|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле.»** | |
|  | |
| 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 в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);
* получить навыки в применении файловых потоков прямого доступа к данным файла;
* получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

1. **Постановка задачи**
2. Разработать класс управления файлом. Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск и вывод записи в файле с помощью линейного поиска; вывод записи по известному адресу в файле, добавление записи в файл. Структура записи файла определена индивидуальным вариантом задания.
3. Разработать класс «Бинарное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.
4. Разработать класс «Сбалансированное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.
5. Разработать приложение, демонстрирующее выполнение всех методов и поиск записей в файле с помощью БДП, СДП и линейного поиска.
6. Провести анализ времени выполнения поиска записей, находящихся в начале, середине и конце большого файла тремя способами.
7. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в СДП при его формировании из двоичного файла.
8. Выполнить тестирование.
9. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

|  |  |  |
| --- | --- | --- |
| № | Тип балансированного  дерева поиска (СДП) | Структура записи (ключ – подчеркнутое поле) |
| 4 | Рандомизированное | Владельцев автомобилей. номер машины, марка, сведения о владельце. |

# **Решение**

**Задание**

1. Структура CarRecord которая будет находится в файле. Данная структура содержит три поля одно поле целочисленное, а два остальных символьный массив.

|  |
| --- |
| struct CarRecord{  int key;  char brand[64];  char owner[64]; }; |

1. Для создания бинарного файла был написан метод createFile. Данный метод преобразует обычный текстовый файл в бинарный.

|  |
| --- |
| void createFile(const string& textFileName, const string& binaryFileName) {  ifstream textFile(textFileName);  file.open(binaryFileName, ios::out | ios::binary);  if (!textFile || !file) {  cout << "Failed to open files." << endl;  return;  }  CarRecord record;  while (textFile >> record.key >> record.brand >> record.owner) {  file.write(reinterpret\_cast<const char\*>(&record), sizeof(CarRecord));  }  textFile.close();  file.close(); } |

1. Для линейного поиска объекта в бинарном файле был написан метод linearSearch. Данный метод ищет объект с нужным ключом и выводит информацию о данном ключе.

|  |
| --- |
| void linearSearch(const string& binaryFileName, int carNumber) {  file.open(binaryFileName, ios::in | ios::binary);  if (!file) {  std::cout << "Failed to open file." << std::endl;  return;  }  CarRecord record;  bool found = false;  while (file.read(reinterpret\_cast<char\*>(&record), sizeof(CarRecord))) {  if (record.key == carNumber) {  cout << "Record found." << endl;  cout << "KEY | Brand\t| Owner Info" << endl;  cout << record.key << " | " << record.brand << "\t| " << record.owner << endl;  found = true;  break;  }  }  if (!found) {  cout << "Record not found." << endl;  }  file.close(); } |

1. Для вывода записей из файла был написан метод printRecords. Данный метод выводит все объекты, которые содержаться в бинарном файле.

|  |
| --- |
| void printRecords(const string& binaryFileName){  file.open(binaryFileName, ios::in | ios::binary);  CarRecord record;  cout << "KEY | Brand\t| Owner Info" << endl;  while (file.read(reinterpret\_cast<char\*>(&record), sizeof(CarRecord))) {  cout << record.key << " | " << record.brand << "\t| " << record.owner << endl;  }  file.close(); } |

1. Что бы добавить объект в конец бинарного файла был написан метод addRecord.

|  |
| --- |
| void addRecord(const string& binaryFileName, CarRecord newRecord) {  file.open(binaryFileName, ios::out | ios::app | ios::binary);  if (!file) {  cout << "Failed to open file." << endl;  return;  }  file.write(reinterpret\_cast<const char\*>(&newRecord), sizeof(CarRecord));   file.close(); } |

1. Для поиска элемента внутри бинарного дерева поиска был написан метод searchRecursive.

|  |
| --- |
| NodeBin\* searchRecursive(NodeBin\* node, int key) {  if (node == nullptr || node->key == key){  return node;  }   if (key < node->key) {  return searchRecursive(node->left, key);  }  else {  return searchRecursive(node->right, key);  } } |

1. Для удаления элемента внутри бинарного дерева поиска был написан метод deleteRecursive.

|  |
| --- |
| NodeBin\* deleteRecursive(NodeBin\* node, int key) {  if (node == nullptr) {  return node;  }   if (key < node->key) {  node->left = deleteRecursive(node->left, key);  } else if (key > node->key) {  node->right = deleteRecursive(node->right, key);  } else {  if (node->left == nullptr) {  NodeBin\* temp = node->right;  delete node;  return temp;  } else if (node->right == nullptr) {  NodeBin\* temp = node->left;  delete node;  return temp;  }   NodeBin\* minValueNode = getMinValueNode(node->right);  node->key = minValueNode->key;  node->ptr = minValueNode->ptr;  node->right = deleteRecursive(node->right, minValueNode->key);  }   return node; } |

1. Для вывода бинарного дерева поиска был написан метод printRecursive.

|  |
| --- |
| void printTreeRecursive(NodeBin\* node, int level) {  if (node != nullptr) {  printTreeRecursive(node->right, level + 1);  for (int i = 0; i < level; i++) {  std::cout << " ";  }  cout << level;  std::cout << "-> " << node->key << "\\" << node->ptr->brand << std::endl;  printTreeRecursive(node->left, level + 1);  }  } |

1. Для поворота рандомизированного древа было написано два метода rotateLeft и rotateRight.

|  |
| --- |
| Node\* rotateRight(Node\* node) {  Node\* newRoot = node->left;  node->left = newRoot->right;  newRoot->right = node;  return newRoot;  }  Node\* rotateLeft(Node\* node) {  Node\* newRoot = node->right;  node->right = newRoot->left;  newRoot->left = node;  return newRoot;  } |

1. Для добавления элемента в рандомизированном древе поиска был написан метод insertNode.

|  |
| --- |
| Node\* insertNode(Node\* node, int key, CarRecord\* ptr) {  if (node == nullptr)  return new Node(key, ptr);   if (key < node->key) {  node->left = insertNode(node->left, key, ptr);  if (node->left->priority > node->priority)  node = rotateRight(node);  } else {  node->right = insertNode(node->right, key, ptr);  if (node->right->priority > node->priority)  node = rotateLeft(node);  }   return node; } |

1. Для удаления элемента в рандомизированном древе поиска был написан метод deleteNode.

|  |
| --- |
| Node\* deleteNode(Node\* node, int key) {  if (node == nullptr)  return nullptr;   if (key < node->key)  node->left = deleteNode(node->left, key);  else if (key > node->key)  node->right = deleteNode(node->right, key);  else {  if (node->left == nullptr) {  Node\* temp = node->right;  delete node;  node = temp;  } else if (node->right == nullptr) {  Node\* temp = node->left;  delete node;  node = temp;  } else {  if (node->left->priority > node->right->priority) {  node = rotateRight(node);  node->right = deleteNode(node->right, key);  } else {  node = rotateLeft(node);  node->left = deleteNode(node->left, key);  }  }  }   return node; } |

1. Для поиска элемента внутри рандомизированного древа написан метод searchRecursive.

|  |
| --- |
| Node\* searchRecursive(Node\* node, int key) {  if (node == nullptr || node->key == key){  return node;  }   if (key < node->key) {  return searchRecursive(node->left, key);  }  else {  return searchRecursive(node->right, key);  } } |

1. Для вывода рандомизированного древа был написан метод printTreeRecursive.

|  |
| --- |
| void printTreeRecursive(Node\* node, int level) {  if (node != nullptr) {  printTreeRecursive(node->right, level + 1);  for (int i = 0; i < level; i++) {  std::cout << " ";  }  cout << level;  std::cout << "-> " << node->key << "\\" << node->ptr->key << "\\" << node->ptr->brand << std::endl;  printTreeRecursive(node->left, level + 1);  } |

1. Для вывода рандомизированного древа был написан метод printTreeRecursive.

|  |
| --- |
| void printTreeRecursive(Node\* node, int level) {  if (node != nullptr) {  printTreeRecursive(node->right, level + 1);  for (int i = 0; i < level; i++) {  std::cout << " ";  }  cout << level;  std::cout << "-> " << node->key << "\\" << node->ptr->key << "\\" << node->ptr->brand << std::endl;  printTreeRecursive(node->left, level + 1);  } |

1. Для проверки бинарного древа поиска была написана функция testFM.

|  |
| --- |
| void testBST(){  string binaryFileName = "binary-test.txt";   BinarySearchTree bst;  bst.Load(binaryFileName);   cout << "===[ BinarySearchTree ]===" << endl;  bst.printTree();  cout << endl;   cout << "Delete elem with key 538398" << endl;  bst.remove(538398);   cout << "===[ BinarySearchTree ]===" << endl;  bst.printTree();  cout << endl;   cout << "Add elem with key 400000" << endl;  CarRecord\* temp = new CarRecord (400000, "BRAND-ADDED", "OWNER-ADDED");  bst.insert\_file(400000, temp, binaryFileName);   cout << "===[ BinarySearchTree ]===" << endl;  bst.printTree();  cout << endl;   cout << "Search elem with key 910542" << endl;  bst.search(910542); } |

1. Для проверки рандомизированного древа поиска была написана функция testRBST.

|  |
| --- |
| void testRBST(){  string binaryFileName = "binary-test.txt";   RandomizedBST rbst;  rbst.Load(binaryFileName);   cout << "===[ RandomizedBST ]===" << endl;  rbst.printTree();  cout << endl;   cout << "Delete elem with key 538398" << endl;  rbst.remove(538398);   cout << "===[ RandomizedBST ]===" << endl;  rbst.printTree();  cout << endl;   cout << "Add elem with key 910000" << endl;  CarRecord\* temp = new CarRecord (910000, "BRAND-ADDED", "OWNER-ADDED");  rbst.insert(910000, temp);    cout << "===[ RandomizedBST ]===" << endl;  rbst.printTree();  cout << endl;   cout << "Search elem with key 910542" << endl;  rbst.search(910542); } |

1. **Тестирование**

Для тестирования кода создадим текстовый файл и введем следующие значения. Данный текстовый файл предоставлен на рисунке 1.

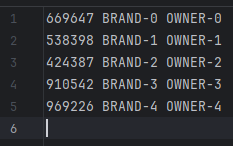


Рисунок 1. Содержание тестового файла

Вызовем функцию, которая проверит работоспособность класса для работы с бинарным файлом. Отработка данной функции предоставлена на рисунке 2.

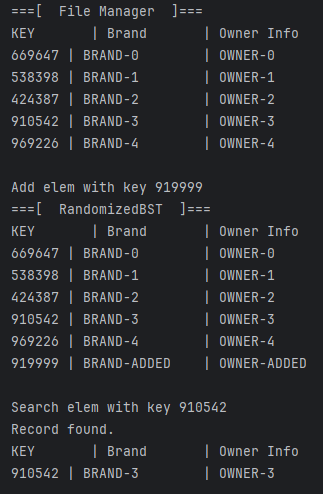


Рисунок 2. Результат, полученный после вызова функции для работы с текстом

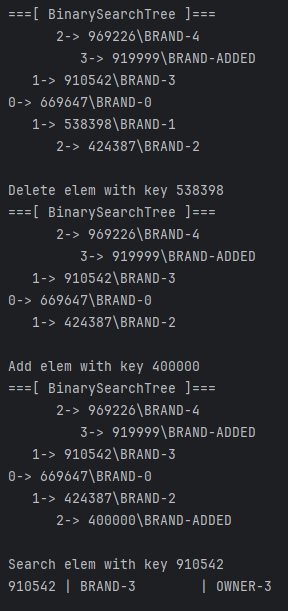


Рисунок 3. Результат отработки функции для работы с бинарным деревом поиска

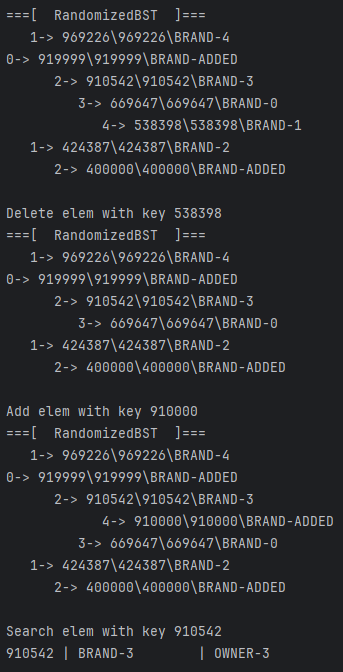


Рисунок 4. Результат отработки функции для работы с рандомизированным деревом поиска

# **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, также были получены навыки по работе с бинарным древом поиска и рандомизированным древом поиска.

# **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;  }  } } |