|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 3** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины.»** | |
|  | |
| Выполнил студент группы ИКБО-33-22 | Шило Ю.С. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2023

# **Цель работы**

Получить навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).

1. **Постановка задачи**
2. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.
3. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы.
4. Для обеспечения прямого доступа к записи в файле элемент хеш-таблицы должен включать обязательные поля: ключ записи в файле, номер записи с этим ключом в файле. Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.
5. Управление хеш-таблицей
   1. Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте. Определения разместить в соответствующем заголовочном файле. Все операции управления хеш-таблицей размещать в этом заголовочном файле.
   2. Тестирование операций выполнять в функции main приложения по мере их реализации.
   3. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется.
   4. Разработать операции по управлению хеш-таблицей.
   5. Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.
   6. Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.
   7. Подготовить тесты (последовательность значений ключей), обеспечивающие:
      1. вставку ключа без коллизии
      2. вставку ключа и разрешение коллизии
      3. вставку ключа с последующим рехешированием
      4. удаление ключа из таблицы
      5. поиск ключа в таблице. Для метода с открытым адресом подготовить тест для поиска ключа, который размещен в таблице после удаленного ключа, с одним значением хеша для этих ключей
   8. Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.
6. Управление двоичным файлом
   1. Операции управления двоичным файлом: создание двоичного файла из текстового, добавить запись в двоичный файл, удалить запись с заданным ключом из файла, прочитать запись файла по заданному номеру записи.
   2. Структура записи двоичного файла и все операции по управлению файлом должны быть размещены в соответствующем заголовочном файле.
   3. Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.
7. Управление файлом посредством хеш-таблицы
8. В заголовочный файл управления файлом посредством хеш-таблицы подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом. Реализовать поочередно все перечисленные ниже операции в этом заголовочном файле, выполняя их тестирование из функции main приложения. После разработки всех операций выполнить их комплексное тестирование.
9. Разработать и реализовать операции
   1. Прочитать запись из файла и вставить элемент в таблицу (элемент включает: ключ и номер записи с этим ключом в файле, и для метода с открытой адресацией возможны дополнительные поля).
   2. Удалить запись из таблицы при заданном значении ключа и соответственно из файла.
   3. Найти запись в файле по значению ключа (найти ключ в хеш-таблице, получить номер записи с этим ключом в файле, выполнить прямой доступ к записи по ее номеру).
   4. Подготовить тесты для тестирования приложения:
   5. Заполните файл небольшим количеством записей.
   6. Включите в файл записи как не приводящие к коллизиям, так и приводящие.
   7. Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.
   8. Заполните файл большим количеством записей (от 500).
   9. Определите время чтения записи с заданным ключом: для первой записи файла, для средней и последней. Проверьте, что время доступа для всех записей одинаково. Сравнить с временем линейного поиска тех же записей в файле.
10. Составить отчет.
11. Ответы на вопросы
    1. Расскажите о назначении хеш-фунции.
    2. Что такое коллизия?
    3. Что такое «открытый адрес» по отношению к хеш-таблице?
    4. Как в хеш-таблице с открытым адресом реализуется коллизия?
    5. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?
    6. Что определяет коэффициент нагрузки в хеш-таблице?
    7. Что такое «первичный кластер» в таблице с открытым адресом?
    8. Как реализуется двойное хеширование?

Таблица 1. Варианта задания

|  |  |  |
| --- | --- | --- |
| № | Тип хеш-таблицы (метод разрешения коллизии) | Структура записи двоичного файла |
| 29 | Открытый адрес (двойное хеширование) | Справочник банков по городам страны. Об отдельном банке хранятся данные: наименование, код банка, адрес (город), форма собственности (коммерческий или государственный). |

# **Решение**

Файловые потоки в языке C++ предоставляют возможность управлять чтением и записью данных в файлы. Они представлены классами ifstream (input file stream) и ofstream (output file stream), а также классом fstream, который объединяет возможности чтения и записи данных.

Класс ifstream используется для чтения данных из файла. Он предоставляет методы и операторы для извлечения данных из файла, такие как оператор потокового ввода (>>), который позволяет извлекать данные из файла в переменные C++.

Класс ofstream, напротив, используется для записи данных в файл. Он предоставляет методы и операторы для записи данных в файл, такие как оператор потокового вывода (<<), который позволяет записывать данные из переменных C++ в файл.

Класс fstream объединяет функциональность классов ifstream и ofstream и позволяет одновременно читать и записывать данные в файл. Это полезно, когда требуется одновременно выполнять операции чтения и записи в одном файле.

Для работы с файловыми потоками в C++ можно использовать различные методы и операторы. Например, метод getline позволяет считывать строку из файла. Методы read и write позволяют читать и записывать данные в бинарных файлах.

Операторы потокового ввода (>>) и потокового вывода (<<) могут использоваться для удобного чтения и записи данных в файлы. Они позволяют извлекать и записывать данные различных типов из/в файлы с помощью перегруженных операторов.

Файловые потоки в C++ предоставляют удобный интерфейс для работы с файловой системой и обработки данных, хранящихся в файлах. Они позволяют эффективно управлять файлами, читать и записывать данные, а также выполнять другие операции, связанные с файлами.

**Задание 1**

1. Что бы пользователь мог удобнее взаимодействовать с нашей программы был написан интерфейс. Мы запрашиваем у пользователя цифру которая соответствует определенной функции после ввода подходящего значения выполняем эту функцию иначе говорим пользователю, что были введены некорректные данные .

|  |
| --- |
| int main() {  string filename;  cout << "Enter filename: ";  cin >> filename;   int choice;  cout << "1. Create file\n"  << "2. Print the contents of the file\n"  << "3. Add a new line to the file\n"  << "4. Read the value of the number\n"  << "5. Determine the number of numbers in the file\n"  << "6. Change file name\n"  << "7. Do task\n"  << "0. Exit\n";     while (choice != 0)  {  cout << "Select action: ";  cin >> choice;  switch (choice) {  case 1:  createFile(filename);  break;  case 2:  printFile(filename);  break;  case 3: {  string line;  cout << "Enter a new line: ";  cin.ignore();  getline(cin, line);  addLine(filename, line);  break;  }  case 4: {  int index;  cout << "Enter the number of the number: ";  cin >> index;  cout << "The number is: " << readIndex(filename, index) << "\n";  break;  }  case 5:  cout << "The number of numbers: " << numberCount(filename) << "\n";  break;  case 6: {  cout << "Enter new filename: ";  cin >> filename;  break;  }  case 7:{  string task\_filename;  cout << "Enter name to task file: ";  cin >> task\_filename;  ofstream task\_file (task\_filename);  task(filename, task\_filename);  break;  }   case 0:  cout << "Exit\n";  break;  default:  cout << "Wrong choice!\n";  break;  }   }  task(filename, "task\_filename");  printFile("task\_filename");   return 0; } |

1. Для вывода текста в консоль мы вызываем функцию printFile, в качестве входных параметров она принимает название файла содержимое, которого мы должны вывести в консоль.

|  |
| --- |
| void printFile(string filename){  ifstream file(filename);  string line;  if (file.is\_open())  while (getline(file, line)) cout << line << std::endl;  else  cout << "File opening error\n";  file.close();  } |

1. Для добавления строки в конец файл была написана функция addLine в качестве входных параметров она принимает название файла и строку которую мы должны добавить.

|  |
| --- |
| void addLine(string filename, string line){  ofstream file (filename, std::ios\_base::app);  if (file.is\_open())  file << line << std::endl;  else  cout << "File opening error\n";  file.close();  } |

1. Для того чтобы пользователю узнать на какой позиции находиться введённое число он должен вызвать функцию readIndex, в качестве входных параметров она принимает имя файла и искомое значение.

|  |
| --- |
| int readIndex(string filename, int index){  ifstream file(filename);  int count = 0;  string line;   if (file.is\_open())  while (getline(file, line)){  string temp;  count++;  for (char unit : line) {  if (isdigit(unit))  temp += unit;  else  {  if (count == index) return stoi(temp);  count++;  temp="";  }   }  }  else  cout << "File opening error\n";   file.close(); } |

1. Что бы узнать количество чисел в нашем файле нам следует воспользоваться функцией numberCount в качестве входных параметров она принимает название файла, в котором мы должны осуществить подсчет. В качестве выходного значения функция возвращает количество чисел в файле.

|  |
| --- |
| int numberCount(string filename){  ifstream file(filename);  int count = 0;  string line;  if (file.is\_open())  while (getline(file, line)){  count++;  for (char unit : line) {  if (!isdigit(unit)) count++;  }  }  else  cout << "File opening error\n";  return count;  } |

1. Для того что бы выполнить задание моего варианты мы должны вызвать функцию task, которая создает новый файл на основе файла, имя которого получает в качестве параметра. В новом файле соблюдаются следующие условия, поставленные вариантом: количество чисел на каждой строке должно быть меньше или равно числу Фибоначчи.

|  |
| --- |
| std::vector <int> fibonacciSeries(int count) {  std::vector <int> temp = {0, 1};  int i\_first = 0, i\_second = 1;  while (count >= i\_second){  temp.push\_back(i\_first + i\_second);  i\_first = i\_second;  i\_second = temp.back();  }  return temp;  }  void task(string filename, string task\_filename){  ifstream file(filename);  string line, temp;  if (file.is\_open())  while (getline(file, temp))  line += temp + " ";  else  cout << "File opening error\n";  string tmp;  int count = 0;  std::vector <int> digit;  for (char unit : line){  if (isdigit(unit))  tmp += unit;  else{  digit.push\_back(stoi(tmp));  count++;  tmp="";  }  }  ofstream task\_file (task\_filename, std::ios\_base::in);  if (task\_file.is\_open()) {  int index = 0;  for (int unit: fibonacciSeries(count)) {  for (int i = 0; i < unit && index < digit.size(); i++) {  task\_file << digit.at(index) << " ";  index++;  }  if (index < digit.size()) task\_file << "\n";  }  }  else  cout << "File opening error\n";  file.close();  task\_file.close();  } |

**Задание 2**

1. Для выполнения этого задания для начала нам необходимо создать структуру наших данных в моей случае это информация о банке.

|  |
| --- |
| struct Bank{  char Name[256];  int Code;  char Address[256];  char Ownership[8];  }; |

1. Что бы пользователь мог удобнее взаимодействовать с нашей программы был написан интерфейс. Мы запрашиваем у пользователя цифру которая соответствует определенной функции после ввода подходящего значения выполняем эту функцию иначе говорим пользователю, что были введены некорректные данные.

|  |
| --- |
| int main() {  string TextFilename, BinaryFilename;   cout << "Enter the name of file:\n";  cin >> TextFilename;  cout << "Enter the file name for the binary file:\n";  cin >> BinaryFilename;   ofstream TextFile (TextFilename);  TextFile << "Name--1 0 Address--0 com" << endl  << "Name--2 1 Address--1 gov" << endl  << "Name--3 2 Address--2 gov" << endl  << "Name--4 3 Address--3 gov" << endl  << "Name--5 4 Address--4 com" << endl  << "Name--6 5 Address--5 gov" << endl  << "Name--7 6 Address--6 com" << endl  << "Name--8 7 Address--7 gov" << endl  << "Name--9 8 Address--8 com" << endl;  TextFile.close();  ifstream BinaryFile (BinaryFilename);  BinaryFile.close();   int choice = -1, index;  cout << "=======================================[ SELECTIONS ]=======================================\n"  << "1. Converting test data from a text file to a binary file;\n"  << "2. Converting data from a binary file to a text file;\n"  << "3. Output all records of the binary file;\n"  << "4. Access to a record by its sequence number in the file;\n"  << "5. Deleting a record with a specified key value;\n"  << "6. Generate information on the banks of the city, with commercial form of ownership;\n"  << "7. Replace the ownership form of the bank with the given code with the opposite one;\n"  << "0. Exiting the programme;\n"  << "=============================================================================================\n";  while (choice != 0){  cout << "Your choice:\n";  cin >> choice;  switch (choice) {  case 0:  return 0;  case 1:  TextToBinary(TextFilename, BinaryFilename);  break;  case 2:  BinaryToText(TextFilename, BinaryFilename);  break;  case 3:  PrintBinary(BinaryFilename);  break;  case 4:  cout << "Enter the index:\n";  cin >> index;  BankAtIndex(BinaryFilename, index);  break;  case 5:  cout << "Enter the index:\n";  cin >> index;  OverwritingBank(BinaryFilename, index - 1);  break;  case 6:  AllCommercial(BinaryFilename);  break;  case 7:  cout << "Enter the index:\n";  cin >> index;  ChangeOwnership(BinaryFilename, index);  break;  default:  cout << "Incorrect value\n";  break;  }  } } |

1. Для преобразования обычного текстового файла в бинарный была написана функция TextToBinary.

|  |
| --- |
| void TextToBinary (string TextFilename, string BinaryFilename){  ifstream InputFile (TextFilename);  ofstream OutFile (BinaryFilename, ios::binary);   if (!InputFile || !OutFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;  while (InputFile >> bank.Name >> bank.Code >> bank.Address >> bank.Ownership)  OutFile.write(reinterpret\_cast<const char\*>(&bank), sizeof(Bank));   cout << "Text data converted to binary successfully!" << endl;  InputFile.close();  OutFile.close(); } |

1. Для преобразования бинарного файла в текстовый была реализована функция BinaryToText.

|  |
| --- |
| void BinaryToText (string TextFilename, string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);  ofstream OutFile (TextFilename);   if (!InputFile || !OutFile) {  cout << "Failed to open files\n";  return;  }  Bank bank;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  OutFile << bank.Name << " \\ " << bank.Code << " \\ " << bank.Address << " \\ " << bank.Ownership << endl;   cout << "Binary data converted to text successfully!" << endl;  InputFile.close();  OutFile.close(); } |

1. Для вывода в консоль содержания бинарного файла в виде читаемого текста была реализована функция PrintBinary.

|  |
| --- |
| void PrintBinary (string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;   cout << "========================[ Print binary ]========================" << endl;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  cout << "Name:: " << bank.Name << " Code:: " << bank.Code << " Address:: " << bank.Address << " Ownership:: " << bank.Ownership << endl;  cout << "================================================================" << endl;   InputFile.close(); } |

1. Чтобы узнать какой элемент стоит на введенной пользователем была реализована функция BankAtIndex. Эта функция выводит в консоль эллемент нашей структуры в виде понятного человеком текста.

|  |
| --- |
| void BankAtIndex (string BinaryFilename, int Index){  ifstream InputFile(BinaryFilename, ios::binary);   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;  InputFile.seekg(Index \* sizeof(Bank));   cout << "=======================[ Bank at index ]=======================" << endl;  if (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  else  cout << "The bank with the" << Index << "does not exist" << endl;  cout << "===============================================================" << endl;   InputFile.close(); } |

1. Чтобы затереть элемент номер которого был введен пользователем последним была реализована функция OverwritingBank.

|  |
| --- |
| void OverwritingBank(string BinaryFilename, int Code){  Bank bank, last;   fstream BinaryFile(BinaryFilename, ios::in | ios::binary);   while (BinaryFile.read(reinterpret\_cast<char\*>(&last), sizeof(last)));  BinaryFile.close();   BinaryFile.open(BinaryFilename, ios::in | ios::binary);  ofstream temp("temp.txt", ios::out | ios::binary);   while (BinaryFile.read(reinterpret\_cast<char\*>(&bank), sizeof(bank))) {  if (bank.Code == Code)  temp.write(reinterpret\_cast<char\*>(&last), sizeof(last));  else if (last.Code != bank.Code)  temp.write(reinterpret\_cast<char\*>(&bank), sizeof(bank));  }   BinaryFile.close();  remove(BinaryFilename.c\_str());  temp.close();  rename("temp.txt", BinaryFilename.c\_str()); } |

1. Для выполнения первого задания варианта была реализована функция AllCommercial, которая выводит в консоль все коммерческие банки.

|  |
| --- |
| void AllCommercial(string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);  Bank bank;   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   cout << "======================[ Commercial banks ]======================" << endl;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  if (bank.Ownership[0] == 'c' && bank.Ownership[1] == 'o' && bank.Ownership[2] == 'm')  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  cout << "================================================================" << endl;   InputFile.close(); } |

1. Для выполнения второго задания варианта была реализована функция ChangeOwnership, которая меняет принадлежность банку на противоположную текущей.

|  |
| --- |
| void ChangeOwnership(string BinaryFilename, int index){  fstream InputFile(BinaryFilename, ios::binary | ios::in | ios::out);  Bank bank;   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  if (bank.Code == index){  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  if (bank.Ownership[0] == 'g')  { bank.Ownership[0] = 'c'; bank.Ownership[1] = 'o'; bank.Ownership[2] = 'm'; }  else  { bank.Ownership[0] = 'g'; bank.Ownership[1] = 'o'; bank.Ownership[2] = 'v'; }  InputFile.write(reinterpret\_cast<char\*>(&bank), sizeof(Bank));  break;  }   InputFile.close(); } |

Интерфейс программы, которая выполняет задание 1 представлена на рисунке 1.

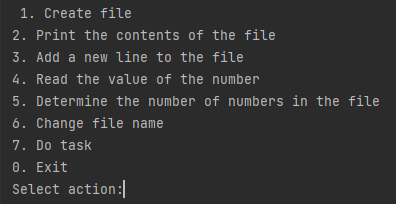


Рисунок 1. Интерфейс программы, выполняющей задание 1

Интерфейс программы, которая выполняет задание 2 представлена на рисунке 2.

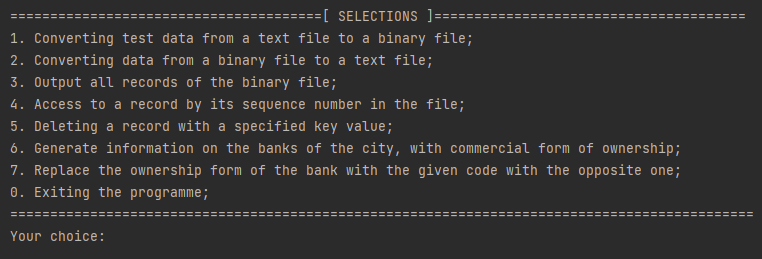


Рисунок 2. Интерфейс программы, выполняющей задание 2

# **Тестирование**

Протестируем программу, выполняющие задание номер 1. Выходные данные: создадим файл на основе образцового. Изначально в файле находятся только числа: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12. Вызовем функцию под номером 3 что бы добавить новое число 13. Затем найдем под каким индексом находится число 8. Введем число 7 что бы протестировать работу функции для индивидуального задания из варианта. Отработка введённых значений предоставлена на рисунке 3.

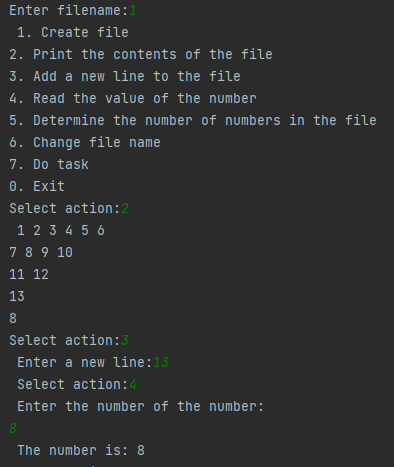


Рисунок 3. Результат тестирования и вывода программы

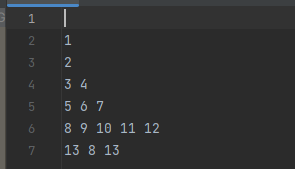


Рисунок 4. Результат тестирования и созданного файла программой, выполняющей задание 1

Протестируем программу, выполняющую задание номер 2. Содержание файла предоставлено на рисунке 5.

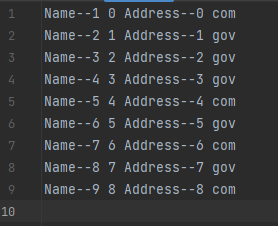


Рисунок 5 – Содержание текстового файла для проверки второго задания

Для проверки кода введем в консоль 1, чтобы сконвертировать обычный текстовый файл в бинарный. Введем в консоль 6 эта функция должна была вывести все банки с коммерческим типом владения. Затем введем 1 что бы затереть первый элемент последним и выведем то что получилось в консоль используя функцию под номером 3.

Вывод программы в консоли подтверждает корректность работы программы (см. рисунок 5).

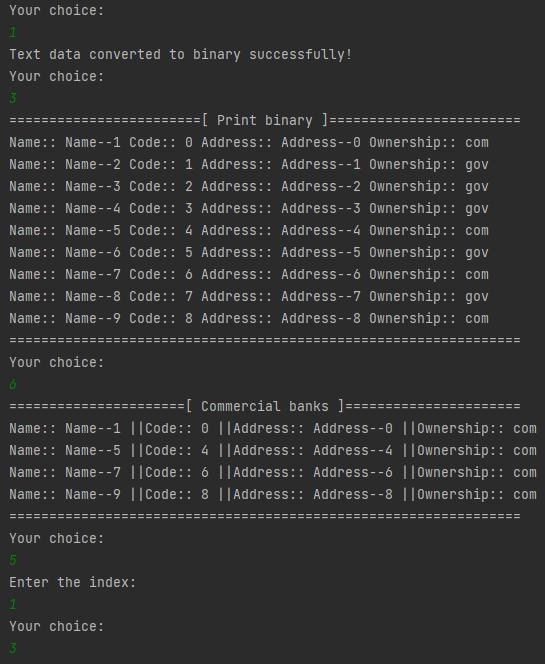
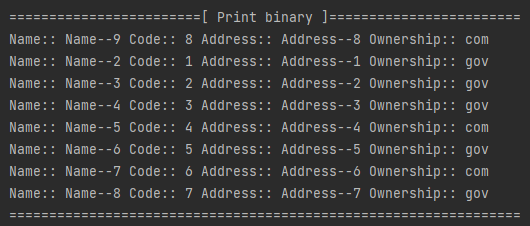
 

Рисунок 5. Результат тестирования и вывода программы, выполняющей задание 2

# **Вывод**

В процессе выполнения данной практической работы я успешно овладел навыками работы с файловыми потоками в языке программирования C++ для управления текстовыми и двоичными файлами. Эти навыки позволят мне эффективно читать, записывать и обрабатывать данные из файлов, что является важной частью разработки программ, работающих с файловой системой.

# **Исходный код программы**

Исходный код файла для выполнения задания 1 -main.cpp

|  |
| --- |
| #include <iostream>  #include <string>  #include <fstream>  #include <vector>  using std::cin, std::cout, std::string, std::ofstream, std::ifstream, std::getline;  // Функция для создания текстового файла, содержащего десятичные числа void createFile(string filename){  ofstream file(filename);   if (file.is\_open())  file << "1 2 3 4 5 6\n"  << "7 8 9 10\n"  << "11 12\n";  else  cout << "File opening error\n";   file.close(); }  // Функция для вывода содержимого текстового файла void printFile(string filename){  ifstream file(filename);  string line;   if (file.is\_open())  while (getline(file, line)) cout << line << std::endl;   else  cout << "File opening error\n";   file.close(); }  // Добавление новой строки в конец файла void addLine(string filename, string line){  ofstream file (filename, std::ios\_base::app);   if (file.is\_open())  file << line << std::endl;  else  cout << "File opening error\n";   file.close(); }  // Вывести эллемент под индексов введенным пользователем int readIndex(string filename, int index){  ifstream file(filename);  int count = 0;  string line;   if (file.is\_open())  while (getline(file, line)){  string temp;  count++;  for (char unit : line) {  if (isdigit(unit))  temp += unit;  else  {  if (count == index) return stoi(temp);  count++;  temp="";  }   }  }  else  cout << "File opening error\n";   file.close(); }  // Определить кол-во чисел в файле int numberCount(string filename){  ifstream file(filename);  int count = 0;  string line;   if (file.is\_open())  while (getline(file, line)){  count++;  for (char unit : line) {  if (!isdigit(unit)) count++;  }  }  else  cout << "File opening error\n";   return count; }  // Задние по вариантам Вариант №29 std::vector <int> fibonacciSeries(int count) {  std::vector <int> temp = {0, 1};  int i\_first = 0, i\_second = 1;   while (count >= i\_second){  temp.push\_back(i\_first + i\_second);  i\_first = i\_second;  i\_second = temp.back();  }   return temp; }  void task(string filename, string task\_filename){  ifstream file(filename);  string line, temp;   if (file.is\_open())  while (getline(file, temp))  line += temp + " ";  else  cout << "File opening error\n";   string tmp;  int count = 0;  std::vector <int> digit;  for (char unit : line){  if (isdigit(unit))  tmp += unit;  else{  digit.push\_back(stoi(tmp));  count++;  tmp="";  }  }   ofstream task\_file (task\_filename, std::ios\_base::in);  if (task\_file.is\_open()) {  int index = 0;  for (int unit: fibonacciSeries(count)) {  for (int i = 0; i < unit && index < digit.size(); i++) {  task\_file << digit.at(index) << " ";  index++;  }  if (index < digit.size()) task\_file << "\n";  }  }  else  cout << "File opening error\n";   file.close();  task\_file.close(); }  int main() {  string filename;  cout << "Enter filename: ";  cin >> filename;   int choice;  cout << "1. Create file\n"  << "2. Print the contents of the file\n"  << "3. Add a new line to the file\n"  << "4. Read the value of the number\n"  << "5. Determine the number of numbers in the file\n"  << "6. Change file name\n"  << "7. Do task\n"  << "0. Exit\n";     while (choice != 0)  {  cout << "Select action: ";  cin >> choice;  switch (choice) {  case 1:  createFile(filename);  break;  case 2:  printFile(filename);  break;  case 3: {  string line;  cout << "Enter a new line: ";  cin.ignore();  getline(cin, line);  addLine(filename, line);  break;  }  case 4: {  int index;  cout << "Enter the number of the number: ";  cin >> index;  cout << "The number is: " << readIndex(filename, index) << "\n";  break;  }  case 5:  cout << "The number of numbers: " << numberCount(filename) << "\n";  break;  case 6: {  cout << "Enter new filename: ";  cin >> filename;  break;  }  case 7:{  string task\_filename;  cout << "Enter name to task file: ";  cin >> task\_filename;  ofstream task\_file (task\_filename);  task(filename, task\_filename);  break;  }   case 0:  cout << "Exit\n";  break;  default:  cout << "Wrong choice!\n";  break;  }   } // Добавить отработку переполнения и то функция отработана   task(filename, "task\_filename");  printFile("task\_filename");   return 0; } |

Исходный код файла для выполнения задания 2 - 222.cpp

|  |
| --- |
| #include <iostream> #include <string> #include <fstream>  using namespace std;  struct Bank{  char Name[256];  int Code;  char Address[256];  char Ownership[8]; };  // Done void TextToBinary (string TextFilename, string BinaryFilename){  ifstream InputFile (TextFilename);  ofstream OutFile (BinaryFilename, ios::binary);   if (!InputFile || !OutFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;  while (InputFile >> bank.Name >> bank.Code >> bank.Address >> bank.Ownership)  OutFile.write(reinterpret\_cast<const char\*>(&bank), sizeof(Bank));   cout << "Text data converted to binary successfully!" << endl;  InputFile.close();  OutFile.close(); } // Done void BinaryToText (string TextFilename, string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);  ofstream OutFile (TextFilename);   if (!InputFile || !OutFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  OutFile << bank.Name << " \\ " << bank.Code << " \\ " << bank.Address << " \\ " << bank.Ownership << endl;   cout << "Binary data converted to text successfully!" << endl;  InputFile.close();  OutFile.close(); } // Done void PrintBinary (string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;   cout << "========================[ Print binary ]========================" << endl;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  cout << "Name:: " << bank.Name << " Code:: " << bank.Code << " Address:: " << bank.Address << " Ownership:: " << bank.Ownership << endl;  cout << "================================================================" << endl;   InputFile.close(); } // Done void BankAtIndex (string BinaryFilename, int Index){  ifstream InputFile(BinaryFilename, ios::binary);   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   Bank bank;  InputFile.seekg(Index \* sizeof(Bank));   cout << "=======================[ Bank at index ]=======================" << endl;  if (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  else  cout << "The bank with the" << Index << "does not exist" << endl;  cout << "===============================================================" << endl;   InputFile.close(); } // Done void OverwritingBank(string BinaryFilename, int Code){  Bank bank, last;   fstream BinaryFile(BinaryFilename, ios::in | ios::binary);   while (BinaryFile.read(reinterpret\_cast<char\*>(&last), sizeof(last)));  BinaryFile.close();   BinaryFile.open(BinaryFilename, ios::in | ios::binary);  ofstream temp("temp.txt", ios::out | ios::binary);   while (BinaryFile.read(reinterpret\_cast<char\*>(&bank), sizeof(bank))) {  if (bank.Code == Code)  temp.write(reinterpret\_cast<char\*>(&last), sizeof(last));  else if (last.Code != bank.Code)  temp.write(reinterpret\_cast<char\*>(&bank), sizeof(bank));  }   BinaryFile.close();  remove(BinaryFilename.c\_str());  temp.close();  rename("temp.txt", BinaryFilename.c\_str()); } // Done void AllCommercial(string BinaryFilename){  ifstream InputFile(BinaryFilename, ios::binary);  Bank bank;   if (!InputFile) {  cout << "Failed to open files\n";  return;  }    cout << "======================[ Commercial banks ]======================" << endl;  while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  if (bank.Ownership[0] == 'c' && bank.Ownership[1] == 'o' && bank.Ownership[2] == 'm')  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  cout << "================================================================" << endl;   InputFile.close(); } // Done void ChangeOwnership(string BinaryFilename, int index){  fstream InputFile(BinaryFilename, ios::binary | ios::in | ios::out);  Bank bank;   if (!InputFile) {  cout << "Failed to open files\n";  return;  }   while (InputFile.read(reinterpret\_cast<char\*>(&bank), sizeof(Bank)))  if (bank.Code == index){  cout << "Name:: " << bank.Name << " ||Code:: " << bank.Code << " ||Address:: " << bank.Address << " ||Ownership:: " << bank.Ownership << endl;  if (bank.Ownership[0] == 'g')  { bank.Ownership[0] = 'c'; bank.Ownership[1] = 'o'; bank.Ownership[2] = 'm'; }  else  { bank.Ownership[0] = 'g'; bank.Ownership[1] = 'o'; bank.Ownership[2] = 'v'; }  InputFile.write(reinterpret\_cast<char\*>(&bank), sizeof(Bank));  break;  }   InputFile.close(); }  int main() {  string TextFilename, BinaryFilename;   cout << "Enter the name of file:\n";  cin >> TextFilename;  cout << "Enter the file name for the binary file:\n";  cin >> BinaryFilename;   ofstream TextFile (TextFilename);  TextFile << "Name--1 0 Address--0 com" << endl  << "Name--2 1 Address--1 gov" << endl  << "Name--3 2 Address--2 gov" << endl  << "Name--4 3 Address--3 gov" << endl  << "Name--5 4 Address--4 com" << endl  << "Name--6 5 Address--5 gov" << endl  << "Name--7 6 Address--6 com" << endl  << "Name--8 7 Address--7 gov" << endl  << "Name--9 8 Address--8 com" << endl;  TextFile.close();  ifstream BinaryFile (BinaryFilename);  BinaryFile.close();   int choice = -1, index;  cout << "=======================================[ SELECTIONS ]=======================================\n"  << "1. Converting test data from a text file to a binary file;\n"  << "2. Converting data from a binary file to a text file;\n"  << "3. Output all records of the binary file;\n"  << "4. Access to a record by its sequence number in the file;\n"  << "5. Deleting a record with a specified key value;\n"  << "6. Generate information on the banks of the city, with commercial form of ownership;\n"  << "7. Replace the ownership form of the bank with the given code with the opposite one;\n"  << "0. Exiting the programme;\n"  << "=============================================================================================\n";  while (choice != 0){  cout << "Your choice:\n";  cin >> choice;  switch (choice) {  case 0:  return 0;  case 1:  TextToBinary(TextFilename, BinaryFilename);  break;  case 2:  BinaryToText(TextFilename, BinaryFilename);  break;  case 3:  PrintBinary(BinaryFilename);  break;  case 4:  cout << "Enter the index:\n";  cin >> index;  BankAtIndex(BinaryFilename, index);  break;  case 5:  cout << "Enter the index:\n";  cin >> index;  OverwritingBank(BinaryFilename, index - 1);  break;  case 6:  AllCommercial(BinaryFilename);  break;  case 7:  cout << "Enter the index:\n";  cin >> index;  ChangeOwnership(BinaryFilename, index);  break;  default:  cout << "Incorrect value\n";  break;  }  } } |