|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Управление файлом»** | |
|  | |
| Выполнил студент группы ИКБО-03-21 | Хречко С.В. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **Цель работы**

Получить навыки применения файловых потоков языка С++ (или файлов языка Си) по управлению текстовым и двоичным файлами.

# **Постановка задачи**

Разработать программу, которая продемонстрирует выполнение упражнений варианта. Результаты выполнения упражнения выводить на монитор.

**Задание 1.**

1. Разработать программу, управления текстовым файлом.
   1. Реализация ввода-вывода на основе файловых потоков С++: ofstream, ifstrim.
   2. Имя физического файла вводится пользователем и передается в функции обработки через параметр.
   3. Разработать функции для выполнения операций над текстовым файлом:
      1. создание текстового файла кодировки ASCII, содержащего десятичные числа по несколько чисел на строке;
      2. вывод содержимого текстового файла;
      3. добавление новой строки в конец файла;
      4. прочитать значение числа, указав его порядковый номер в файле, и вернуть его значение;
      5. определить количество чисел в файле.
   4. Разработать приложение и выполнить тестирование всех функций. Приложение должно содержать диалоговый интерфейс на основе текстового меню.
   5. Приложение должно осуществлять контроль открытия и существования файла в основной программе перед вызовом функций. Перед закрытием файла, проверять отсутствие ошибок ввода и вывода (метод good).
   6. Создать модуль и перенести в него все отлаженные функции. Исключить функции из приложения. Отладить приложение, подключив к нему модуль с функциями.
   7. Разработать функции для реализации дополнительных операций, определенных вариантом и сохранить их в модуле с остальными функциями.

Вариант № 29. Условие задания:

|  |
| --- |
| Создать новый файл из значений исходного, размещая на каждой строке нового файла количество чисел в соответствии с числами последовательности Фибоначчи. |

1. Разработать программу управления двоичными файлами с записями фиксированной длины. Общие требования: файл состоит из записей определенной структуры, согласно варианту. Записи имеют ключ, уникальный в пределах файла.
   1. Разработать структуру записи двоичного файла согласно варианту задания.
   2. Подготовить тестовые данные в текстовом файле с кодировкой ASCII, в соответствии со структурой записи варианта. При открытии файла выполнить контроль его существования и открытия.
   3. Имя файла вводит пользователь.
   4. При открытии файла обеспечить контроль существования и открытия файла.
   5. При применении механизма прямого доступа к записи файла выполнить контроль присутствия записи с заданным номером в файле.
   6. Разработать функции для выполнения операций:
      1. преобразование тестовых данных из текстового файла в двоичный файл;
      2. преобразование данных из двоичного файла в текстовый;
      3. вывод всех записей двоичного файла;
      4. доступ к записи по ее порядковому номеру в файле, используя механизм прямого доступа к записи в двоичном файле;
      5. удаление записи с заданным значением ключа, выполнить путем замены на последнюю запись.
      6. манипулирование записями в двоичном файле согласно дополнительным операциям, определенным в варианте;
   7. Сохраните функции в новом модуле.
   8. Разработать приложение, демонстрирующее выполнение всех операций, подключив к нему модуль с функциями.
2. Выполнить тестирование приложения, продемонстрировав выполнение всех операций.

Вариант № 29. Условие задания:

|  |  |
| --- | --- |
| Структура записи | Справочник банков по городам страны. Об отдельном банке хранятся данные: наименование, код банка, адрес (город), форма собственности (коммерческий или государственный). |
| Доп. операция | 1. Сформировать сведения по банкам города, с коммерческой формой собственности. 2. Заменить у банка с заданным кодом форму собственности на противоположную. |

# **Решение**

В С++ аппарат работы с файлами основан на объектах классов, хранящихся в файлах ostream и istream, fstream.

Класс файловых потоков – это иерархия, базовым классов этой иерархии является абстрактный класс с именем ios, который находится на вершине иерархии. Он используется только в качестве базового класса для остальных потоков ввода-вывода.

Поток – это логический интерфейс к файлу.

Тестовый файл – это последовательность символов, сохраненная на внешнем носителе. Текстовый файл может быть создан текстовым редактором кодировки ASCII и средствами программы.

**Функции используемые в первом задании.**

По условию задания требовалось создать файл, в котором находятся числа, по несколько на строке. Функция проверяет открытие файла, а затем записывает в файл случайные числа от 0 до 99 по четыре числа в строке, всего записывается количество чисел, передаваемое в функции в качестве параметра size.

|  |
| --- |
| int generateFile(string fileName, int size){      ofstream fout(fileName);      if (!fout.is\_open()){          cout << "file doesn't exist" << endl;          fout.close();          return 1;      }      for (int i = 0; i < size; i++){          fout << rand() % 100 << " ";          if((i+1) % 4 == 0) fout << "\n";      }      fout << "\n";      fout.close();      return 0;  } |

По условию задания требовалось выводить содержимое файла, функция проверяет открытие файла, а затем построчно в цикле выводит содержимое.

|  |
| --- |
| int readFile(string fileName){      ifstream fin(fileName);      if (!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      while (fin.good()){          string tmp;          getline(fin, tmp);          cout << tmp << endl;      }      fin.close();      return 0;  } |

По условию задания требовалось добавлять в конец файла новую строку, функция принимает на вход строку, которую нужно добавить и после проверки открытия файла, по переданному имени, дописывает строку в конец, для этого файл открывается для дописывания, а не для перезаписи.

|  |
| --- |
| int addToFile(string fileName, string toAdd){      ofstream fout(fileName, ios::app);      if (!fout.is\_open()){          cout << "file doesn't exist" << endl;          fout.close();          return 1;      }      fout << toAdd << " " << endl;      fout.close();      return 0;  } |

По условию задания требовалось найти и вернуть число по его порядковому номеру, функция проверяет открытие файла, затем в к цикле построчно читает файл и разбивает строки на слова. Затем каждое слово проверяется на то, является ли оно числом, выполняется с помощью функции stoi и блока try catch. Если слово является числом то мы увеличиваем счетчик на единицу, когда счетчик дошел до переданного значения index, мы понимаем, что нашли искомое число и возвращаем его, если счетчик с переданным значением так и не сравняется до конца файла, то это значит, что пользователь передал порядковый номер больше чем количество чисел в файле, а значит такого искомого числа не существует.

|  |
| --- |
| string findNumber(string fileName, int index){      ifstream fin(fileName);      if (!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return "";      }      int count = 1;      while (fin.good()){          string tmp;          getline(fin, tmp);          string word = "";          for (int i = 0; i < tmp.length(); i++){              if (tmp[i] != ' '){                  word += tmp[i];              }              else{                  try{ //checking if the word is a number                      int num = stoi(word);                      if (count != index){                          count++;                      }                      else{                          count++;                          return word;                      }                  }                  catch(std::invalid\_argument const& ex){                    }                  word = "";              }          }      }      fin.close();      return "";  } |

По условию задания требовалось определить количество чисел в файле. Функция проверяет открытие файла, затем, аналогично функции поиска числа по индексу в цикле файл считывается построчно и строки разбиваются на слова, затем слова проверяются на бытие числом, но в отличие от функции поиска числа здесь мы всегда проходим до конца файла, и возвращаем значение счетчика.

|  |
| --- |
| int numbersCount(string fileName){      ifstream fin(fileName);      if (!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return -1;      }      int count = 0;      while (fin.good()){          string tmp;          getline(fin, tmp);          string word = "";          for (int i = 0; i < tmp.length(); i++){              if (tmp[i] != ' '){                  word += tmp[i];              }              else{                  try{ //checking if the word is a number                      int num = stoi(word);                      count++;                  }                  catch(std::invalid\_argument const& ex){                    }                  word = "";              }          }      }        fin.close();      return (count);  } |

По условию задания варианта требовалось на основе созданного текстового файла с числами создать еще один, в котором разместить числа из файла таким образом, чтобы на строках нового файла были расположены в количестве равном элементам последовательности Фибоначчи. Функция проверяет открытие файла, а открывает файл, который будет считаться новым или же создает его, если его него не существует. Затем, работая с числами в файле по аналогии с двумя предыдущими функциями, записывает числа в количестве последовательности Фибоначчи. Для этого используются четыре числовые переменные a, b, c и j. Первые три используются для подсчета элементов последовательности Фибоначчи, а последняя является счетчиком записываемых на одну строчку слов, она сравнивается с переменной c, и когда становится равна ей, мы начинаем записывать числа на новую строку и рассчитываем следующий жлемент последовательности Фибоначчи.

|  |
| --- |
| int fibbonachiFile(string fileName){      ofstream fout(fileName.substr(0, fileName.length() - 4) + "\_fibbonachi.txt");      ifstream fin(fileName);      if (!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      int a = 0, b = 0, c = 1;      int j = 0;      while(fin.good()){          string tmp;          getline(fin, tmp);          string word = "";          for (int i = 0; i < tmp.length(); i++){              if (tmp[i] != ' '){                  word += tmp[i];              }              else{                  try{ //checking if the word is a number                      int num = stoi(word);                      fout << num << " ";                      j++;                      if(j >= c){                          fout << "\n";                          a = b;                          b = c;                          c = a + b;                          j = 0;                      }                  }                  catch(std::invalid\_argument const& ex){                    }                  word = "";              }          }      }      fin.close();      fout.close();      return 0;  } |

**Функции используемые во втором задании.**

По условию задания требовалось преобразовать данные из текстового файла в двоичный, функция проверяет открытие текстового файла, а бинарный файл создает, если его нет. Далее построчно читает текстовый файл и разбивает его на слова, разделителем является точка с запятой. Далее считанные слова записываются в структуру, по четыре поля на одну структуру, так как предполагается, что записи в файле соответствует структуре данных. Далее структура записывается в бинарный файл, для этого файл открыт для записи, как бинарный.

|  |
| --- |
| int toBinary(string fileName, string fileNameBin){//read from text file and write to binary file      ifstream fin(fileName);      fstream fout(fileNameBin, ios::out|ios::binary|ios::trunc);      if(!fin.is\_open() || !fout.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      while(fin.good()){          Bank bank;          vector<string> tokens;          for (string each; getline(fin, each, ';'); tokens.push\_back(each));          for(int i = 0; i < tokens.size(); i++){              //cout << tokens.size() << " " << tokens[i] << endl;              if(i%4 == 0) bank.id = stoi(tokens[i]);              else if(i%4 == 1) strcpy(bank.name, tokens[i].c\_str());              else if(i%4 == 2) strcpy(bank.city, tokens[i].c\_str());              else if(i%4 == 3){                  bank.commercialOrState = (tokens[i] == "true") ? true : false;                  fout.write((char\*)&bank, sizeof(bank));              }          }        }      fin.close();      fout.close();      return 0;  } |

По условию задания требовалось преобразовать данные из бинарного файла в текстовый. Функция проверяет открытие файлов, далее считывает структуры данных из бинарного файла, затем записывает в текстовый файл в том же формате в пределах одной записи, разделитель между полями точка с запятой. Для удобочитаемости при переводе из бинарного файла в текстовый добавляются переносы строк. По условиям задачи не требовалось приводить данные нового текстового файла в абсолютно такой же формат, что и исходные данные. Поэтому в программе используется три файла: исходный бинарный и текстовый, текстовый и исходный не могут быть одним и тем же файлом.

|  |
| --- |
| int toText(string fileNameBin, string fileName){//read from binary file and write to text file      fstream fin(fileNameBin, ios::in|ios::out|ios::binary);      ofstream fout(fileName);        if(!fin.is\_open() || !fout.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      Bank bank;      while(fin.read((char\*)&bank, sizeof(bank))){          fout << bank.id << ";" << bank.name << ";" << bank.city << ";" << (bank.commercialOrState? "true" : "false") << endl;      }      fin.close();      fout.close();      return 0;  } |

По условию задания требовалось выводить информацию из бинарного файла на экран. Функция проверяет открытие файла и открывает его, как бинарный файл, далее считывает записи в файле соответствующие структурам данных и форматировано выводит информацию на экран.

|  |
| --- |
| int printBinary(string fileNameBin){//print from binary file      fstream fin(fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      while(fin.read((char\*)&bank, sizeof(bank))){          cout << bank.id << ";" << bank.name << ";" << bank.city << ";" << (bank.commercialOrState? "true" : "false") << endl;      }      fin.close();      return 0;  } |

По условию задания требовалось получать запись по порядковому номеру. Функция принимает переменную по ссылке, в которую положит найденную запись, а возвращает код, сообщающий об успешности или неуспешности выполнения задачи. Функция проверяет открытие файла, после чего перемещает указатель чтения файла на требуемое количество записей, и считывает информацию о записи, если считывание было не успешным, это означает, что в файле нет записи с переданным порядковым номером, если же чтение успешно, то функция присваивает переданной по ссылке переменной полученную запись.

|  |
| --- |
| int getBank(string fileNameBin, Bank &bank, int ind){//get bank by ind from binary file      //returns 0 if bank was found, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin(fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }        fin.seekg(sizeof(Bank)\*(ind-1), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      bank = readBank;      fin.close();      return 0;  } |

По условию задания требовалось удалить запись по ее ключу, в нашем варианте задания это id банка, путем замены записи на последнюю. Функция проверяет открытие файла, далее ищет в нем последнюю запись и сохраняет ее. После чего в цикле, начиная с начала файла, считывает записи и сравнивает id записи с искомым, когда требуемый id найден, программа перезаписывает запись с этим id сохраненной последней записью. Если в файле искомый id не найден, то функция возвращает соответствующий код.

|  |
| --- |
| int deleteBank(string fileNameBin, int id){//delete bank by id from binary file      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin(fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }        fin.seekg(-sizeof(Bank), ios::end);      Bank lastBank;      fin.read((char\*)&lastBank, sizeof(lastBank));      Bank bank;      fin.seekg(0, ios::beg);      while(fin.read((char\*)&bank, sizeof(bank))){          if(bank.id == id){              fin.seekp(-sizeof(bank), ios::cur);              fin.write((char\*)&lastBank, sizeof(lastBank));              fin.close();              return 0;          }      }      fin.close();      return 2;  } |

По условию задания требовалось сменить форму собственности у банка с заданным id. Функция проверяет открытие файла, затем аналогично предыдущей функции ищет запись с нужным id, по нахождении, меняет форму собственности на противоположную, так как форма собственности хранится как булева переменная, то смена выполняется присваиванием отрицания самого поля. Если id не найден, то функция возвращает соответственный код.

|  |
| --- |
| int changeOwnership(string fileNameBin, int id){//change bank commercialOrState by id from binary file      //returns 0 if bank was changed, 1 if file doesn't exist, 2 if bank wasn't found      fstream finoutBin(fileNameBin, ios::in|ios::out|ios::binary);      if(!finoutBin.is\_open()){          cout << "file doesn't exist" << endl;          finoutBin.close();          return 1;      }      Bank bank;      finoutBin.seekg(0, ios::beg);      while(finoutBin.read((char\*)&bank, sizeof(bank))){          if(bank.id == id){              bank.commercialOrState = !bank.commercialOrState;              finoutBin.seekp(-sizeof(bank), ios::cur);              finoutBin.write((char\*)&bank, sizeof(bank));              finoutBin.close();              return 0;          }      }      finoutBin.close();      return 2;  } |

По условию задания требовалось вывести все коммерческие банки определенного города. Функция проверяет открытие файла, после чего читает записи в файле, проверяя условие, что город в записи соответствует переданному и банк является коммерческим, если условие выполняется, то такая запись печатается, функция проверяет все записи в файле.

|  |
| --- |
| int printBanksOfCity(string fileNameBin, string city){//print commercial banks of city from binary file      fstream fin(fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      while(fin.good()){          Bank bank;          fin.read((char\*)&bank, sizeof(bank));          if(bank.city == city && bank.commercialOrState){//banks of city and commercial              cout << bank.id << ";" << bank.name << ";" << bank.city << ";" << (bank.commercialOrState? "commercial" : "state") << endl;          }      }      fin.close();      return 0;  } |

Программа, решающая задачу 1, предлагает ввести имя файла и сгенерировать его, далее предлагается ввести данные требуемые для выполнения заданий, как дописать строку в конец файла и найти число по его индексу (рис 1).

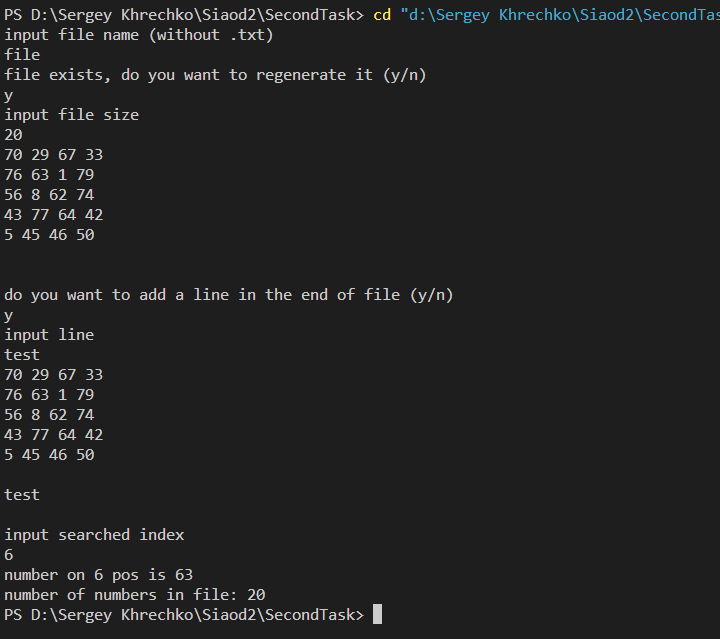


Рисунок 1. Интерфейс программы

Программа, решающая задачу 2, предлагает ввести имена файлов и предлагает ввести данные требуемые для проверки выполнения заданий, таких как найти банк по порядковому номеру или вывести коммерческие банки определенного города (рис 2).

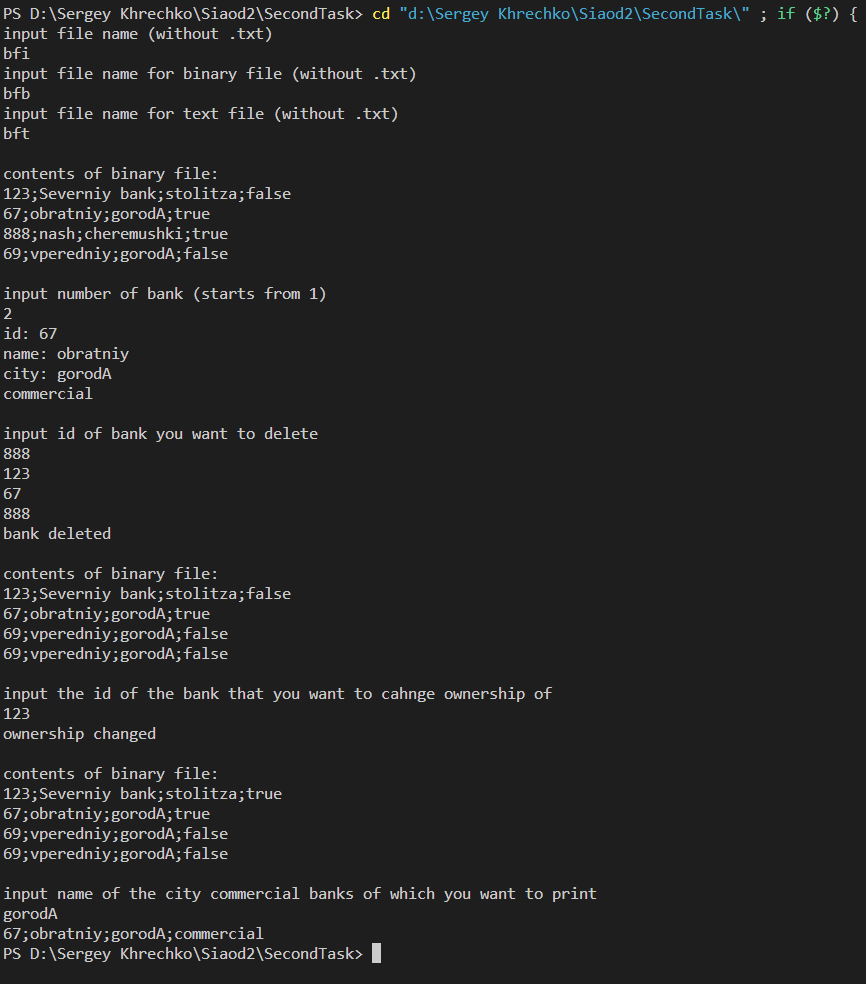


Рисунок 2. Интерфейс программы

# **Тестирование**

**Тестирование функций первой задачи.**

Для тестирования функции генерации файла сначала используем уже существующий и попросим сгенерировать его заново, на (рис 3 и 4) видно, что файл действительно теперь содержит 25 чисел.

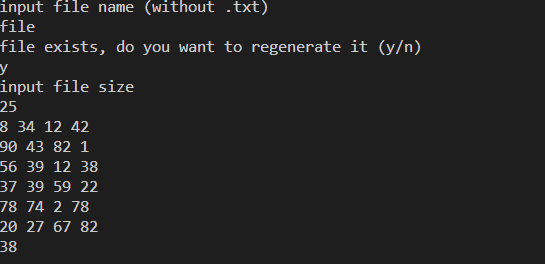


Рисунок 3. Тестирование функции создания файла

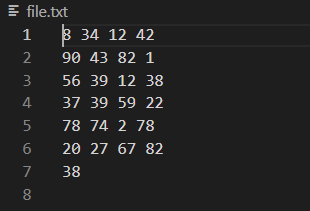


Рисунок 4. Тестирование функции создания файла

Теперь используем несуществующий файл, как видно на (рис. 5 и 6) программа сказала, что файл не существует и предложила создать его, после согласия файл был создан с нужным количеством чисел.

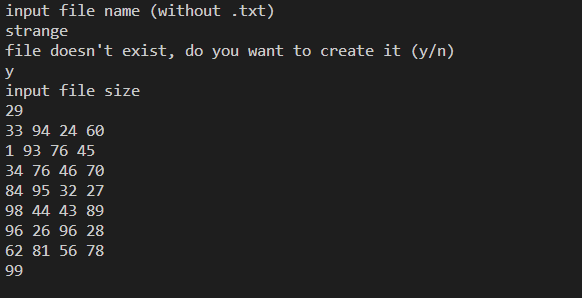


Рисунок 5. Тестирование функции создания файла

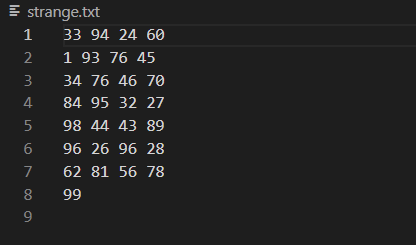


Рисунок 6. Тестирование функции создания файла

Функция чтения файла уже использовалась в тестировании функции создания файла, проведем еще один тест с файлом strange, на этот раз пусть будет 15 чисел, как мы можем видеть, на стандартный поток вывода выводится содержимое файла.

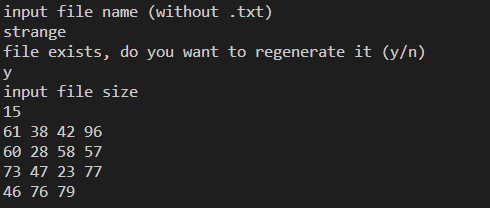


Рисунок 7. Тестирование функции чтения

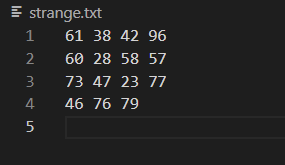


Рисунок 8. Тестирование функции чтения

Протестируем функцию добавления строки в конец файла, программа предлагает нам ввести строку. Вводим какую-нибудь строку и можем наблюдать, что, действительно, строка была добавлена в файл (рис. 9 и 10).

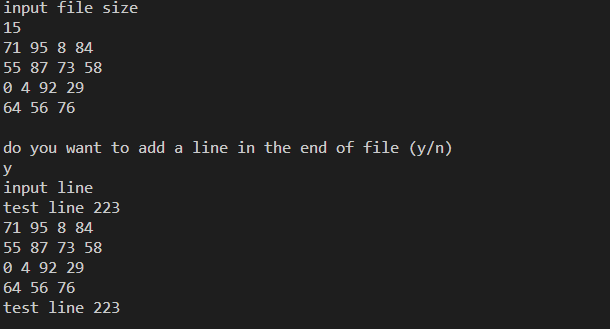


Рисунок 9. Тестирование функции добавления строки

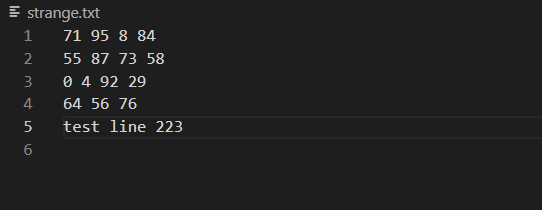


Рисунок 10. Тестирование функции добавления строки

Теперь попробуем пустую строку, она не совсем, пустая, символ переноса строки там все-таки есть, иначе это была бы не строка, как мы можем наблюдать в файле стало на одну строку больше, значит эксперимент успешен и программа работает корректно.

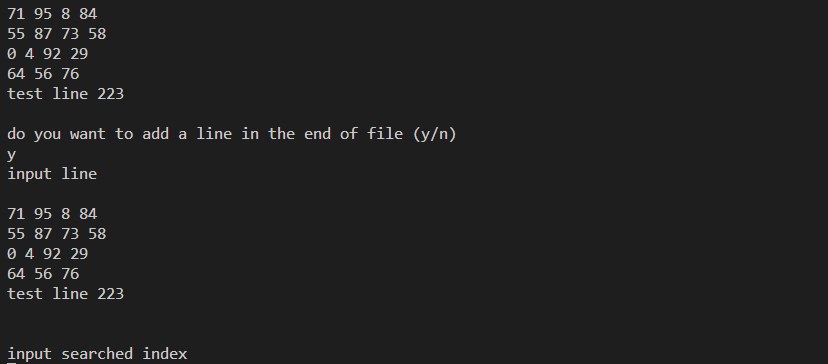


Рисунок 11. Тестирование функции добавления строки

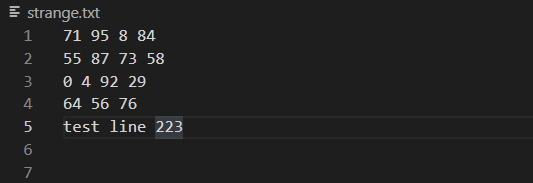


Рисунок 12. Тестирование функции добавления строки

Протестируем функцию поиска числа. Для этого приведено три теста (рис. 13-15), в первом и третьем производится попытка найти число с индексами -1 и 17, так как в файле 16 чисел и два слова, то по индексам -1 и 17 мы ожидаем не найти ничего, во втором тесте задан индекс 16, число на 16 позиции стоит после двух слов не являющихся числами, это позволит проверить реагирует ли функция на слова, не являющиеся числами. Как можно видеть по результатам теста, функция работает корректно во всех трех случаях.



Рисунок 13. Тестирование функции поиска числа

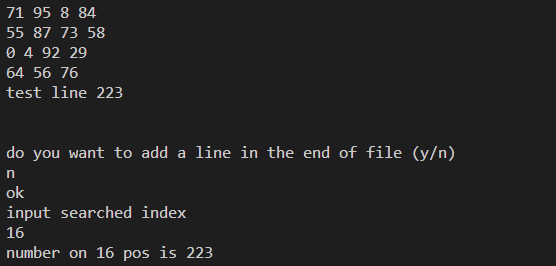


Рисунок 14. Тестирование функции поиска числа

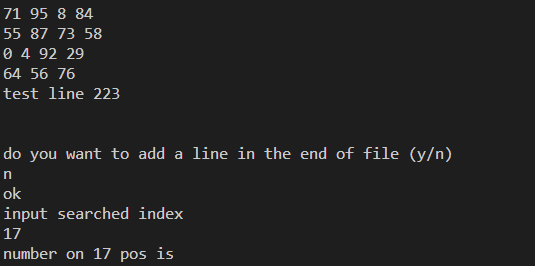


Рисунок 15. Тестирование функции поиска числа

Протестируем функцию поиска количества чисел в файле, хорошо подойдет тест из тестирования предыдущей функции. Здесь в файле помимо чисел также присутствуют слова, из тестирования видно, что функция не считает слова за числа и выводит верное число чисел.

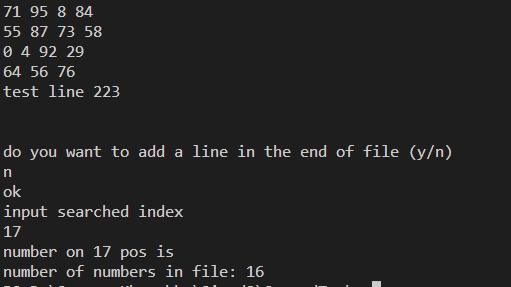


Рисунок 16. Тестирование функции поиска количества чисел

Протестируем функцию создания нового файла по дополнительному заданию из варианта. В последней строке может оказаться количество чисел меньшее, чем должно было бы быть в соответствии с последовательностью Фибоначчи, так как числе может попросту не хватать, по результатам видно, что программа работает корректно (рис. 17-18).

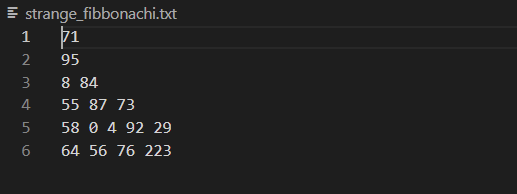


Рисунок 17. Тестирование функции создания файла Фибоначчи

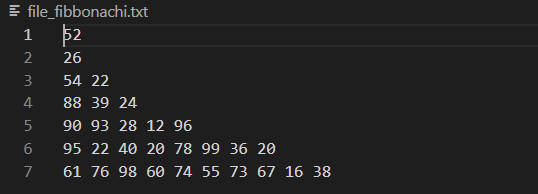


Рисунок 17. Тестирование функции создания файла Фибоначчи

**Тестирование функций второй задачи.**

Так выглядит файл с исходными данными, используемыми для проведения тестирования (рис. 18).

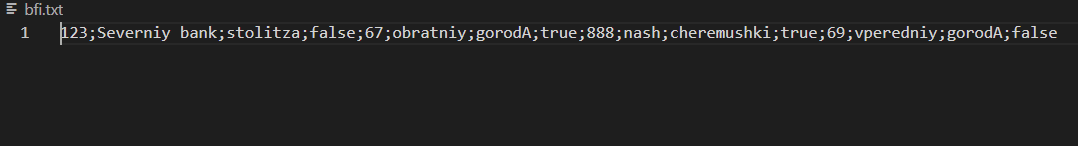


Рисунок 18. Исходный файл

Протестируем функцию перевода исходных данных в бинарный файл, для проверки будем использовать также и функцию чтения бинарного файла (рис. 19) Затем добавим в исходный файл еще одну запись (рис. 20). Мы можем наблюдать, что обе функции работают корректно.

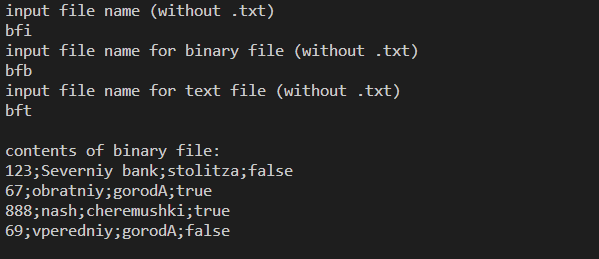


Рисунок 19. Тестирование функций перевода в бинарный файл и вывода

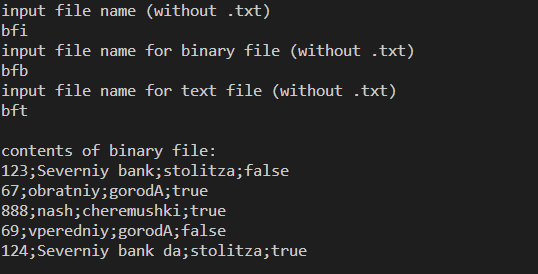


Рисунок 20. Тестирование функций перевода в бинарный файл и вывода

Протестируем функцию перевода данных из бинарного файла в текстовый. В созданном текстовом файле после выполнения программы оказалось представленное на (рис. 21). Мы можем наблюдать, что в нем то же самое, что и было выведено функцией чтения бинарного файла, следовательно, функция перевода бинарного файла в текстовый работает корректно.

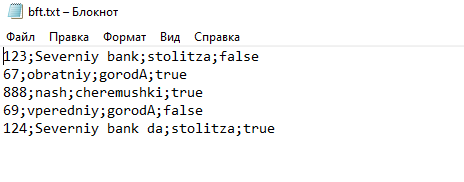


Рисунок 21. Содержимое текстового файла

Протестируем функцию нахождения записи по порядковому номеру, мы можем наблюдать как и успешное нахождение (рис. 22), так и не успешное, так как порядковый номер слишком велик (рис. 23), в обоих случаях программа работает корректно.

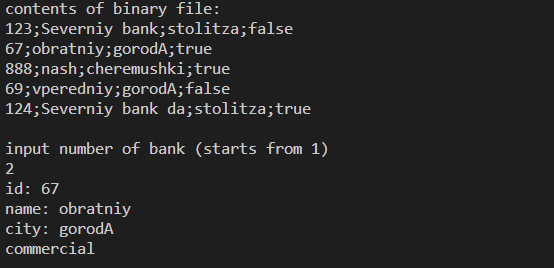


Рисунок 22. Тестирование функции нахождения банка

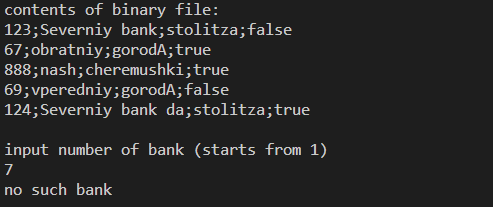


Рисунок 23. Тестирование функции нахождения банка

Протестируем функцию удаления. В первом случае (рис. 24) мы удаляем существующий банк, программа сообщает, что банк удален и при следующем выводе мы видим, что как и требовалось по заданию удаляемый банк был заменен на последнюю запись. Во втором случае (рис. 25) мы пытаемся удалить несуществующий банк, программа сообщает, что банка не существует и изменений при выводе не наблюдается.

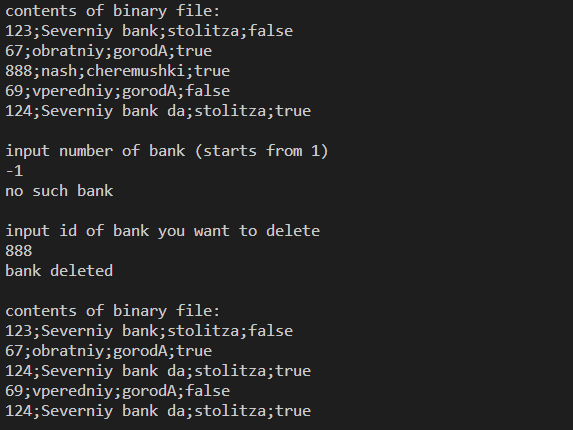


Рисунок 24. Тестирование функции удаления банка

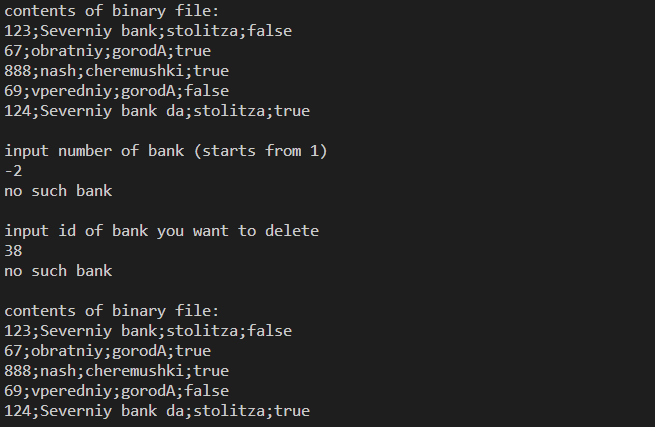


Рисунок 25. Тестирование функции удаления банка

Протестируем функцию смены формы собственности банка. Аналогично предыдущим тестированиям проверим смену формы собственности у существующего банка (рис. 26) и несуществующего (рис. 27). В первом случае меняли собственность у банка с id 124, на выводе видно, что последнее поле отвечающее за форму собственности сначала имело значение true, а после false. Во втором случае программа вывело сообщение об отсутствии банка, вывод: функция работает корректно.

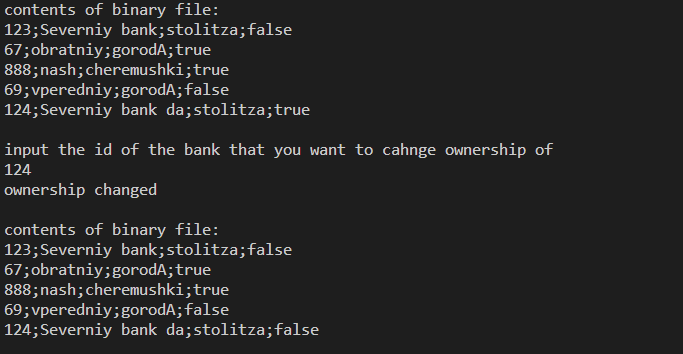


Рисунок 26. Тестирование функции смены собственности банка

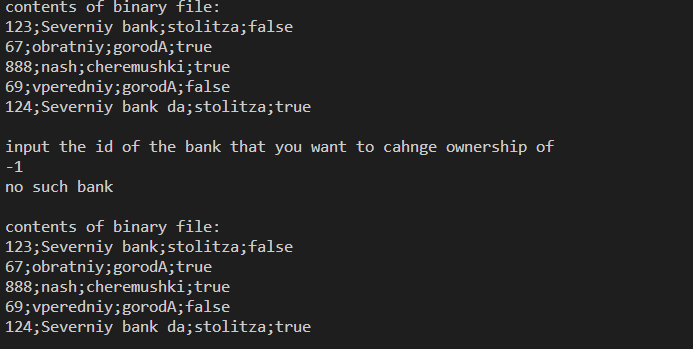


Рисунок 27. Тестирование функции смены собственности банка

Протестируем функцию вывода коммерческих банков определенного города. На (рис. 28-30) представлены результаты тестирования. В первом случае в городе gorodA есть только один город с коммерческой формой собственности, это банк 67, во втором же случае мы поменяли форму собственности у банка 69 на коммерческую, и теперь в gorodA есть два коммерческих банка, что нам и вывела программа. Из третьего случая мы видим, что, как и ожидалось, в городе, в котором нет ни одного банка вообще, программа вывела ничего, потому что требуемых банков нет.

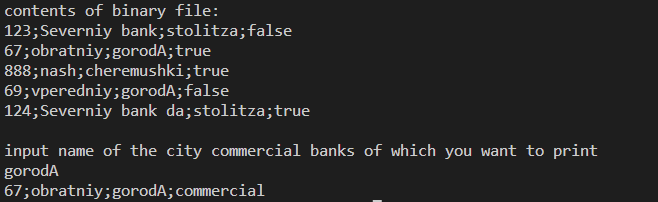


Рисунок 28. Тестирование функции вывода коммерческих банков города

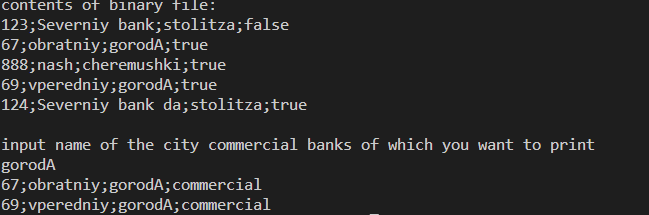


Рисунок 29. Тестирование функции вывода коммерческих банков города

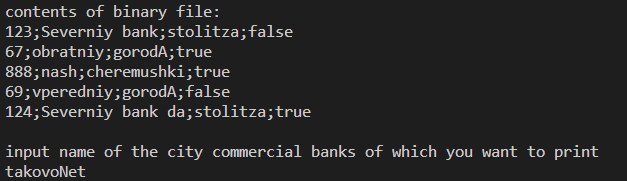


Рисунок 30. Тестирование функции вывода коммерческих банков города

Из результатов выполнения программы видно:

1. \*выводы, которые можно сделать на основе полученных результатов\*

# **Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы работы с поразрядными операциями и их реализацию на языке программирования C++
2. Научился программировать автоматическое тестирование простых программ

# **Исходный код программы**

|  |
| --- |
| void bitmult()  {  }  int main()  {  } |