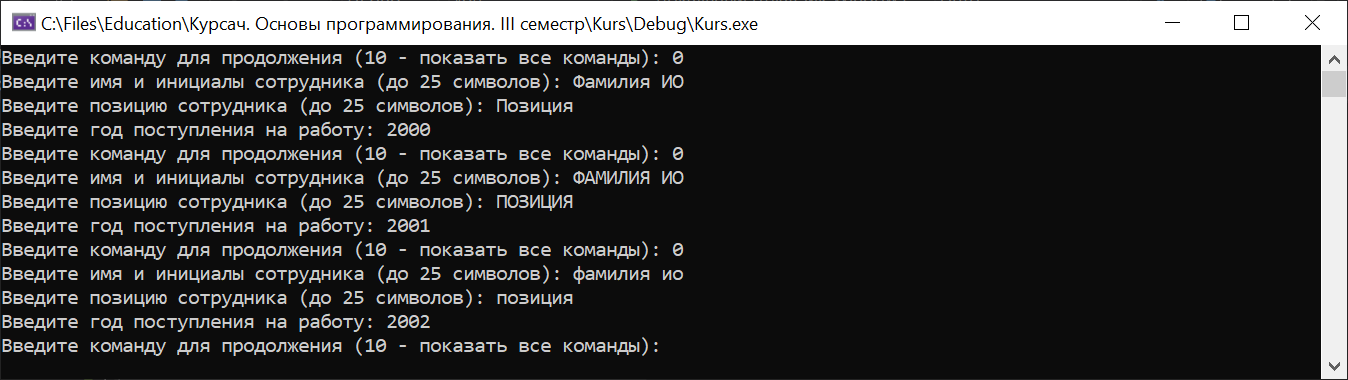
}

workers->insert\_before(workers->end(), Worker(name, position, year));

#define max

}

Пример выполнения:



* Вывод всех записей на экран.

Отображение всех записей на экране происходит вызовом функции showAll(). Функция выводит записи в виде таблицы.

void showAll(LinkList<Worker>\* li)

{

if (li->isEmpty()) {

cout << "Список сотрудников пуст\n";

return;

}

LinkList<Worker>::iterator iter = li->begin();

cout << setw(26) << "Фамилия и инициалы"

<< setw(26) << "Позиция"

<< setw(26) << "Год поступления на работу" << '\n';

while (iter != li->end()) {

cout << setw(26) << iter->name()

<< setw(26) << iter->position()

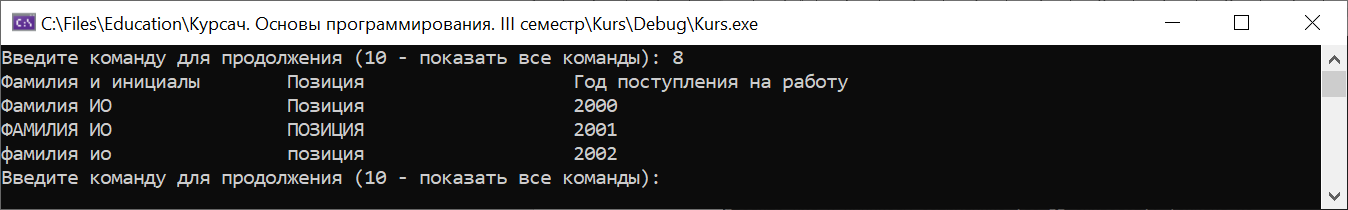
<< setw(26) << iter->yearOfEmployment() << '\n';

++iter;

}

}

Пример выполнения:



* Удаление записи.

Удаление записи происходит вызовом функции removeWorker(LinkList<Worker> \*). Функция принимает на вход указатель на список сотрудников и при выполнении приглашает пользователя ввести необходимые данные с подсказками об ограничениях ввода. Перед непосредственным удалением записи из контейнера, вызывается функция поиска записи по фамилии сотрудника \_\_findWorkerByName(LinkList<Worker>\*, const char\*).

void removeWorker(LinkList<Worker>\* workers)

{

#undef max

char name[100];

cout << "Введите имя и инициалы сотрудника (до 25 символов): ";

cin.getline(name, 25);

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

LinkList<Worker>::iterator it = \_\_findWorkerByName(workers, name);

if (!it) {

cout << "Работника с такой фамилией в списке нет\n";

return;

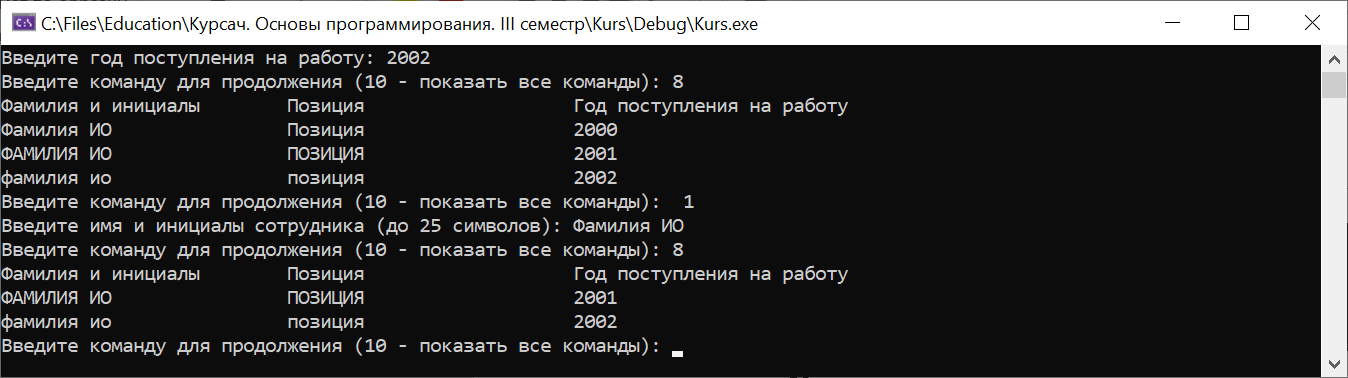
}

workers->erase(it);

#define max

}

Пример выполнения:



* Поиск записей по году поступления на работу

Поиск осуществляется вызовом функции findWorker(LinkList<Worker>\*, LinkList<Worker>\*, const int &). Функция принимает на вход исходный список, производит в нем поиск нужных записей и вставляет из в выходной список, который также передается на ее вход.

Приглашение ко вводу данных вынесено в отдельную функцию askYearForSearch(), которая проверяет введенное пользователем значение на соответствие типу и возвращает его. После отображения полученных строк на экране, список с найденными значениями удаляется.

Вывод полученного при поиске списка на экран выполняет функция respondOnSearchQuery(LinkList<Worker>\*), производящая форматирование вывода и проверяет было ли найдено хотя бы одна строка.

Найденные записи отображаются в виде таблицы.

void findWorker(const LinkList<Worker>\* workers, LinkList<Worker> \*outList, const int &year)

{

if (workers->isEmpty())

return;

LinkList<Worker>::iterator it = workers->begin();

while (it != workers->end()) {

if ((\*it).yearOfEmployment() == year)

outList->insert\_before(outList->end(), \*it);

++it;

}

}

const int askYearForSearch()

{

#undef max

int year = 0;

cout << "Введите год для поиска: ";

cin >> year;

while (!cin) {

cout << "Введено недопустимое значение. Повторите ввод: ";

cin.clear();

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

cin >> year;

}

return year;

#define max

}

void respondOnSearchQuery(LinkList<Worker>\* li)

{

if (li->isEmpty()) {

cout << "Такого работника в списке нет.\n";

return;

}

LinkList<Worker>::iterator iter = li->begin();

cout << "Найдены записи:\n";

cout << setw(26) << "Фамилия и инициалы"

<< setw(26) << "Позиция"

<< setw(26) << "Год поступления на работу" << '\n';

while (iter != li->end()) {

cout << setw(26) << iter->name()

<< setw(26) << iter->position()

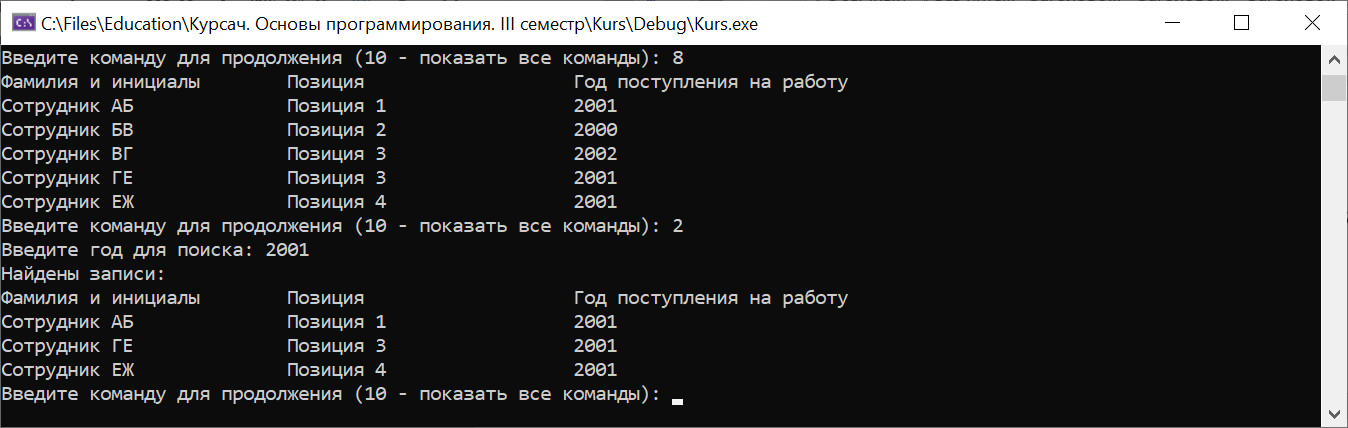
<< setw(26) << iter->yearOfEmployment() << '\n';

++iter;

}

}

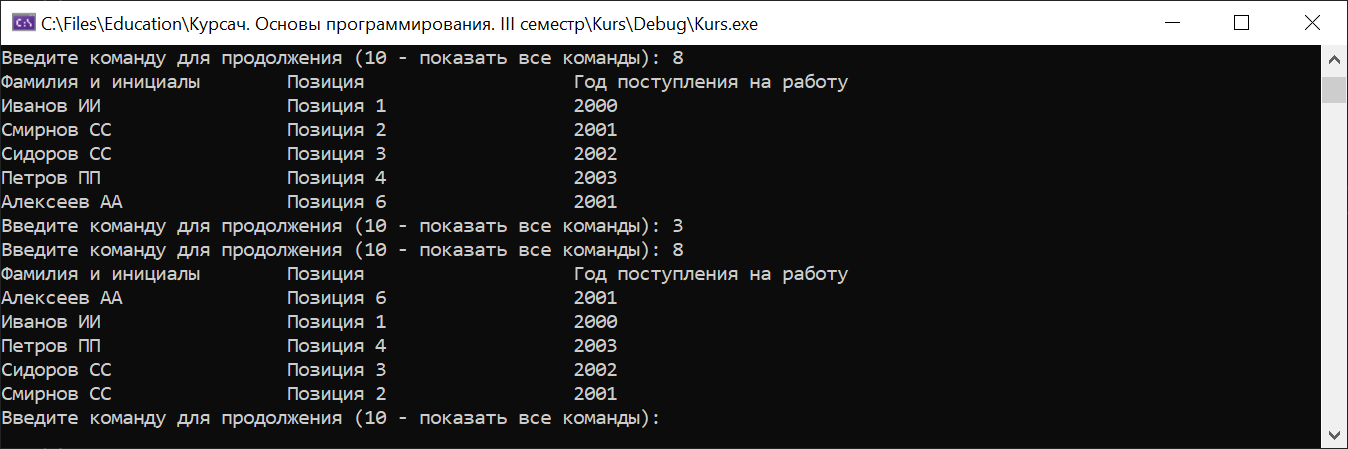
Пример выполнения:



* Сортировка

Сортировка списка выполняется вызовом метода sort() списка. Так как в классе Worker переопределены операции сравнения, дополнительного кода для реализации сортировки не требуется.

Пример выполнения:



* Запись в файл

Запись в файл осуществляется вызовом функции saveAsFile(LinkList<Worker>\*, const char\*). Функция принимает на вход исходный список и сохраняет его строки в текстовый файл по пути, который был передан функции. Функция возвращает логическое значение, означающее успешное или неудачное завершение записи в файл. Неудачная запись в файл происходит при неудачной попытке открыть файл, путь к которому был передан в функцию.

Приглашение на ввод пути к файлу вынесено в отдельную функцию askFilePath(const char\*), которая производит проверку на соответствие введенного пути разрешенной длине строки.

После сохранения списка в файл, если такой файл уже существовал, он будет перезаписан.

void askFilePath(char\* filePath)

{

#undef max

cout << "Введите полный путь к файлу (до 100 символов): ";

cin.getline(filePath, 100);

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(filePath, 25, '\n');

}

#define max

}

bool saveAsFile(const LinkList<Worker>\* workers, const char\* filePath)

{

fstream fs;

fs.open(filePath, ios\_base::out);

if (!fs.is\_open())

return false;

LinkList<Worker>::iterator iter = workers->begin();

while (iter != --workers->end()) {

fs << iter->name()

<< '|' << iter->position()

<< '|' << iter->yearOfEmployment() << '\n';

++iter;

}

fs << iter->name()

<< '|' << iter->position()

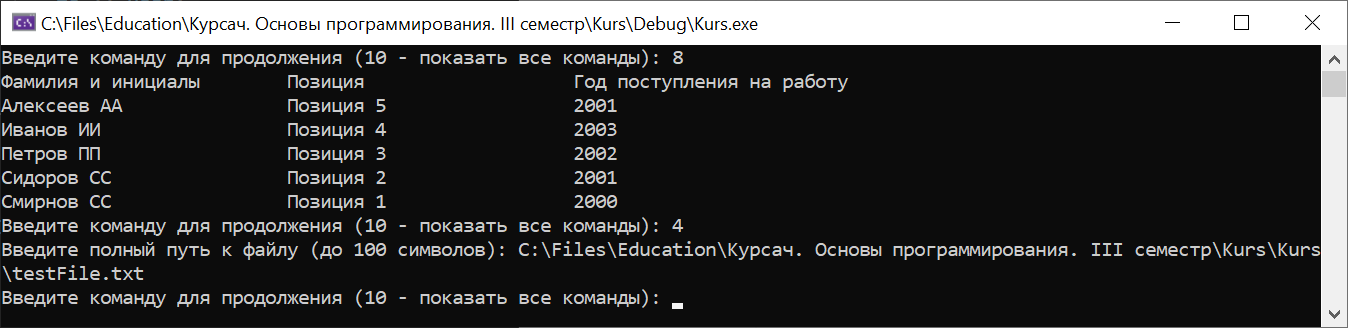
<< '|' << iter->yearOfEmployment();

fs.close();

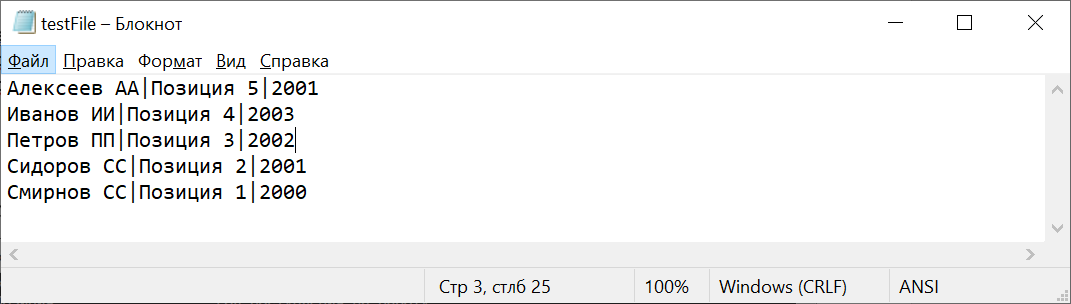
return true;

}

Пример выполнения:



Содержимое файла testFile.txt после выполнения программы:



* Чтение из файла

Чтение из файла осуществляется вызовом функции readFromFile(LinkList<Worker>\*, const char\*). Функция принимает на вход исходный список и добавляет в него новые записи из файла путь которого был передан функции. Функция возвращает логическое значение, означающее успешное или неудачное завершение чтения из файла. Неудачное чтение из файла происходит при неудачной попытке открыть файл, путь к которому был передан в функцию.

Приглашение на ввод пути к файлу вынесено в ту же функцию, что и запись в файл askFilePath(const char\*).

При чтении из файла, если список уже не пуст, старые значения удалены не будут.

bool readFromFile(LinkList<Worker>\* outList, const char\* filePath)

{

#undef max

char name[25]{};

char position[25]{};

int year = 0;

fstream fs;

fs.open(filePath, ios\_base::in);

if (!fs.is\_open())

return false;

while (!fs.eof()) {

fs.getline(name, 25, '|');

fs.getline(position, 25, '|');

fs >> year;

fs.ignore(1, '\n');

outList->insert\_before(outList->end(), Worker(name, position, year));

}

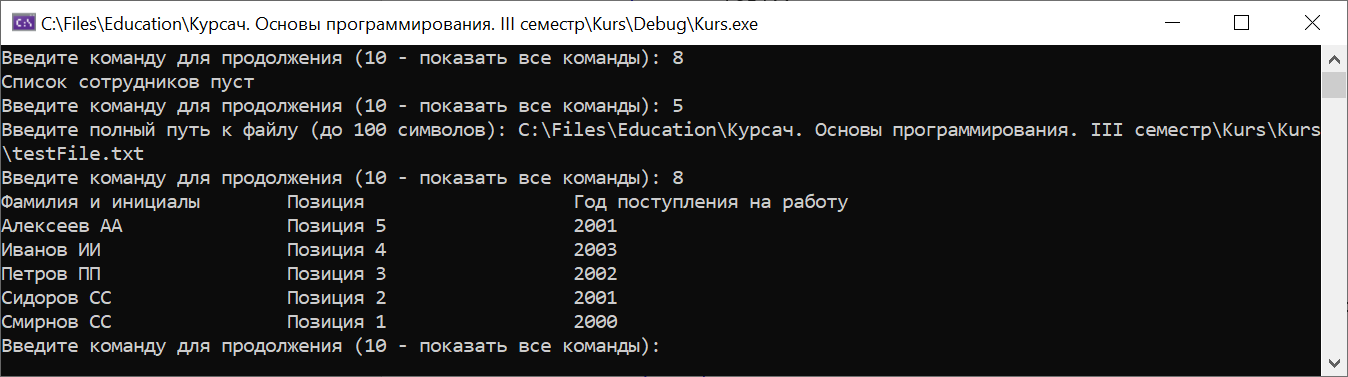
fs.close();

#define max

return true;

}

Пример выполнения:



* Редактирование данных

Редактирование данных производится в двух вариантах: редактирование фамилии и редактирование позиции сотрудника. При этом вызываются функции changeName(LinkList<Worker>\*, const char\*) и changePosition(LinkList<Worker>\*, const char\*) соответственно. Обе функции принимают на вход исходный список и имя сотрудника, для которого нужно внести изменения. Поиск записи в списке производится в функции \_\_findWorkerByName(LinkList<Worker>\*, const char\*), приведенной выше.

Приглашение на ввод имени искомого сотрудника вынесено в отдельную функцию askName(const char\*), которая производит проверку на соответствие введенного пути разрешенной длине строки.

void askName(char\* name)

{

#undef max

cout << "Введите имя сотрудника, данные которого нужно изменить: ";

cin.getline(name, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

#define max

}

void changeName(LinkList<Worker>\* workers, const char\* name)

{

#undef max

LinkList<Worker>::iterator iter = \_\_findWorkerByName(workers, name);

if (!iter) {

cout << "Работника с такой фамилией в списке нет\n";

return;

}

char newName[25];

cout << "Введите новые фамилию и инициалы (до 25 символов): ";

cin.getline(newName, 25, '\n');

if (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(newName, 25, '\n');

}

iter->setName(newName);

#define max

}

void changePosition(LinkList<Worker>\* workers, const char\* name)

{

#undef max

LinkList<Worker>::iterator iter = \_\_findWorkerByName(workers, name);

if (!iter) {

cout << "Работника с такой фамилией в списке нет\n";

return;

}

char pos[25];

cout << "Введите новую позицию (до 25 символов): ";

cin.getline(pos, 25, '\n');

if (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(pos, 25, '\n');

}

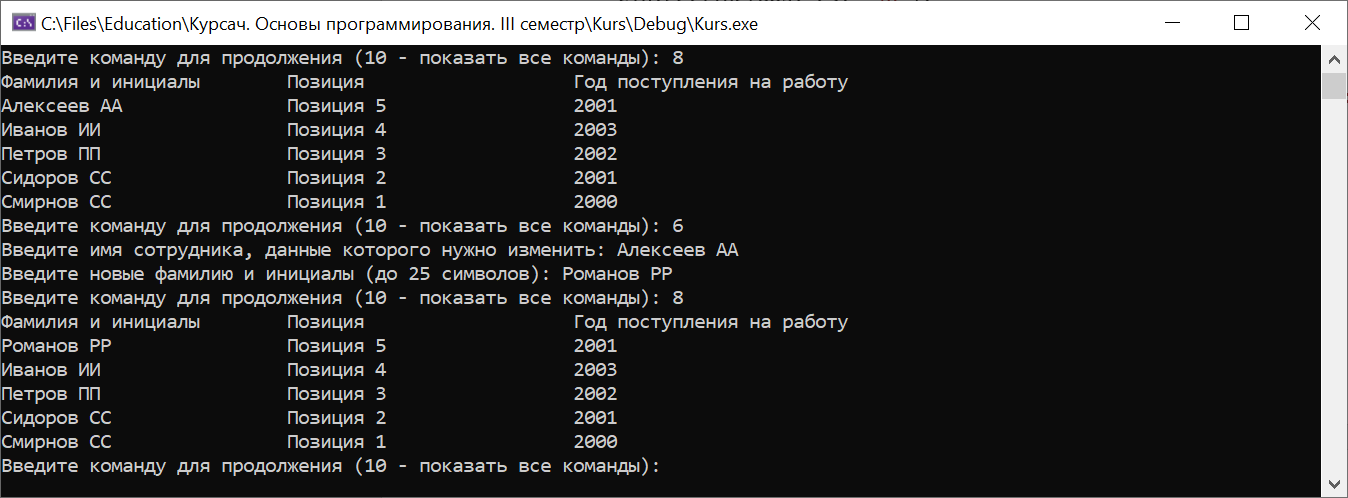
iter->setPosition(pos);

#define max

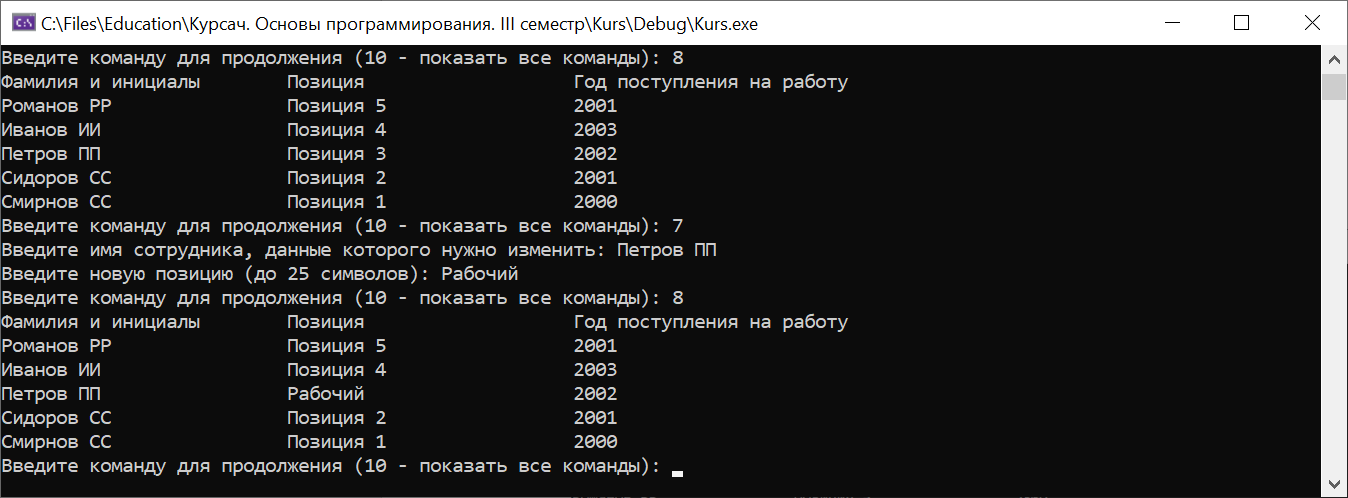
}

Пример выполнения:

- изменение имени:



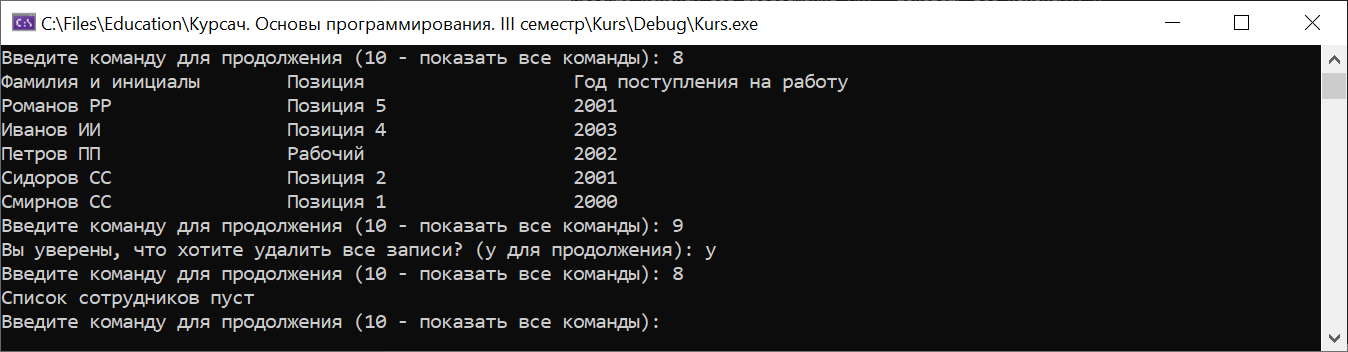
- изменение позиции:



* Очистка списка

Очистка списка производится вызовом метода списка clear(), но для того, чтобы дать пользователю возможность отменить очистку, была реализована отдельная функция, которая требует у пользователя подтверждения его намерений.

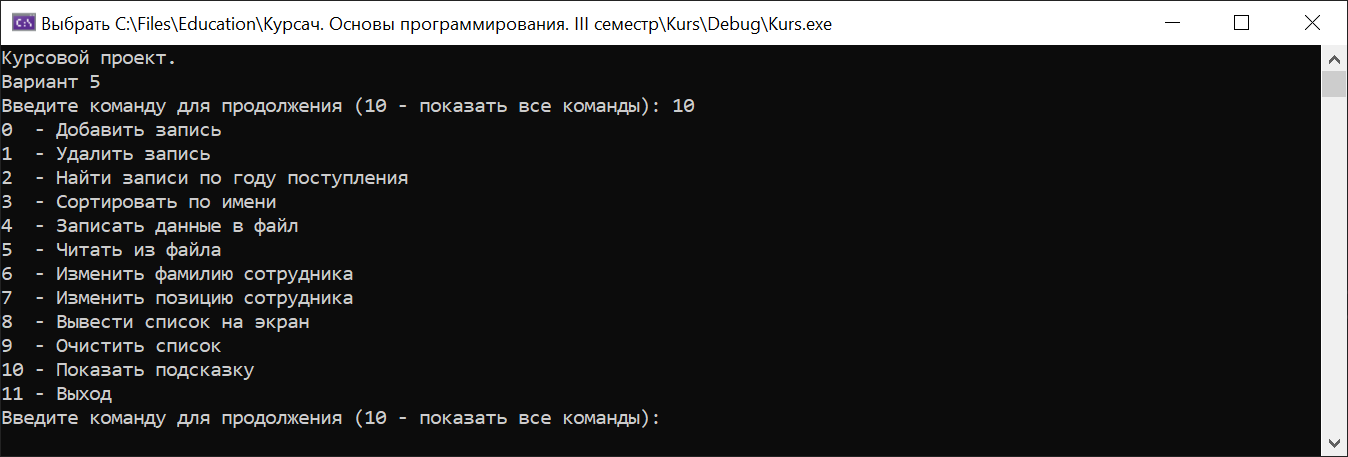
Пример выполнения:



Как можно заметить из кода программы (Приложение 1), для хранения данных выбран указатель на связный список. Это сделано для упрощения поиска утечек памяти: так как память выделяется только в связном списке, при верном ее удалении все утечки должны быть вне связного списка.

1. **Описание пользовательского интерфейса**

При запуске программы, чтобы не перегружать интерфейс, на экран выводится приглашение ко вводу команды с подсказкой для вызова меню всех команд. Меню всех команд представляет собой набор возможных команд с результатом их работы.



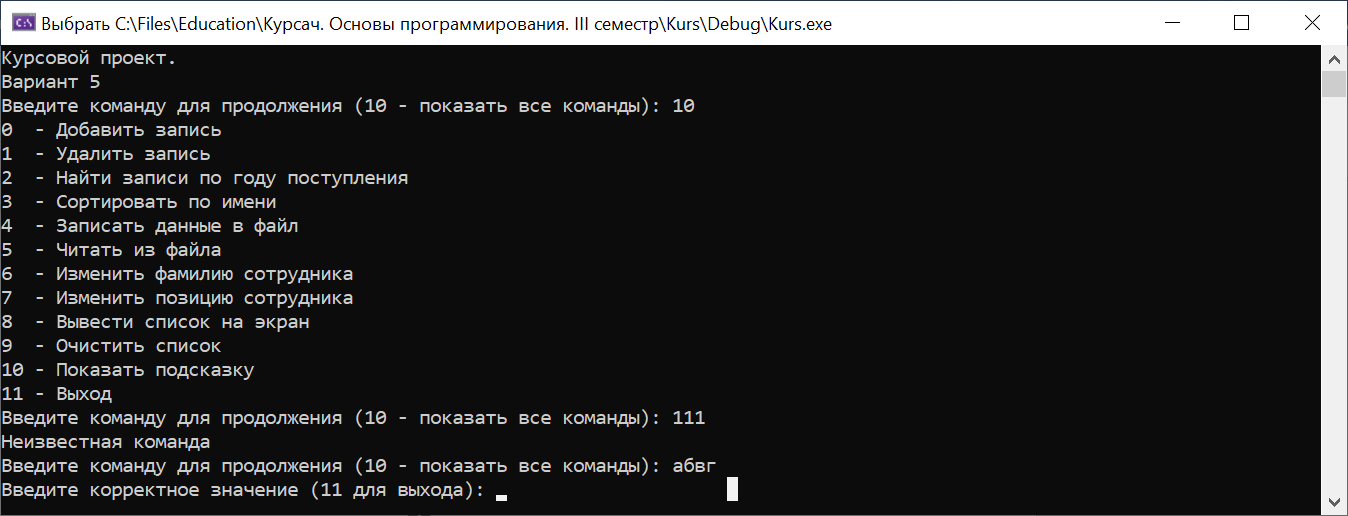
Выбор действия осуществляется вводом одной из предложенных команд.

При каждом вводе команды следует приглашение к дальнейшему вводу, которое соответствует введенной команде. При необходимости в тексте приглашения ко вводе есть подсказка для возможных вариантов действий.

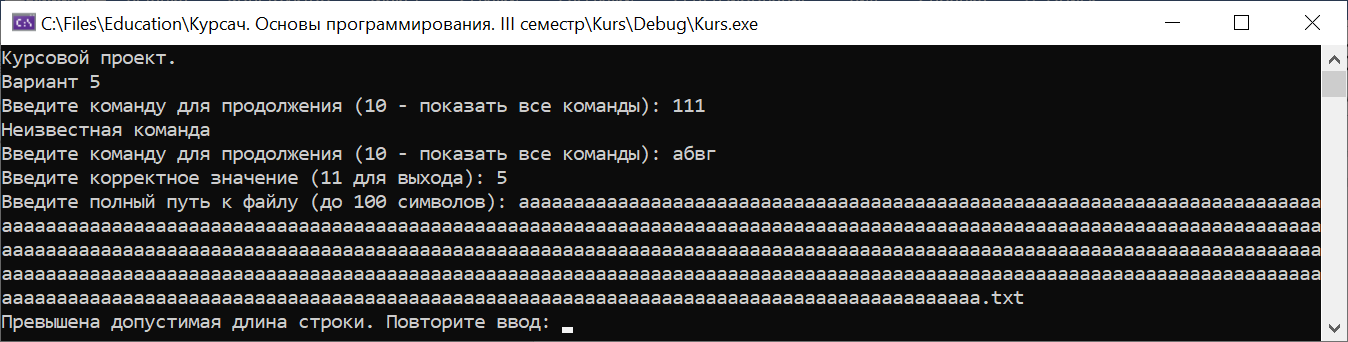
1. **Результаты тестирования**

Результаты тестирования приводятся ниже

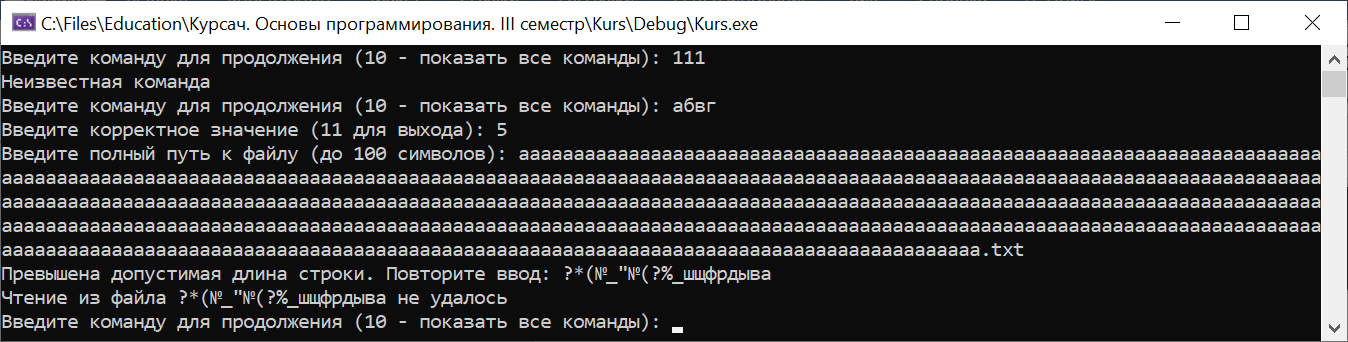
Результат выполнения программы при вводе неверной команды меню



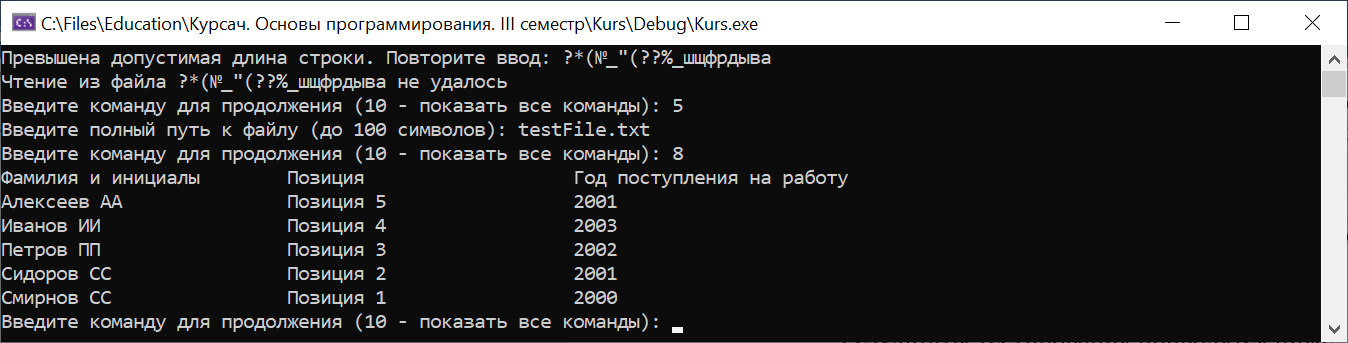
Результат выполнения чтения данных из файла при превышении допустимой длины файла



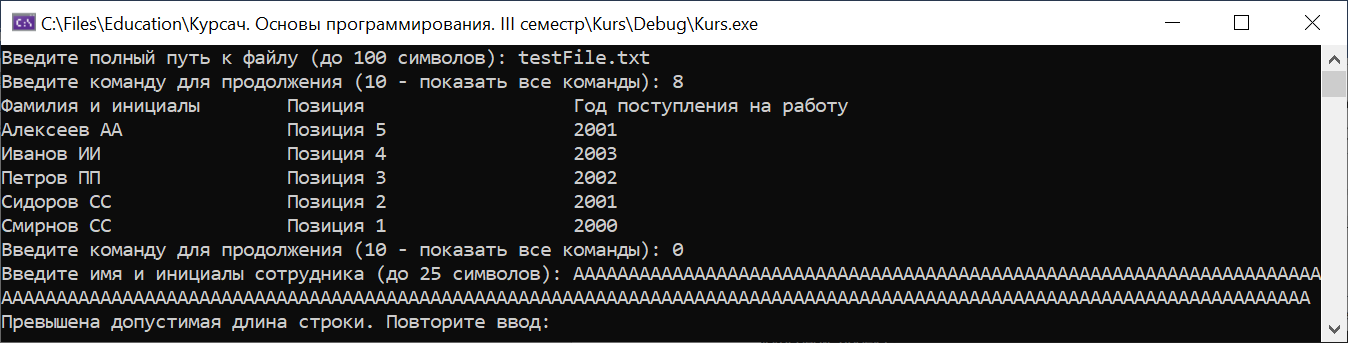
Результат выполнения чтения данных из файла при некорректном пути к файлу



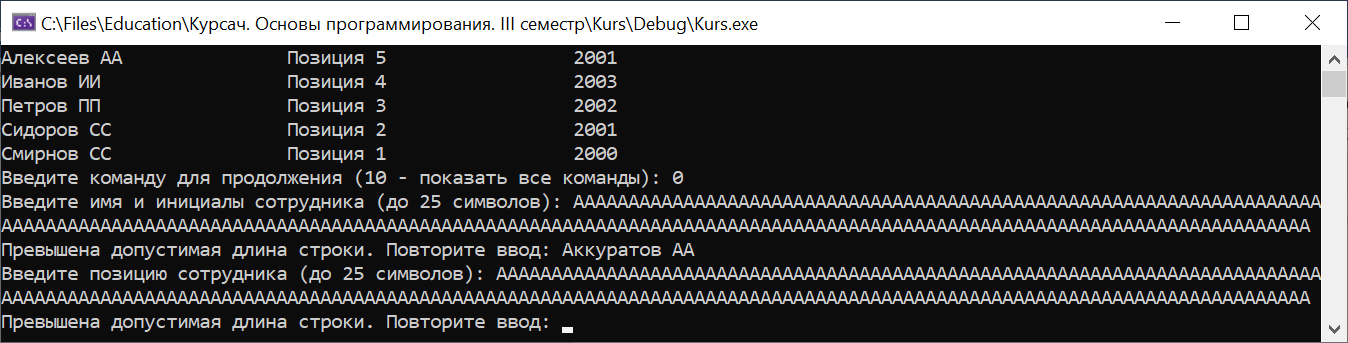
Результат выполнения чтения данных из файла при корректном пути к файлу, а также результат выполнения команды вывода списка на экран



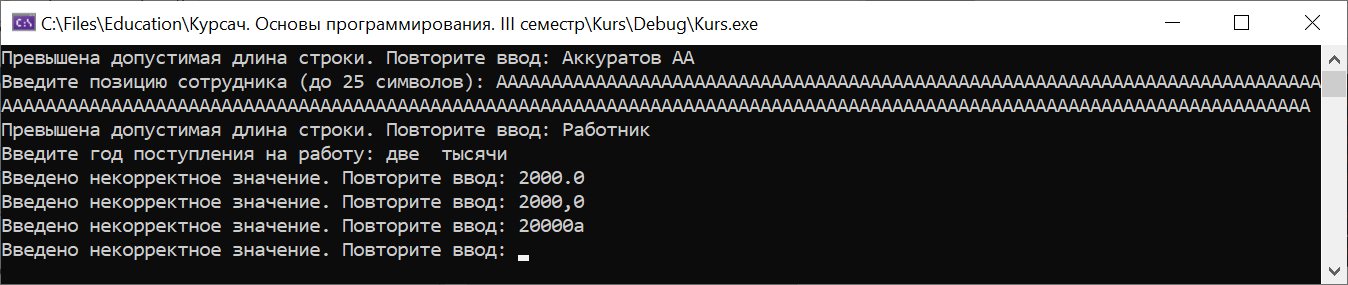
Результат выполнения добавления новой записи при превышении допустимой длины имени сотрудника.



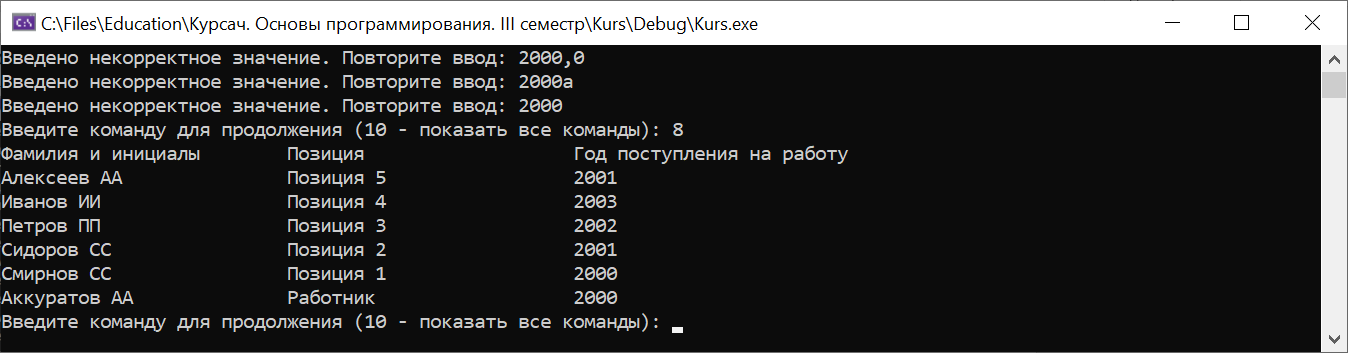
Результат выполнения добавления новой записи при превышении допустимой длины позиции сотрудника.



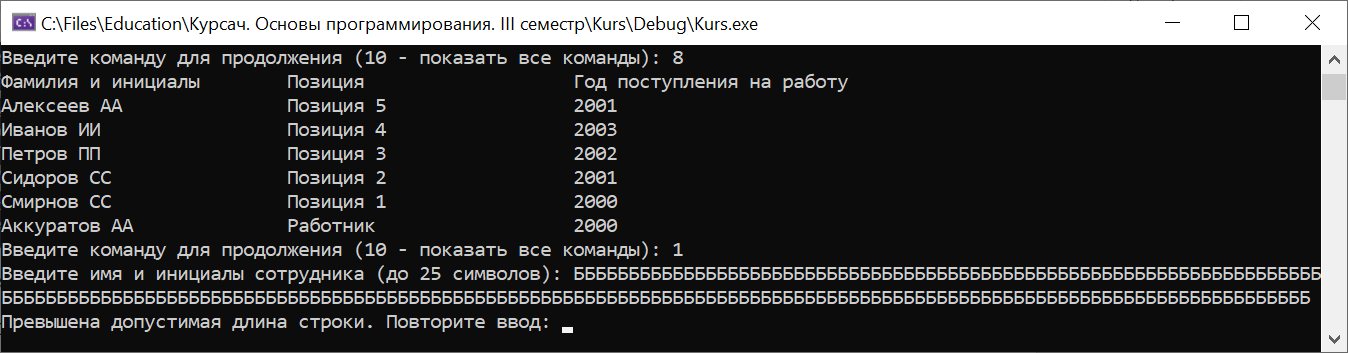
Результат выполнения добавления новой записи при некорректном вводе года вступления работника в должность.



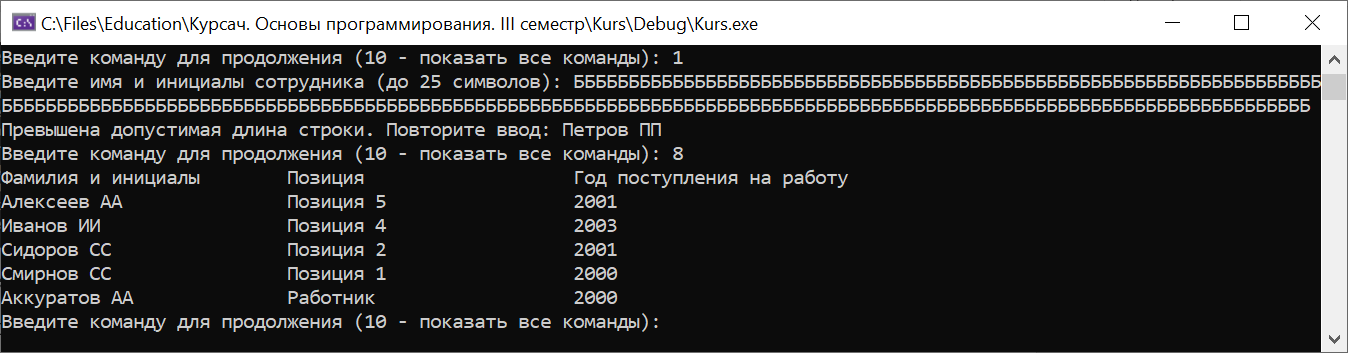
Результат выполнения добавления новой записи при корректных имени, позиции и годе приема на работу



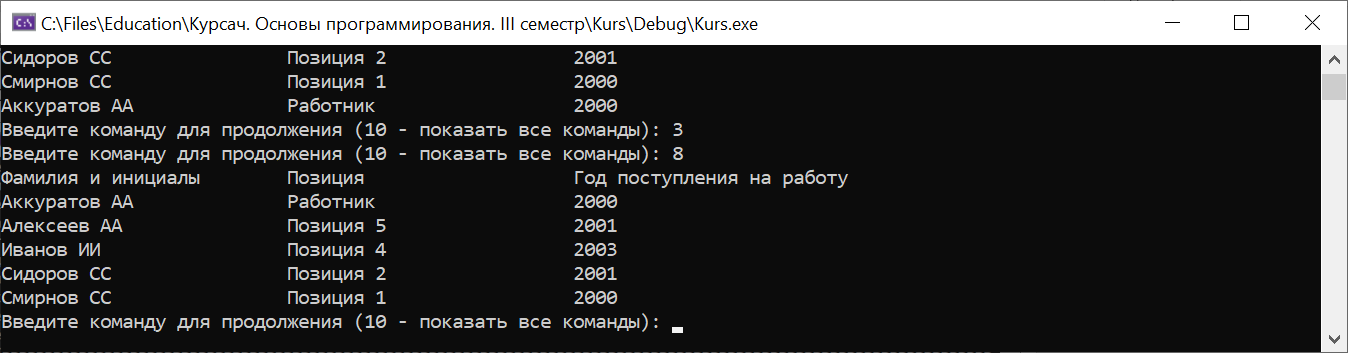
Результат выполнения удаления строки при превышении допустимой длины имени сотрудника.



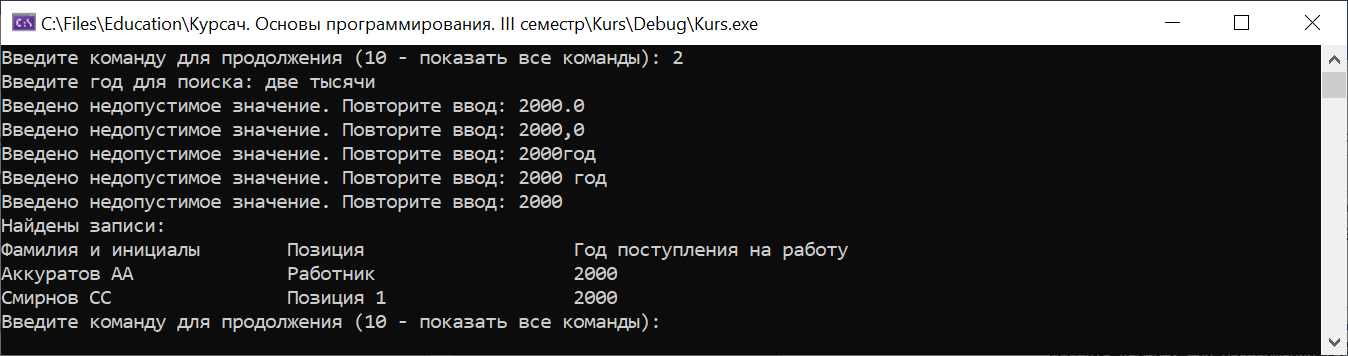
Результат выполнения удаления строки при корректном вводе имени сотрудника



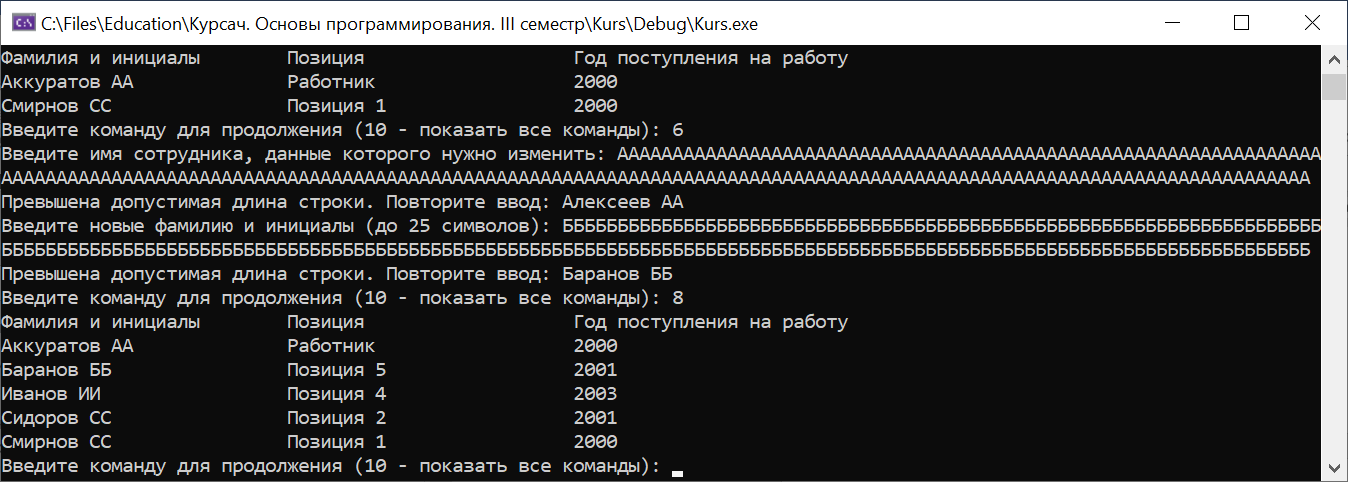
Результат выполнения сортировки по имени



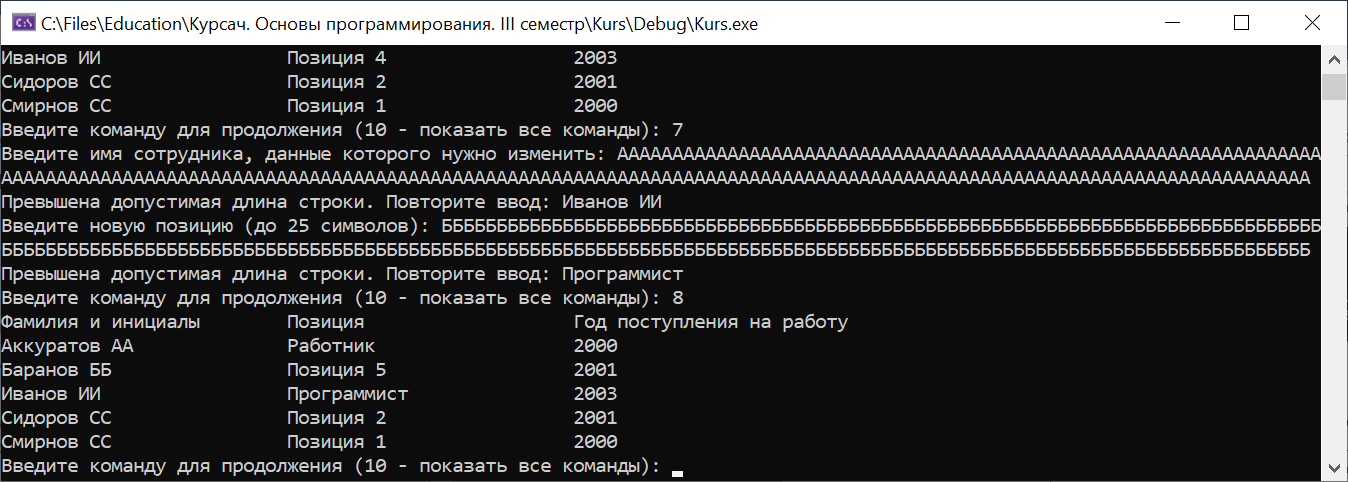
Результат выполнения поиска сотрудников по году приема на работу, при некорректном вводе года и далее при корректном вводе года



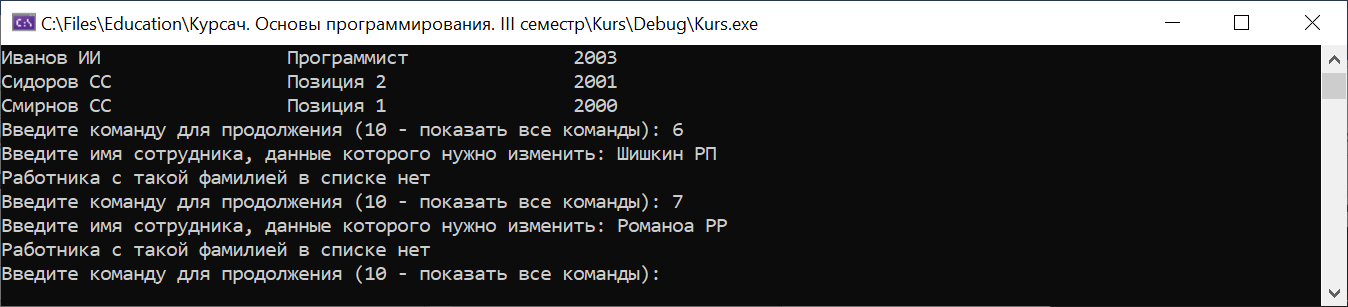
Результат выполнения изменения фамилии сотрудника при превышении допустимой длины искомой и новой фамилии и при корректно введенной фамилии



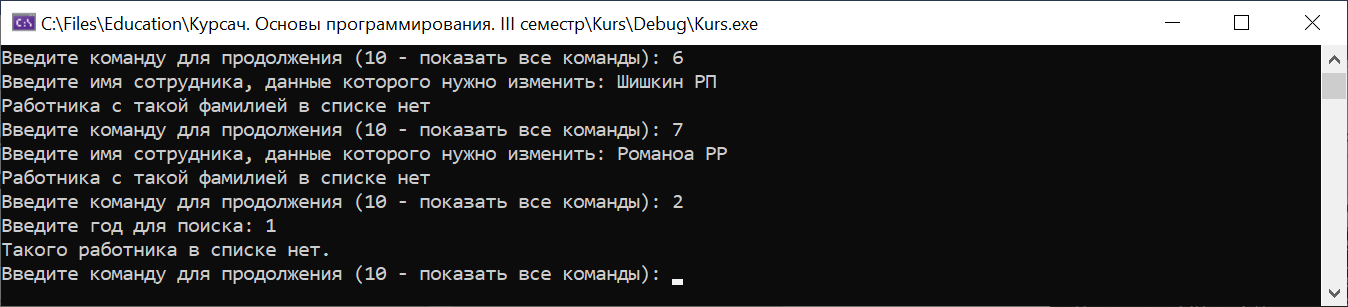
Результат выполнения изменения позиции сотрудника при превышении допустимой длины искомой фамилии и изменяемой позиции и при корректно введенных фамилии и позиции



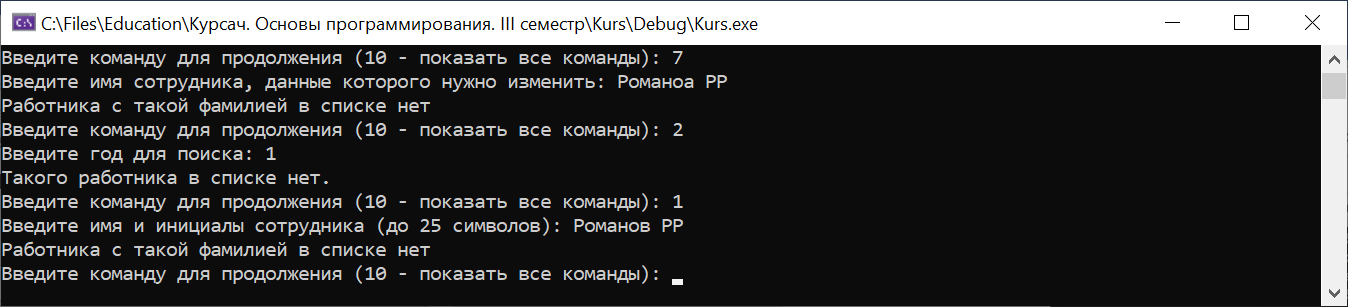
Результат выполнения изменения позиции и имени сотрудника при отсутствии найденных строк

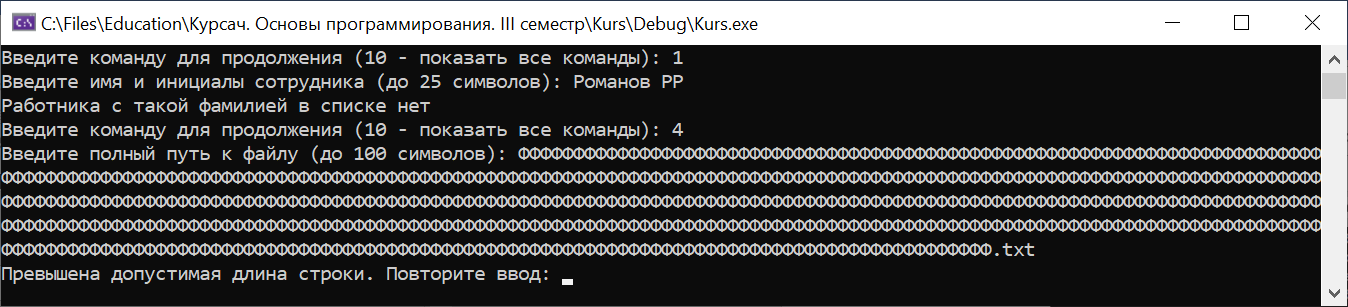


Результат выполнения поиска сотрудников по году приема на работу при отсутствии найденных строк

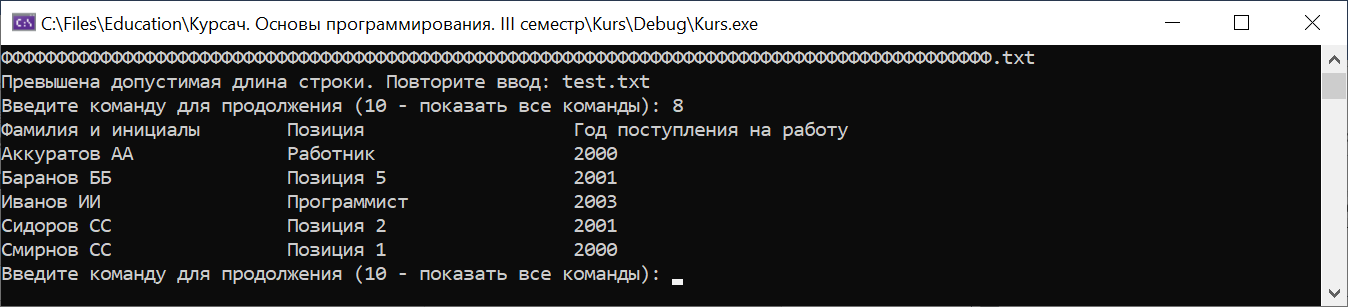


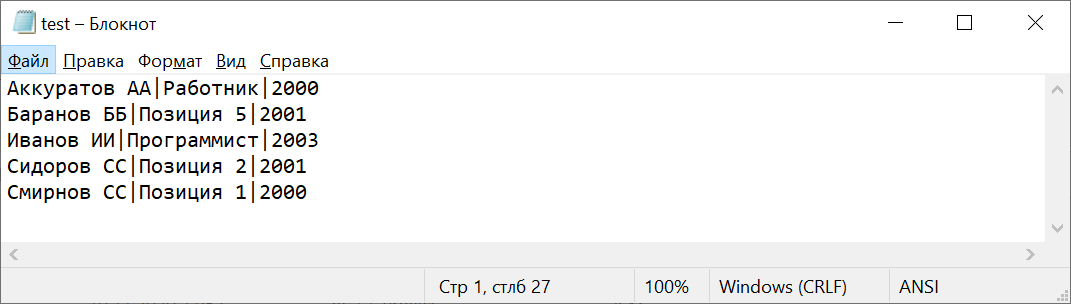
Результат выполнения удаления строки при отсутствии сотрудника в списке



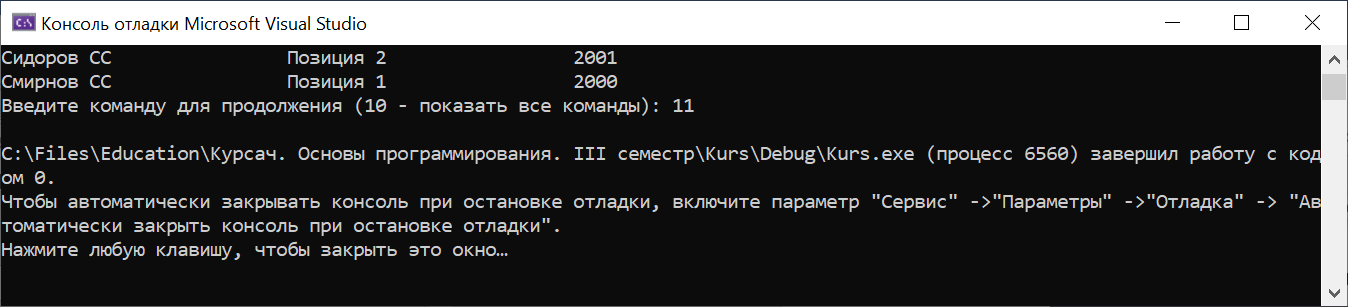
Результат выполнения записи в файл при превышении допустимой длины имени файла

Результат выполнения записи в файл при корректном имени файла





Программа выполнялась непрерывно в течении одной сессии. При завершении программы утечек памяти не выявлено



1. **Заключение**

В ходе выполнения курсового проекта была разработана программа для заданной предметной области «Отдел кадров сотрудников», которая позволяет вводить информацию, хранить еe в файле, осуществлять поиск, модификацию, сортировку и удаление данных. Для хранения данных при работе программы разработана структура данных двунаправленный связный список. Для хранения данных реализован класс Worker.

Программа имеет следующие достоинства:

- есть возможность сохранения данных в файл и их загрузки из файла для хранения данных при закрытии приложения

- есть возможность редактирования записей для исправления ошибок и дополнения данных

- есть возможность сортировки списка для упрощения ручного поиска в списке

- программа выполняет поставленную задачу правильно и без ошибок

- в программе выполняются всевозможные проверки на некорректность введенных значений для исключения ошибок

- все выполняемые программой функции выполнены в виде отдельных процедур, что облегчает введение в программу дополнительных возможностей.

Программа имеет следующие недостатки

- программа выполнена в виде консольного приложения, что усложняет понимание пользователем интерфейса

- хранение данных организовано в текстовом файле. Такая организация хранения заставляет выгружать все данные из хранилища в оперативную память, что при большом объеме данных увеличивает продолжительность выполнения процедуры

.

1. **Список используемой литературы**
2. Ключарев А.А., Матьяш В.А., Щекин С.В. Структуры и алгоритмы обработки данных: Учебное пособие / СПбГУАП. СПб., 2004.
3. Страуструп, Б. Язык программирования C++ специальное издание / Б. Страуструп ; пер.: С. Анисимов, М. Кононов ; ред.: Ф. Андреев, А. Ушаков. - [Б. м.] : Бином-Пресс, 2008. - 1098 с.
4. Кнут, Д. Искусство программирования [в 3 т.]. Т. 1. Основные алгоритмы / Д. Кнут ; ред. Ю. В. Козаченко. - 3-е изд. - М. : Вильямс, 2014. - 720 с.
5. Прата, Стивен Язык программирования С++. Лекции и упражнения, 6-е изд. Пер. с англ. – М.:ООО «И.Д. Вильямс», 2012. – 1248 с.: ил. - Парал. тит. англ.

# **Приложение 1**

Текст программы

## Файл main.cpp

#include <iostream>

#include <fstream>

#include <cstring>

#include <iomanip>

#include <Windows.h>

#include "LinkList.h"

#include "Worker.h"

using namespace std;

// проверка на утечки памяти

#define \_CRTDBG\_MAP\_ALLOC

#include <cstdlib>

#include <crtdbg.h>

#define DBG\_NEW new(\_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

#define new DBG\_NEW

//определения функций

//Функции чтения из стандартного порта

const int askYearForSearch();

void askFilePath(char \*filePath);

//Функции записи в стандартный порт

void showHint();

void respondOnSearchQuery(LinkList<Worker>\* li);

void showAll(LinkList<Worker>\* li);

void askName(char\* name);

//Пользовательские функции

short getCommand();

void addWorker(LinkList<Worker>\* workers);

void removeWorker(LinkList<Worker>\* workers);

void findWorker(const LinkList<Worker>\* workers, LinkList<Worker>\*li, const int& year);

bool readFromFile(LinkList<Worker>\* workers, const char\* filePath);

bool saveAsFile(const LinkList<Worker>\* workers, const char\* filePath);

void clearList(LinkList<Worker>\* workers);

void changeName(LinkList<Worker>\* workers, const char \*name);

void changePosition(LinkList<Worker>\* workers, const char\* name);

//Служебные функции

LinkList<Worker>::iterator \_\_findWorkerByName(LinkList<Worker>\* workers, const char\* name);

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout.setf(ios\_base::adjustfield, ios\_base::left);

cout << "Курсовой проект.\nВариант " << (81 - 1) % 19 + 1 << '\n';

short command = -1;

LinkList<Worker> \*workers = new LinkList<Worker>;

do {

command = getCommand();

switch (command)

{

case 0:

addWorker(workers);

break;

case 1:

removeWorker(workers);

break;

case 2: {

LinkList<Worker>\* li = new LinkList<Worker>;

findWorker(workers, li, askYearForSearch());

respondOnSearchQuery(li);

delete li;

break;

}

case 3:

workers->sort();

break;

case 4: {

char filePath[100];

askFilePath(filePath);

if (!saveAsFile(workers, filePath))

cout << "Запись в файл " << filePath << " не удалась\n";

break;

}

case 5: {

char filePath[100];

askFilePath(filePath);

if (!readFromFile(workers, filePath))

cout << "Чтение из файла " << filePath << " не удалось\n";

break;

}

case 6: {

char name[25];

askName(name);

changeName(workers, name);

break;

}

case 7: {

char name[25];

askName(name);

changePosition(workers, name);

break;

}

case 8:

showAll(workers);

break;

case 9:

clearList(workers);

break;

case 10:

showHint();

break;

case 11:

break;

default:

cout << "Неизвестная команда\n";

break;

}

} while (command != 11);

delete workers;

// Для обнаружения утечек памяти

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT);

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

return 0;

}

short getCommand()

{

#undef max

short com;

cout << "Введите команду для продолжения (10 - показать все команды): ";

cin >> com;

while (!cin) {

cin.clear();

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

cout << "Введите корректное значение (11 для выхода): ";

cin >> com;

}

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

return com;

#define max

}

void addWorker(LinkList<Worker>\* workers)

{

#undef max

char name[100];

char position[100];

int year = 0;

cout << "Введите имя и инициалы сотрудника (до 25 символов): ";

cin.getline(name, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

cout << "Введите позицию сотрудника (до 25 символов): ";

cin.getline(position, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(position, 25, '\n');

}

cout << "Введите год поступления на работу: ";

cin >> year;

while (!cin || cin.peek() != '\n') {

cin.clear();

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

cout << "Введено некорректное значение. Повторите ввод: ";

cin >> year;

}

workers->insert\_before(workers->end(), Worker(name, position, year));

#define max

}

void removeWorker(LinkList<Worker>\* workers)

{

#undef max

char name[100];

cout << "Введите имя и инициалы сотрудника (до 25 символов): ";

cin.getline(name, 25);

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

LinkList<Worker>::iterator it = \_\_findWorkerByName(workers, name);

if (!it) {

cout << "Работника с такой фамилией в списке нет\n";

return;

}

workers->erase(it);

#define max

}

void findWorker(const LinkList<Worker>\* workers, LinkList<Worker> \*outList, const int &year)

{

if (workers->isEmpty())

return;

LinkList<Worker>::iterator it = workers->begin();

while (it != workers->end()) {

if ((\*it).yearOfEmployment() == year)

outList->insert\_before(outList->end(), \*it);

++it;

}

}

const int askYearForSearch()

{

#undef max

int year = 0;

cout << "Введите год для поиска: ";

cin >> year;

while (!cin || cin.peek() != '\n') {

cout << "Введено недопустимое значение. Повторите ввод: ";

cin.clear();

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

cin >> year;

}

return year;

#define max

}

bool readFromFile(LinkList<Worker>\* outList, const char\* filePath)

{

#undef max

char name[25]{};

char position[25]{};

int year = 0;

fstream fs;

fs.open(filePath, ios\_base::in);

if (!fs.is\_open())

return false;

while (!fs.eof()) {

fs.getline(name, 25, '|');

fs.getline(position, 25, '|');

fs >> year;

fs.ignore(1, '\n');

outList->insert\_before(outList->end(), Worker(name, position, year));

}

fs.close();

#define max

return true;

}

bool saveAsFile(const LinkList<Worker>\* workers, const char\* filePath)

{

fstream fs;

fs.open(filePath, ios\_base::out);

if (!fs.is\_open())

return false;

LinkList<Worker>::iterator iter = workers->begin();

while (iter != --workers->end()) {

fs << iter->name()

<< '|' << iter->position()

<< '|' << iter->yearOfEmployment() << '\n';

++iter;

}

fs << iter->name()

<< '|' << iter->position()

<< '|' << iter->yearOfEmployment();

fs.close();

return true;

}

void clearList(LinkList<Worker>\* workers)

{

#undef max

if (workers->isEmpty())

return;

char com;

cout << "Вы уверены, что хотите удалить все записи? (y для продолжения): ";

cin >> com;

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

if (com != 'y')

return;

workers->clear();

#define max

}

void changeName(LinkList<Worker>\* workers, const char\* name)

{

#undef max

LinkList<Worker>::iterator iter = \_\_findWorkerByName(workers, name);

if (!iter) {

cout << "Работника с такой фамилией в списке нет\n";

return;

}

char newName[25];

cout << "Введите новые фамилию и инициалы (до 25 символов): ";

cin.getline(newName, 25, '\n');

if (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(newName, 25, '\n');

}

iter->setName(newName);

#define max

}

void changePosition(LinkList<Worker>\* workers, const char\* name)

{

#undef max

LinkList<Worker>::iterator iter = \_\_findWorkerByName(workers, name);

if (!iter) {

cout << "Работника с такой фамилией в списке нет\n";

return;

}

char pos[25];

cout << "Введите новую позицию (до 25 символов): ";

cin.getline(pos, 25, '\n');

if (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(pos, 25, '\n');

}

iter->setPosition(pos);

#define max

}

LinkList<Worker>::iterator \_\_findWorkerByName(LinkList<Worker>\* workers, const char\* name)

{

LinkList<Worker>::iterator iter = workers->begin();

while (iter != workers->end()) {

if (!strcmp(name, iter->name()))

return iter;

++iter;

}

return LinkList<Worker>::iterator(nullptr);

}

void askFilePath(char\* filePath)

{

#undef max

cout << "Введите полный путь к файлу (до 100 символов): ";

cin.getline(filePath, 100);

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(filePath, 100, '\n');

}

#define max

}

void showHint()

{

cout << "0 - Добавить запись\n";

cout << "1 - Удалить запись\n";

cout << "2 - Найти записи по году поступления\n";

cout << "3 - Сортировать по имени\n";

cout << "4 - Записать данные в файл\n";

cout << "5 - Читать из файла\n";

cout << "6 - Изменить фамилию сотрудника\n";

cout << "7 - Изменить позицию сотрудника\n";

cout << "8 - Вывести список на экран\n";

cout << "9 - Очистить список\n";

cout << "10 - Показать подсказку\n";

cout << "11 - Выход\n";

}

void respondOnSearchQuery(LinkList<Worker>\* li)

{

if (li->isEmpty()) {

cout << "Такого работника в списке нет.\n";

return;

}

LinkList<Worker>::iterator iter = li->begin();

cout << "Найдены записи:\n";

cout << setw(26) << "Фамилия и инициалы"

<< setw(26) << "Позиция"

<< setw(26) << "Год поступления на работу" << '\n';

while (iter != li->end()) {

cout << setw(26) << iter->name()

<< setw(26) << iter->position()

<< setw(26) << iter->yearOfEmployment() << '\n';

++iter;

}

}

void showAll(LinkList<Worker>\* li)

{

if (li->isEmpty()) {

cout << "Список сотрудников пуст\n";

return;

}

LinkList<Worker>::iterator iter = li->begin();

cout << setw(26) << "Фамилия и инициалы"

<< setw(26) << "Позиция"

<< setw(26) << "Год поступления на работу" << '\n';

while (iter != li->end()) {

cout << setw(26) << iter->name()

<< setw(26) << iter->position()

<< setw(26) << iter->yearOfEmployment() << '\n';

++iter;

}

}

void askName(char\* name)

{

#undef max

cout << "Введите имя сотрудника, данные которого нужно изменить: ";

cin.getline(name, 25, '\n');

while (!cin) {

cin.clear();

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

cout << "Превышена допустимая длина строки. Повторите ввод: ";

cin.getline(name, 25, '\n');

}

#define max

}

## Файл Worker.h

#pragma once

#include <cstring>

class Worker

{

public:

Worker(char\* name, char\* position, const int& year);

Worker(const Worker& worker);

bool operator==(const Worker& rightVal);

bool operator>(const Worker& rightVal);

bool operator<(const Worker& rightVal);

bool operator!=(const Worker& rightVal);

Worker& operator=(const Worker& rightVal);

const char\* name() const;

const char\* position() const;

const int yearOfEmployment() const;

void setName(const char\* name);

void setPosition(const char\* position);

private:

//Максимальная длина строк в классе

static const int m\_maxStringSize = 25;

//Строки представлен как char (а не const char),

//чтобы была возможность вызывать стандартную функцию копирования строк

char m\_name[m\_maxStringSize];

char m\_position[m\_maxStringSize];

int m\_yearOfEmployment;

};

## Файл Worker.cpp

#include "Worker.h"

Worker::Worker(char\* name, char\* position, const int& year)

: m\_yearOfEmployment(year)

{

std::strcpy(m\_name, name);

std::strcpy(m\_position, position);

}

Worker::Worker(const Worker& rightVal)

{

std::strcpy(m\_name, rightVal.name());

std::strcpy(m\_position, rightVal.position());

m\_yearOfEmployment = rightVal.yearOfEmployment();

}

bool Worker::operator==(const Worker& rightVal)

{

return !std::strcmp(m\_name, rightVal.name());

}

bool Worker::operator>(const Worker& rightVal)

{

return std::strcmp(m\_name, rightVal.name()) > 0;

}

bool Worker::operator<(const Worker& rightVal)

{

return std::strcmp(m\_name, rightVal.name()) < 0;

}

bool Worker::operator!=(const Worker& rightVal)

{

return !operator==(rightVal);

}

Worker& Worker::operator=(const Worker& rightVal)

{

if (this == &rightVal)

return \*this;

std::strcpy(m\_name, rightVal.name());

std::strcpy(m\_position, rightVal.position());

m\_yearOfEmployment = rightVal.m\_yearOfEmployment;

return \*this;

}

const char\* Worker::name() const

{

return m\_name;

}

const char\* Worker::position() const

{

return m\_position;

}

const int Worker::yearOfEmployment() const

{

return m\_yearOfEmployment;

}

void Worker::setName(const char\* name)

{

std::strcpy(m\_name, name);

}

void Worker::setPosition(const char\* position)

{

std::strcpy(m\_position, position);

}

## Файл LinkList.h

#pragma once

#include <iostream>

template<typename T> class LinkList;

template<typename Node> class ListIterator;

//Шаблонный класс для хранения элементов и их указателей

template<typename T>

class node

{

public:

using Value = typename T;

node() : m\_data(nullptr), m\_next(nullptr), m\_previous(nullptr) {}

node(Value\* data)

: m\_data(data)

, m\_next(nullptr)

, m\_previous(nullptr)

{}

node(Value\* data, node\* next, node\* previous)

: m\_data(data)

, m\_next(next)

, m\_previous(previous)

{}

~node() { delete m\_data; }

Value\* data() const { return m\_data; }

node\* next() const { return m\_next; }

node\* previous() const { return m\_previous; }

void setData(Value\* data) { m\_data = data; }

void setNext(node\* next) { m\_next = next; }

void setPrevious(node\* previous) { m\_previous = previous; }

private:

node\* m\_next;

node\* m\_previous;

Value\* m\_data;

};

//Класс итератора для упращения обхода элементов

template<typename Node>

class ListIterator

{

using ValueType = typename Node::Value;

public:

ListIterator(Node\* data = nullptr) : m\_ptr(data) {}

ValueType& operator\*() { return \*m\_ptr->data(); }

ValueType\* operator->() { return m\_ptr->data(); }

bool operator==(const ListIterator& rightIter) { return m\_ptr == rightIter.m\_ptr; }

bool operator!=(const ListIterator& rightIter) { return !operator==(rightIter); }

bool operator==(Node\* rightPtr) { return m\_ptr == rightPtr; }

bool operator!=(Node\* rightPtr) { return !operator==(rightPtr); }

bool operator!() { return !m\_ptr; }

operator ValueType\* () { return m\_ptr->data(); }

ListIterator operator++(int)

{

if (!m\_ptr->next())

return \*this;

ListIterator temp = \*this;

++(\*this);

return temp;

}

ListIterator operator++()

{

if (!m\_ptr->next())

return \*this;

m\_ptr = m\_ptr->next();

return \*this;

}

ListIterator operator--()

{

if (!m\_ptr)

return \*this;

m\_ptr = m\_ptr->previous();

return \*this;

}

ListIterator operator--(int)

{

if (!m\_ptr->previous())

return \*this;

ListIterator temp = \*this;

--(\*this);

return temp;

}

//Чтобы не добавлять класс как дружественный в LinkList<T>, функция объявлена как public

//Это избавляет от лишней зависимости и позволяет использовать итератор с другими списками

Node\* \_\_getPtr() const { return m\_ptr; }

private:

Node\* m\_ptr;

};

template<typename T>

class LinkList

{

using Node = node<T>;

friend class Node;

public:

using iterator = ListIterator<Node>;

LinkList() : m\_begin(nullptr), m\_end(nullptr), m\_size(0) {}

LinkList(const LinkList<T>& li)

{

LinkList<T>::iterator it = li.begin();

LinkList<T>::iterator thisIt = begin();

while (it != li.end())

insert(thisIt++, \*it++);

}

~LinkList()

{

clear();

delete m\_end;

}

int size() const { return m\_size; }

bool isEmpty() const { return !m\_size; }

iterator begin() const { return iterator(m\_begin); }

iterator end() const { return iterator(m\_end); }

LinkList<T>& operator=(const LinkList<T>& li)

{

if (this == &li)

return \*this;

LinkList<T>::iterator it = li.begin();

LinkList<T>::iterator thisIt = begin();

while (it != li.end())

insert(thisIt++, \*it++);

return \*this;

}

bool operator==(const LinkList<T>& li)

{

if (li.size() != size())

return false;

LinkList<T>::iterator it = li.begin();

LinkList<T>::iterator thisIt = begin();

while (it != li.end())

if (\*it++ != \*thisIt++)

return false;

return true;

}

/\*

\* Вставляет элемент после итераторв pos

\* Возвращает итератор на вставленный элемент

\*/

iterator insert(iterator pos, const T& elem)

{

//Вставка в пустой список

if (!pos) {

if (!isEmpty())

return pos;

m\_begin = new Node(new T(elem), nullptr, nullptr);

m\_end = new Node(nullptr, nullptr, m\_begin);

m\_begin->setNext(m\_end);

++m\_size;

return begin();

}

Node\* posPtr = pos.\_\_getPtr();

//Вставка в конец списка

if (posPtr->next() == m\_end) {

posPtr->setNext(new Node(new T(elem), m\_end, posPtr));

m\_end->setPrevious(posPtr->next());

++m\_size;

return end();

}

//Вставка нового элемена

posPtr->setNext(new Node(new T(elem), posPtr->next(), posPtr));

++m\_size;

return iterator(posPtr->next());

}

/\*

\* Вставляет элемент перед итератором pos

\* Возвращает итератор на вставленный элемент

\*/

iterator insert\_before(iterator pos, const T& elem)

{

//Вставка в пустой список

if (!pos)

return insert(pos, elem);

if (pos != begin())

return insert(iterator(pos.\_\_getPtr()->previous()), elem);

Node \*newBegin = new Node(new T(elem), m\_begin, nullptr);

m\_begin->setPrevious(newBegin);

m\_begin = newBegin;

return iterator(newBegin);

}

/\*

\* Удаляет элемент, на который указывает итератор pos

\* Возвращает итератор на элемент после удаляемого

\*/

iterator erase(iterator pos)

{

if (pos == end())

return pos;

//Удаление первого элемента

if (pos == begin()) {

Node\* newBegin = m\_begin->next();

newBegin->setPrevious(nullptr);

delete m\_begin;

m\_begin = newBegin;

--m\_size;

return begin();

}

//удаление элемента

Node\* nextPtr = pos.\_\_getPtr()->next();

Node\* prevPtr = pos.\_\_getPtr()->previous();

nextPtr->setPrevious(prevPtr);

prevPtr->setNext(nextPtr);

delete pos.\_\_getPtr();

--m\_size;

return iterator(nextPtr);

}

void clear()

{

iterator it = begin();

while (!isEmpty())

erase(it++);

}

/\*

\* Линейный поиск элемента elem

\* Возвращает итератор на найденный элемент

\* Если такого элемента нет, возвращает итератор на начало списка

\*/

iterator find(const T& elem)

{

if (isEmpty())

return begin();

if (elem == \*begin())

return begin();

iterator it = begin();

while (it != end()) {

if(\*it == elem)

return it;

++it;

}

return end();

}

/\*

\* Сортирует списко в соответствии с флагом order

\* true - по возрастанию

\* false - по убыванию

\* Тип T должен быть простым или иметь прераторы сравнения

\*

\* Выбрана сортировка перестановками, так как нет произвольного доступа к жлементам

\*/

void sort()

{

if (isEmpty())

return;

Node\* ptr = m\_begin;

Node \*cmpPtr = m\_begin->next();

while (ptr != m\_end) {

while (cmpPtr != m\_end) {

if (\*ptr->data() > \*cmpPtr->data()) {

\_\_swap(ptr, cmpPtr);

ptr = cmpPtr;

}

cmpPtr = cmpPtr->next();

}

ptr = ptr->next();

cmpPtr = ptr->next();

}

}

private:

void \_\_swap(Node\* lPtr, Node\* rPtr)

{

if (lPtr->next() == rPtr) {

Node\* tempNext = rPtr->next();

Node\* tempPrev = lPtr->previous();

tempNext->setPrevious(lPtr);

if (tempPrev)

tempPrev->setNext(rPtr);

else

m\_begin = rPtr;

rPtr->setNext(lPtr);

rPtr->setPrevious(tempPrev);

lPtr->setPrevious(rPtr);

lPtr->setNext(tempNext);

return;

}

Node\* tempNext = rPtr->next();

Node\* tempPrev = rPtr->previous();

tempNext->setPrevious(lPtr);

tempPrev->setNext(lPtr);

lPtr->next()->setPrevious(rPtr);

if (lPtr->previous())

lPtr->previous()->setNext(rPtr);

else

m\_begin = rPtr;

rPtr->setNext(lPtr->next());

rPtr->setPrevious(lPtr->previous());

lPtr->setNext(tempNext);

lPtr->setPrevious(tempPrev);

}

Node\* m\_begin;

Node\* m\_end;

int m\_size;

};