|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Структуризация многоэлементных**  **структур средствами struct»** | |
|  | |
| Выполнил студент группы ИКБО-09-22 | Гришин А. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получение навыков по реализации многоэлементных структур данных средствами структуры данных – таблица, в языке программирования С++.

1. **Постановка задачи**
2. Разработать набор операций для управления таблицей, созданной на основе динамического массива. Структура записи элемента таблицы определена вариантом индивидуального задания.
   1. Создать проект, добавить заголовочный файл. Включить в заголовочный файл:
      1. разработанный тип данных, определяющий структуру элемента таблицы (записи).
      2. определенная на базе типа struct структура хранения данных по таблице, содержащая: размер таблицы – n, массив из n элементов типа записи варианта.
      3. реализованная функция вывода таблицы на экран.
      4. операции варианта задания, оформленные как функции с параметрами.
   2. Для заполнения отдельной записи с клавиатуры разработать функцию, которая принимает в качестве параметра пустую запись, а возвращает заполненную.
   3. Наполнение таблицы данными должна выполнять функция вставки или добавления записи, включенная в список операций варианта. Эта функция должна приять через параметр заполненную запись, выполнить действие по вставке или добавления, согласно алгоритму варианта этой операции.
3. Разработать программу, демонстрирующую работу всех функций с массивом или записей.
4. Разработать набор тестовых данных по наполнению таблицы. Включить в меню программы возможность автоматического ввода разработанных тестовых данных.
5. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Персональный вариант – 7.

Таблица 1. Вариант задания

|  |
| --- |
| Учет кадров организации. Сотрудник: табельный номер (трехзначное число), фамилия и инициалы, код занимаемой должности, дата поступления на работу. Создать справочник по должностям – массив из 5 элементов, структура элемента: код (двухзначное число), название должности. Справочник заполните при инициализации.  Операции   1. Заполнить с клавиатуры запись по одному сотруднику. 2. Вставить в таблицу сведения по сотруднику так, чтобы запись разместилась после последней записи с такой же должностью. 3. Заменить у всех сотрудников код заданной должности на новый код (перевели на другую должность) 4. Удалить сотрудников, занимающих заданную должность. |

# **. Решение**

* 1. **Краткая теория по теме**

Таблица – это массив, элемент которого имеет сложную структуру, реализуемую в языке С++ посредством типа struct. Элемент таблицы определяет объект предметной области, который характеризуется несколькими свойствами различного типа. Например, объект Книга, в библиотечной системе может быть определен следующим набором свойств: ISBN, Автор, Название, Издательство, Год издания, Количество страниц, Цена. Элементы таблицы содержат поле – ключ, значение которого однозначно идентифицирует данные элемента. Так ISBN – ключ для данной книги. Таблицу можно реализовать контейнером vector библиотеки STL, в котором в качестве элемента выступает тип данных struct.

Структура – тип определяемый пользователем. Этот тип данных используется для представления в программе объектов, обладающих несколькими свойствами возможно различного типа. Например, объект Книга, может быть определена свойствами: фамилия автора, название книги, год издания, издательство. Структура позволяет представить объект, обладающий несколькими свойствами, одной переменной. Формат определения типа:

|  |
| --- |
| struct Employee {  unsigned int id;  char LastName[50];  char Initials[50];  unsigned int CodeofHeld;  char StartDate[50];  };  struct Position {  int CodeofHeld;  char Name[50];  }; |

Правила создания типа struct:

1. Имя поля должно быть уникально в пределах одного типа struct.
2. Тип поля – это любой тип языка, включая массивы и структуры Си.
3. Если несколько полей имеют одинаковый тип, то их имена можно перечислить через запятую.
   1. **. Дополнительные функции**

Для упрощения работы над реализацией программы и повышения читаемости кода введем дополнительные функции: PrintEmployees - для вывода таблицы сотрудников на экран, FillNewEmployee – для заполнения записи по одному сотруднику и FillDirectory – для заполнения справочника по должностям.

|  |
| --- |
| void PrintEmployees(Employee\* employeesTable, int rowsCount, int posCount, Position\* positions) {  cout << "=====================================================================================================" << endl;  cout << "| Табельный номер |" << " Фамилия |" << " Инициалы |" << " Код должн. |" << " Должность |" << " Дата поступления |" << endl;  cout << "=====================================================================================================" << endl;  int size = 0;  for (int i = 0; i < rowsCount; i++)  {  cout.width(0);  cout << "| " << employeesTable[i].ID;  cout.width(14);  cout << "|" << employeesTable[i].LastName;  cout.width(16 - strlen(employeesTable[i].LastName));  cout << "|" << employeesTable[i].Initials;  cout.width(6);  cout << "|" << employeesTable[i].CodeofHeld;  cout.width(14 - 2);  for (int j = 0; j < posCount; j++) {  if (employeesTable[i].CodeofHeld == positions[j].CodeofHeld) {  cout << "|" << positions[j].Name;  cout.width(22 - strlen(positions[j].Name));  break;  }  }  cout << "|" << employeesTable[i].StartDate;  cout.width(19 - strlen(employeesTable[i].StartDate)); cout << "|" << endl;  };  cout << "=====================================================================================================" << endl;  }  void FillDirectory(Position\* positions, int num) {  for (int i = 0; i < num; i++) {  Position pos;  cout << "Введите код должности: "; cin >> pos.CodeofHeld;  cout << "Введите название этой должности: "; gets\_s(pos.Name);  cout << endl << endl;  }  }  void FillNewEmployee(Employee& employee) {  cout << endl;  cout << "Заполнение записи для сотрудника" << endl;  cout << "Введите табельный номер (трёхзначное число): "; cin >> employee.ID;  cin.ignore();  cout << "Введите фамилию: "; gets\_s(employee.LastName);  cout << "Введите инициалы: "; gets\_s(employee.Initials);  cout << "Введите код занимаемой должности (двузначное число): "; cin >> employee.CodeofHeld;  cin.ignore();  cout << "Введите дату поступления на работу: "; gets\_s(employee.StartDate);  } |

* 1. **. Выполнение первой операции задания - заполнить с клавиатуры запись по одному сотруднику.**

При выборе выполнения первой операции для таблицы, в которой необходимо заполнить запись по одному сотруднику и вставить её в конец таблицы, вызывается функция FillNewEmployee, принимающая в качестве аргумента пустой объект типа Employee – структура сотрудника.

После заполнения записи сотрудника её необходимо вставить в таблицу, поэтому количество строк (сотрудников) в таблице увеличивается, следовательно увеличивается размер массива, посредством функции realloc, и вызывается функция AddEmployeeTableEnd, принимающая запись сотрудника для вставки, саму таблицу сотрудников и количество строк в таблице.

|  |
| --- |
| void AddEmployeeTableEnd(Employee employee, Employee\* employeesTable, int rowsCount) {  employeesTable[rowsCount - 1] = employee;  } |

В самой функции происходит присвоение последней ячейке таблицы значение заполненной записи.

* 1. **Выполнение второй операции задания - вставить в таблицу сведения по сотруднику так, чтобы запись разместилась после последней записи с такой же должностью.**

При выборе выполнения второй операции для таблицы, в которой необходимо заполнить запись по одному сотруднику и вставить её так, чтобы запись была размещена после последней записи с такой же должностью, также вызывается функция FillNewEmployee, принимающая в качестве аргумента пустой объект типа Employee – структура сотрудника.

После заполнения записи сотрудника происходит та же последовательность действий, но уже вызывается функция AddEmployeeAfterSameHeld, принимающая запись сотрудника для вставки, саму таблицу сотрудников и количество строк в таблице.

|  |
| --- |
| void AddEmployeeAfterSameHeld(Employee employee, Employee\* employeesTable, int rowsCount) {  for (int i = 0; i < rowsCount; i++) {  if (i == rowsCount - 1) {  employeesTable[rowsCount - 1] = employee;  break;  }  if (employeesTable[i].CodeofHeld == employee.CodeofHeld && employeesTable[i + 1].CodeofHeld != employee.CodeofHeld) {  for (int j = rowsCount - 1; j > i; j--) {  employeesTable[j] = employeesTable[j - 1];  }  employeesTable[i + 1] = employee;  break;  }  }  } |

В функции происходит проход по всем элементам таблицы. Если cотрудников с нужным номером должности не нашлось, то запись добавляется в конец таблицы. Если сотрудники с таким номером должности имеются в таблице, то происходит поиск позиции последнего из них и вставка записи сотрудника после данной позиции. Для корректного вывода таблицы последующие элементы сдвигаются на одну позицию вправо.

* 1. **Выполнение третьей операции задания - заменить у всех сотрудников код заданной должности на новый код (перевели на другую должность).**

Для реализации данной операции пользователю необходимо ввести коды двух должностей, по которому будет производится замена должностей.

Так, после ввода кодов, вызывается функция ChangeCodeofHeld для замены кодов сотрудников посредством изменения значения CodeofHeld. Данная функция принимает саму таблицу, количество сотрудников и два кода должностей.

|  |
| --- |
| void ChangeCodeofHeld(Employee\* employeesTable, int rowsCount, int num1, int num2) {  for (int i = 0; i < rowsCount; i++) {  if (employeesTable[i].CodeofHeld == num1) {  employeesTable[i].CodeofHeld = num2;  }  }  } |

В самой функции происходит перебор элементов таблицы и, если найден сотрудник, код должности которого совпадает с введённым, то происходит замена кода должности сотрудника на второй введённый код.

* 1. **Выполнение четвертой операции задания - удалить сотрудников, занимающих заданную должность.**

Для реализации данной операции пользователю необходимо ввести код должности, по которому будет производится удаление сотрудников.

|  |
| --- |
| void DeleteEmployeesByCode(Employee\* employeesTable, int& rowsCount, int code) {  for (int i = 0; i < rowsCount; i++) {  if (employeesTable[i].CodeofHeld == code) {  for (int j = i; j < rowsCount; j++) {  employeesTable[j] = employeesTable[j + 1];  }  rowsCount--;  i--;  }  }  } |

В самой функции происходит перебор элементов таблицы и, если найден сотрудник, код должности которого совпадает с введённым, то происходит удаление данного сотрудника путём сдвига всех следующих сотрудников на одну позицию влево. Также уменьшается значение переменной количества сотрудников в таблице и счётчика цикла, так как элемент на позиции счетчика был удален.

* 1. **Описание работы программы и её интерфейса**

В самом начале программы выводится меню выбора изначальной таблицы: пустая или с изначальным набором данных. Далее выводится основное меню операций для работы с таблицей. Для демонстрации возьмем пустую таблицу и добавим несколько сотрудников (рис. 1).

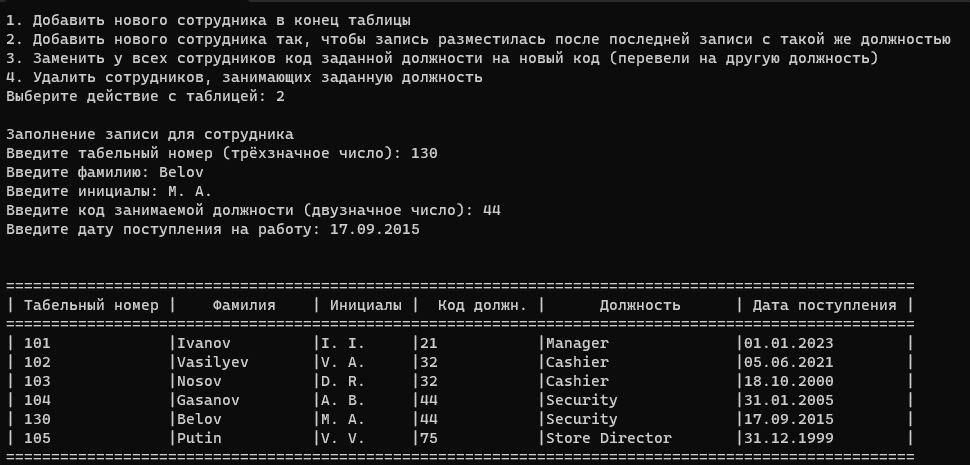


Рисунок 1. Интерфейс работы программы

Демонстрация работы остальных операций программы будет описана в разделе тестирования программы.

1. **Тестирование**

Протестируем работу программы для всех операций. Для наглядности будем использовать готовый набор данных для таблицы.

Рассмотрим выполнение первой операции (рис. 2).

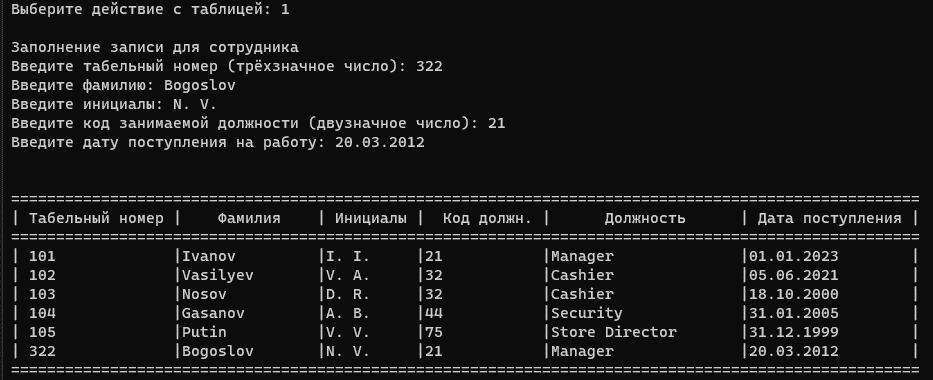


Рисунок 2. Тестирование работы первой операции

Тестирование первой операции завершено успешно. Протестируем работу второй операции при нахождении сотрудников с определённым кодом должности, как у созданного сотрудника (рис. 3) и при его отсутствии (рис. 4).

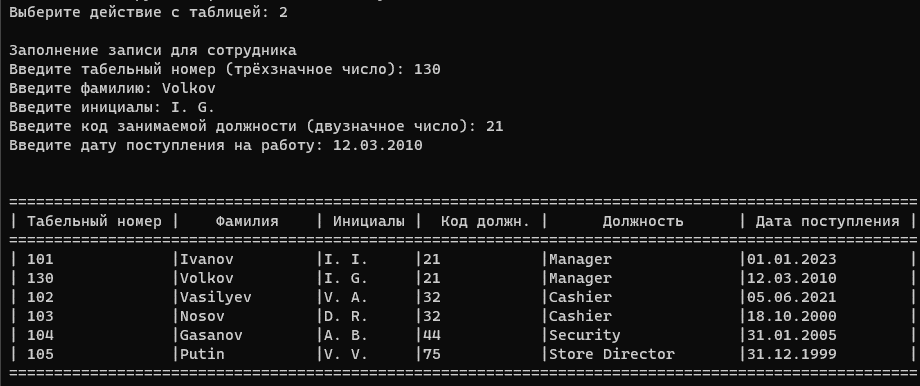


Рисунок 3. Тестирование работы второй операции

при нахождении сотрудников с определённым кодом должности

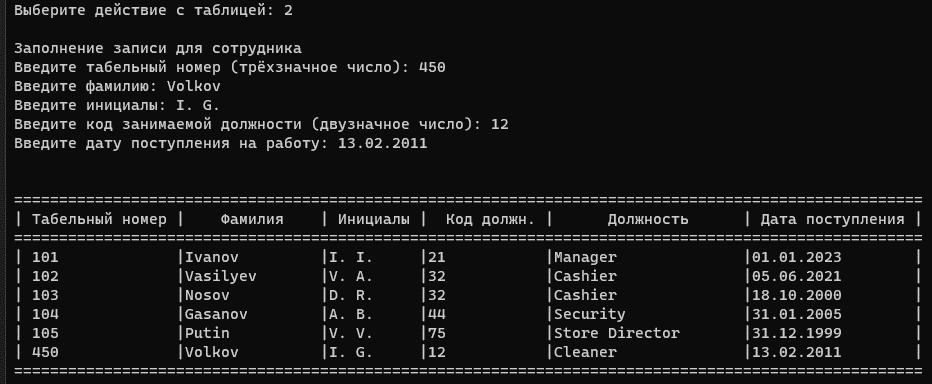


Рисунок 4. Тестирование работы второй операции

при отсутствии сотрудников с определённым кодом должности

Тестирование второй операции завершено успешно. Протестируем работу третьей операции при нахождении сотрудников с определённым кодом должности (рис. 5) и при отсутствии таких (рис. 6).

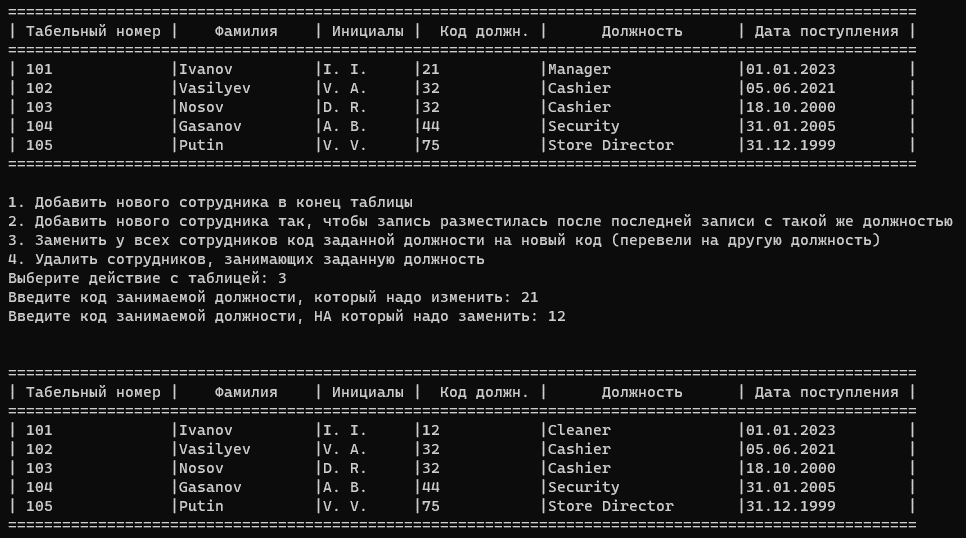


Рисунок 5. Тестирование работы третьей операции при нахождении

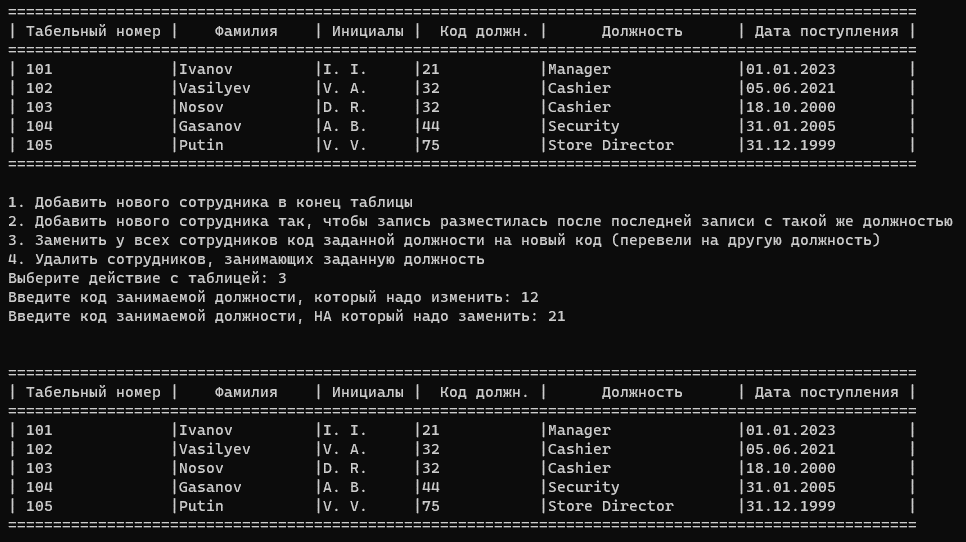


Рисунок 6. Тестирование работы третьей операции при отсутствии

При вводе должности под кодом 12 таблица не изменила своих данных, значит тестирование третьей операции завершено успешно. Протестируем работу четвертой операции при нахождении сотрудников с определённым кодом должности (рис. 7) и при их отсутствии (рис. 8).

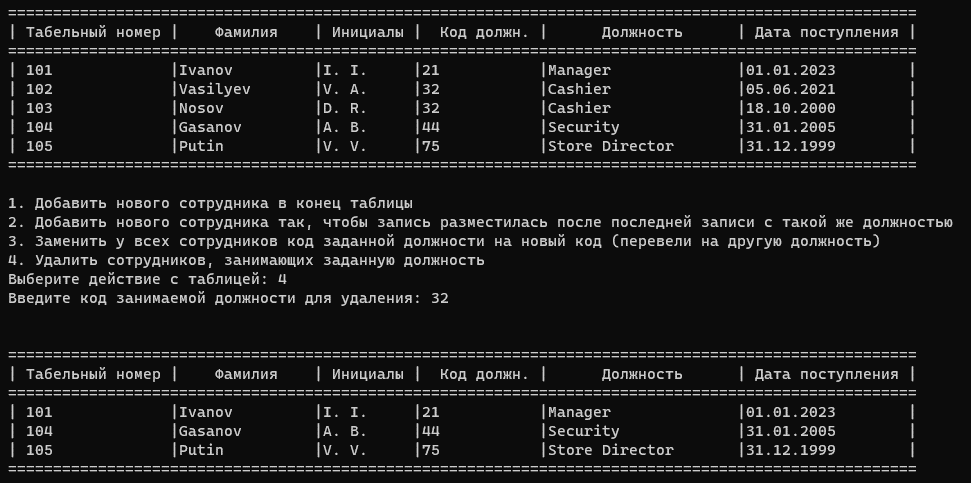


Рисунок 7. Тестирование работы четвертой операции

при нахождении сотрудников с определённым кодом должности

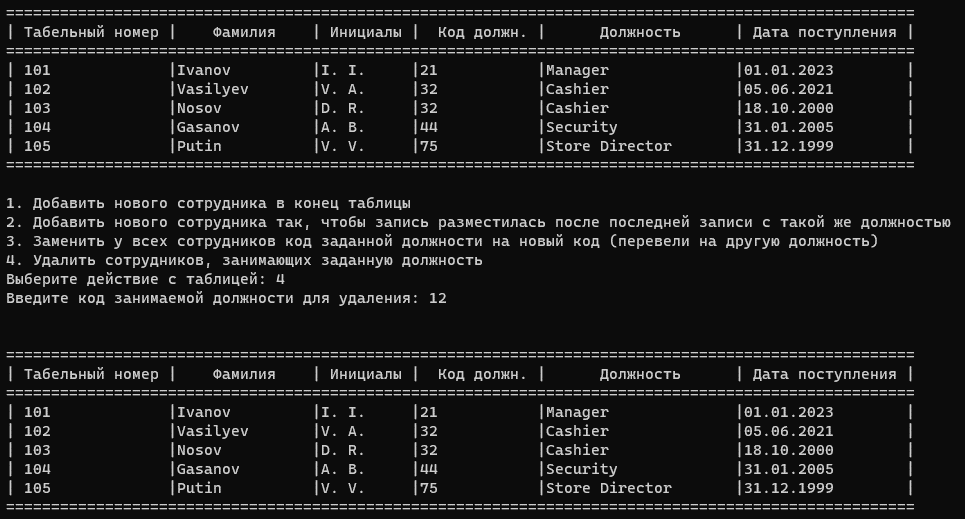


Рисунок 8. Тестирование работы четвертой операции

при отсутствии сотрудников с определённым кодом должности

Тестирование четвертой операции завершено успешно. Все тесты пройдены успешно.

1. **Вывод**

В результате выполнения работы я приобрел навыки по работе со структурами и реализации многоэлементных структур данных средствами структуры данных – таблица, в языке программирования С++.

1. **Исходный код программы**

Исходный код файла main.cpp

|  |
| --- |
| #include <iostream>  #include "Employees.h"  using namespace std;  int main()  {  setlocale(LC\_ALL, "Russian");  cout << "Лабораторная работа №4 по теме Структуры. Гришин Андрей" << endl;  int choice;  cout << "1. Работа с пустой таблицей" << endl;  cout << "2. Работа с таблицей, заполненной автоматически тестовыми данными" << endl;  cout << "Выберите пункт меню: "; cin >> choice;  Employee\* employeesTable;  Employee employee;  Position\* positions;  int rowsCount = 0;  int posCount = 0;  switch (choice) {  case 2: {  rowsCount = 5;  posCount = 5;  employeesTable = new Employee[rowsCount]  {  {101, "Ivanov", "I. I.", 21, "01.01.2023"},  {102, "Vasilyev", "V. A.", 32, "05.06.2021"},  {103, "Nosov", "D. R.", 32, "18.10.2000"},  {104, "Gasanov", "A. B.", 44, "31.01.2005"},  {105, "Putin", "V. V.", 75, "31.12.1999"}  };  positions = new Position[posCount]{  {21, "Manager"},  {32, "Cashier"},  {44, "Security"},  {75, "Store Director"},  {12, "Cleaner"}  };  break;  }  default: {  employeesTable = new Employee[rowsCount];  cout << endl;  cout << "Заполните справочник должностей: " << endl;  cout << "Введите количество должностей: "; cin >> posCount;  positions = new Position[posCount];  }  }  while (true) {  cout << endl << endl;  PrintEmployees(employeesTable, rowsCount, posCount, positions);  cout << "\n1. Добавить нового сотрудника в конец таблицы" << endl;  cout << "2. Добавить нового сотрудника так, чтобы запись разместилась после последней записи с такой же должностью" << endl;  cout << "3. Заменить у всех сотрудников код заданной должности на новый код (перевели на другую должность)" << endl;  cout << "4. Удалить сотрудников, занимающих заданную должность" << endl;  cout << "Выберите действие с таблицей: "; cin >> choice;  cin.ignore();  switch (choice)  {  case 1: {  FillNewEmployee(employee);  rowsCount++;  employeesTable = (Employee\*)realloc(employeesTable, (rowsCount) \* sizeof(Employee));  AddEmployeeTableEnd(employee, employeesTable, rowsCount);  break;  }  case 2: {  FillNewEmployee(employee);  rowsCount++;  employeesTable = (Employee\*)realloc(employeesTable, (rowsCount) \* sizeof(Employee));  AddEmployeeAfterSameHeld(employee, employeesTable, rowsCount);  break;  }  case 3: {  int num1, num2;  cout << "Введите код занимаемой должности, который надо изменить: "; cin >> num1;  cout << "Введите код занимаемой должности, НА который надо заменить: "; cin >> num2;  ChangeCodeofHeld(employeesTable, rowsCount, num1, num2);  break;  }  case 4: {  int code;  cout << "Введите код занимаемой должности для удаления: "; cin >> code;  DeleteEmployeesByCode(employeesTable, rowsCount, code);  employeesTable = (Employee\*)realloc(employeesTable, (rowsCount) \* sizeof(Employee));  break;  }  }  }  } |

Исходный код заголовочного файла Employees.h

|  |
| --- |
| #pragma once  using namespace std;  struct Employee {  int ID;  char LastName[50];  char Initials[50];  int CodeofHeld;  char StartDate[50];  };  struct Position {  int CodeofHeld;  char Name[50];  };  void PrintEmployees(Employee\* employeesTable, int rowsCount, int posCount, Position\* positions) {  cout << "=====================================================================================================" << endl;  cout << "| Табельный номер |" << " Фамилия |" << " Инициалы |" << " Код должн. |" << " Должность |" << " Дата поступления |" << endl;  cout << "=====================================================================================================" << endl;  int size = 0;  for (int i = 0; i < rowsCount; i++)  {  cout.width(0);  cout << "| " << employeesTable[i].ID;  cout.width(14);  cout << "|" << employeesTable[i].LastName;  cout.width(16 - strlen(employeesTable[i].LastName));  cout << "|" << employeesTable[i].Initials;  cout.width(6);  cout << "|" << employeesTable[i].CodeofHeld;  cout.width(14 - 2);  for (int j = 0; j < posCount; j++) {  if (employeesTable[i].CodeofHeld == positions[j].CodeofHeld) {  cout << "|" << positions[j].Name;  cout.width(22 - strlen(positions[j].Name));  break;  }  }  cout << "|" << employeesTable[i].StartDate;  cout.width(19 - strlen(employeesTable[i].StartDate)); cout << "|" << endl;  };  cout << "=====================================================================================================" << endl;  }  void FillDirectory(Position\* positions, int num) {  for (int i = 0; i < num; i++) {  Position pos;  cout << "Введите код должности: "; cin >> pos.CodeofHeld;  cout << "Введите название этой должности: "; gets\_s(pos.Name);  cout << endl << endl;  }  }  void FillNewEmployee(Employee& employee) {  cout << endl;  cout << "Заполнение записи для сотрудника" << endl;  cout << "Введите табельный номер (трёхзначное число): "; cin >> employee.ID;  cin.ignore();  cout << "Введите фамилию: "; gets\_s(employee.LastName);  cout << "Введите инициалы: "; gets\_s(employee.Initials);  cout << "Введите код занимаемой должности (двузначное число): "; cin >> employee.CodeofHeld;  cin.ignore();  cout << "Введите дату поступления на работу: "; gets\_s(employee.StartDate);  }  void AddEmployeeTableEnd(Employee employee, Employee\* employeesTable, int rowsCount) {  employeesTable[rowsCount - 1] = employee;  }  void AddEmployeeAfterSameHeld(Employee employee, Employee\* employeesTable, int rowsCount) {  for (int i = 0; i < rowsCount; i++) {  if (i == rowsCount - 1) {  employeesTable[rowsCount - 1] = employee;  break;  }  if (employeesTable[i].CodeofHeld == employee.CodeofHeld && employeesTable[i + 1].CodeofHeld != employee.CodeofHeld) {  for (int j = rowsCount - 1; j > i; j--) {  employeesTable[j] = employeesTable[j - 1];  }  employeesTable[i + 1] = employee;  break;  }  }  }  void ChangeCodeofHeld(Employee\* employeesTable, int rowsCount, int num1, int num2) {  for (int i = 0; i < rowsCount; i++) {  if (employeesTable[i].CodeofHeld == num1) {  employeesTable[i].CodeofHeld = num2;  }  }  }  void DeleteEmployeesByCode(Employee\* employeesTable, int& rowsCount, int code) {  for (int i = 0; i < rowsCount; i++) {  if (employeesTable[i].CodeofHeld == code) {  for (int j = i; j < rowsCount; j++) {  employeesTable[j] = employeesTable[j + 1];  }  rowsCount--;  i--;  }  }  } |