Лабораторная работа № 6

Тема работы: Потоки ввода/вывода. Работа с файлами.

Содержание: Организация работы с текстовыми и бинарными файлами. Файлы последовательного и произвольного доступа.

Цель работы: Изучить стандартную библиотеку iostream. Научиться работать с файлами. Записать/прочитать данные в текстовом и бинарном виде.

Потоки ввода/вывода

В языке C++ для организации работы с файлами используются классы потоков ifstream(ввод), ofstream(вывод) и fstream(ввод и вывод) (рис. 6.1).

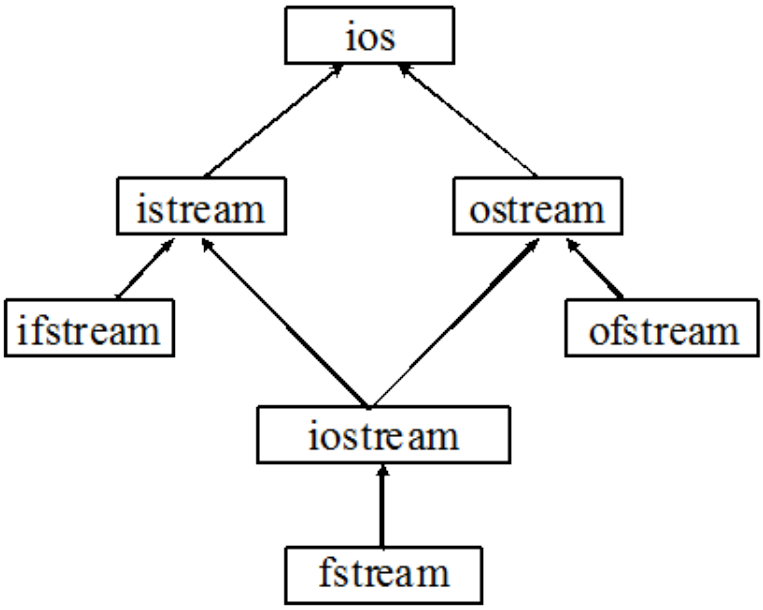

Рисунок 6.1. – часть иерархии классов потоков ввода-вывода

Перечисленные классы являются производными от *istream*, *ostream* и *iostream* соответственно. Операции ввода−вывода выполняются так же, как и для других потоков, то есть компоненты - функции, операции и манипуляторы могут быть применены и к потокам файлов. Различие состоит в том, как создаются объекты и как они привязываются к требуемым файлам.

В C++ файл открывается путем стыковки его с соответствующим потоком.

Рассмотрим организацию связывания потока с некоторым файлом. Для этого используются конструкторы классов *ifstream* и *ofsream*:

```
ofstream(const char* Name, int nMode = ios::out, int nPot =
filebuf::openprot);
ifstream(const char* Name, int nMode = ios::in, int nPot =
filebuf::openprot);
```

Первый аргумент определяет имя файла (единственный обязательный параметр).

Второй аргумент задает режим для открытия файла и представляет *битовое ИЛИ* (|) величин:

ios::app при записи данные добавляются в конец файла, даже если текущая позиция была перед этим изменена функцией *ostream::seekp*;

ios::ate указатель перемещается в конец файла. Данные записываются в текущую позицию (произвольное место) файла;

ios::in поток создается для ввода; если файл уже существует, то он сохраняется;

ios::out поток создается для вывода(по умолчанию для всех *ofstream объектов*); если файл уже существует, то он уничтожается;

ios::trunc если файл уже существует, его содержимое уничтожается (длина равна нулю). Этот режим действует по умолчанию, если *ios::out* установлен, а *ios::ate, ios::app* или *ios::in* не установлены;

ios::nocreate если файл не существует, функциональные сбои;

ios::noreplace если файл уже существует, функциональные сбои;

ios::binary ввод−вывод будет выполняться в двоичном виде (по умолчанию текстовой режим).

Возможны следующие комбинации перечисленных выше величин:

ios::out | *ios::trunc* удаляется существующий файл и(или) создается для записи;

ios::out | *ios::app* открывается существующий файл для дозаписи в конец файла;

ios::in | *ios::out* открывается существующий файл для чтения и записи;

ios::in | *ios::out* | *ios::trunc* существующий файл удаляется и (или) создается для чтения и записи;

ios::in | *ios::out* | *ios::app* открывается существующий файл для чтения и дозаписи в конец файла.

Третий аргумент – данное класса *filebuf* используется для установки атрибутов доступа к открываемому файлу.

Возможные значения *nProt*:

filebuf::sh_compat режим совместного использования;

filebuf::sh_none режим *Exclusive :* не используется совместно;

filebuf::sh_read режим совместного использования для чтения;

filebuf::sh_write режим совместного использования для записи.

Для комбинации атрибутов *filebuf::sh_read* и *filebuf::sh_write* используется операция логическое ИЛИ (||).

В отличие от рассмотренного подхода можно создать поток, используя конструктор без аргументов. Позже вызвать функцию *open*, имеющую те же аргументы, что и конструктор, при этом второй аргумент не может быть задан по умолчанию:

```
void open(const char* name, int mode, int prot = fileout::openprot);
```

Организация работы с текстовыми и бинарными файлами

Только после того как поток создан и соединен с некоторым файлом (используя либо конструктор с параметрами, либо функцию open), можно выполнять ввод информации в файл или вывод из файла.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class String
{
  char *st;
  int size;
public:

  String(char *ST, int SIZE) : size(SIZE)
  {
    st = new char[size];
    strcpy_s(st, size+1, ST);
  }
  ~String() { delete[] st; }
  String(const String &s) // копирующий конструктор необходим, так как
  {                       // при перегрузке << в функцию operator переда-
    st = new char[s.size]; // ется объект, содержащий указатель на строку,
    strcpy_s(st,s.size+1, s.st); // а в конце вызовется деструктор для
  }                              // объекта obj
  friend ostream &operator<<(ostream &, const String);
  friend istream &operator>>(istream &, String &);
};

ostream &operator<<(ostream &out, const String obj)
{
  out << obj.st << endl;
  return out;
}

istream &operator>>(istream &in, String &obj)
{
  in >> obj.st;
  return in;
}

int main()
{
  String s("asgg", 10), ss("aaa", 10);
```

```
  int state;
  ofstream out("file");

  if (!out)
  {
    cout << "ошибка открытия файла" << endl;
    return 1;       // или exit(1)
  }
  out << "123" << endl;
  out << s << ss << endl;       //  запись в файл
  ifstream in("file");
  while (in >> ss)              //  чтение из файла
  {
    cout << ss << endl;
  }
  in.close();
  out.close();
  return 0;
}
```

В приведенном примере в классе содержится копирующий конструктор, так как в функцию *operator* передается объект *obj*, компонентой которого является строка. В инструкции *cout << s << ss* копирующий конструктор вызывается вначале для объекта *ss*, затем для s, после этого выполняется перегрузка в порядке, показанном ранее. При завершении каждой из функций *operator<<* (вначале для *s*, затем для *ss*) будет вызван деструктор.

В операторе *if (!out)* вызывается (как и ранее для потоков) функция *ios::operator!* для определения, успешно ли открылся файл.

Условие в заголовке оператора *while* автоматически вызывает функцию класса *ios* перегрузки операции *void **:

```
operator void *() const {
    if (state&(badbit | failbit)) return 0;
    return (void *)this;
}
```

т.е. выполняется проверка; если для потока устанавливается *failbit* или *badbit*, то возвращается *0*.

Файлы последовательного доступа

В C++ файлу не предписывается никакая структура. Для последовательного поиска данных в файле программа обычно начинает считывать данные с начала файла до тех пор, пока не будут считаны требуемые данные. При поиске новых данных этот процесс вновь повторяется.

Данные, содержащиеся в файле последовательного доступа, не могут быть модифицированы без риска разрушения других данных в этом файле. Например, если в файле содержится информация

```
Коля 12 Александр 52
```

то при модификации имени Коля на имя Николай может получиться следующее:

```
НиколайАлександр 52
```

Аналогично в последовательности целых чисел 12 –1 132 32554 7 для хранения каждого из них отводится *sizeof(int)* байт. А при форматированном выводе их в файл они занимают различное число байт. Следовательно, такая модель ввода−вывода неприменима для модификации информации на месте. Эта проблема может быть решена перезаписью (с одновременной модификацией) в новый файл информации из старого. Это сопряжено с проблемой обработки всей информации при модификации только одной записи.

Следующая программа выполняет перезапись информации из одного файла в два других, при этом в первый файл из исходного переписываются только числа, а во второй − вся остальная информация.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

void error(char *s1, char *s2 = "")  // вывод сообщения об ошибке
{
  cerr << s1 << "  " << s2 << endl;  // при открытии потока для файла
  exit(1);
}

int main(int argc, char **argv)
{
  char *buf = new char[20];
  int i;
  ifstream f1;                    // входной поток
  ofstream f2, f3;                // выходные потоки
  f1.open(argv[1], ios::in);      // открытие исходного файла
  if (!f1)                        // проверка состояния потока
  error("ошибка открытия файла", argv[1]);
  f2.open(argv[2], ios::out);     // открытие 1 выходного файла
  if (!f2) error("ошибка открытия файла", argv[2]);
  f3.open(argv[3], ios::out);     // открытие 2 выходного файла
  if (!f3) error("ошибка открытия файла", argv[3]);
  while (f1.getline(buf, 20, ' '))  // считывание в буфер до 20 символов
  {
    if (int n = f1.gcount() <= 1)   // число реально считанных символов
      continue;                     // пропуск последующих пробелов
                    // проверка на только цифровую строку
    for (i = 0; *(buf + i) && (*(buf + i) >= '0' && *(buf + i) <= '9');
i++);
    if (!*(buf + i))  f2 << ::atoi(buf) << ' ';    // преобразование в
  // число и запись в файл f2
    else f3 << buf << ' ';            // просто выгрузка буфера в файл f3
  }
  delete[] buf;
  f1.close();                          // закрытие файлов
  f2.close();
  f3.close();
}
```

В программе для ввода имен файлов использована командная строка, первый параметр – имя файла источника(входного), а два других – имена файлов приемников(выходных).Для работы с файлами использованы функции

– *open*, *close*, *getline и gcount*. Более подробное описание функций приведено ниже.

Другой пример использования файлов последовательного доступа можно продемонстрировать на примере программы слияния двух упорядоченных текстовых файлов в третий так же упорядоченный файл.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
using namespace std;

void error(char *s1, char *s2 = "")  // вывод сообщения об ошибке
{
  cout << s1 << "  " << s2 << endl;      // при открытии потока для файла
  exit(1);
}

int main()
{
  char *buf = new char[20];
  int i1, i2, n1, n2;
  ifstream f1, f2;                          // входной поток
  ofstream f3;                      // выходные потоки
  f1.open("f1.txt", ios::in);       // открытие 1 исходного файла
  if (!f1)                           // проверка состояния потока
    error("ошибка открытия файла", "f1.txt");
  f2.open("f2.txt", ios::in);     // открытие 2 исходного файла
    if (!f2) error("ошибка открытия файла", "f2.txt");
  f3.open("f3.txt", ios::out);     // открытие  выходного файла
  if (!f3) error("ошибка открытия файла", "f3.txt");
  f1.getline(buf, 5, ' ');
  if (n1 = f1.gcount()) { buf[n1 - 1] = '\0'; i1 = atoi(buf); }
  f2.getline(buf, 5, ' ');
  if (n2 = f1.gcount()) { buf[n2 - 1] = '\0'; i2 = atoi(buf); }
  while (n1 && n2)
  {
    while (i1 < i2 && n1)  // считывание в буфер до 5 символов
    {
      f3 << i1 << ' ';
      f1.getline(buf, 5, ' ');
      if (n1 = f1.gcount()) { buf[n1 - 1] = '\0'; i1 = atoi(buf); }
    }
    while (i1 >= i2 && n2)  // считывание в буфер до 5 символов
    {
      f3 << i2 << ' ';
      f2.getline(buf, 5, ' ');
      if (n2 = f2.gcount()) { buf[n2 - 1] = '\0'; i2 = atoi(buf); }
    }
  }
  while (n1)  // считывание в буфер до 20 символов
  {
    f3 << i1 << ' ';
    f1.getline(buf, 5, ' ');
    if (n1 = f1.gcount()) { buf[n1 - 1] = '\0'; i1 = atoi(buf); }
  }
  while (n2)  // считывание в буфер до 20 символов
  {
    f3 << i2 << ' ';
    f2.getline(buf, 5, ' ');
    if (n2 = f2.gcount()) { buf[n2 - 1] = '\0'; i2 = atoi(buf); }
```

```
  }
  delete[] buf;
  f1.close();                              // закрытие файлов
  f2.close();
  f3.close();
}
```

Использование функций *seekg*, *tellg* позволяет позиционировать текущую позицию в файле, то есть осуществлять прямой доступ в файл. Ниже приведена программа, выполняющая ввод символов в файл с их одновременным упорядочиванием(при вводе) по алфавиту. Обмен информацией с файлом осуществляется посредством функций *get* и *put*.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
using namespace std;

void error(char *s1, char *s2 = "")
{
  cerr << s1 << "  " << s2 << endl;
  exit(1);
}

int main()
{
  char c, cc;
  int n;
  fstream f;                          // выходной поток
  long p, pp;                         // streampos p,pp;
  f.open("aaaa", ios::in | ios::out | ios::trunc); // открытие выходного
  // файла
  if (!f) error("ошибка открытия файла", "aaaa");
  f.seekp(0);               // установить текущий указатель в начало потока
  while (1)
  {
    cin >> c;                              // ввод очередного символа
    if (c == 'q' || f.bad()) break;
    f.seekg(0, ios::beg);           // перемещение указателя в начало файла
    while (1)
    {
      if (((cc = f.get()) > c) || (f.eof()))
      {
        if (f.eof())                // в файле нет символа, большего  c
        {
          f.clear(0);
          p = f.tellg();            // позиция EOF в файле
        }
        else
        {
          p = f.tellg();    p--;    // позиция первого символа большего c
          f.seekg(-1, ios::end);  // указатель на последний элемент в файле
          pp = f.tellg();           // позиция последнего элемента в файле
          while (p <= pp)           // сдвиг в файле на один символ
          {
            cc = f.get();
            f.seekp(pp + 1, ios::beg);
            f.put(cc);
            if (--pp >= 0)          // проверка на невыход за начало файла
              f.seekg(pp);
          }
```

```
        }
        f.seekp(p);     // позиционирование указателя в позицию p
        f.put(c);         // запись введенного символа c в файл
        break;
      }
    }
  }
  f.close();
  return 0;
}
```

Каждому объекту класса *istream* соответствует указатель *get*(указывающий на очередной вводимый из потока байт) и указатель *put*(соответственно на позицию для вывода байта в поток). Классы *istream* и *ostream* содержат по две перегруженные компоненты - функции для перемещения указателя в требуемую позицию в потоке (связанном с ним файле). Такими функциями являются *seekg*(переместить указатель для извлечения из потока) и *seekp*(переместить указатель для помещения в поток).

```
istream& seekg(streampos pos);
istream& seekg(streamoff off, ios::seek_dir dir);
```

Сказанное выше справедливо и для функций *seekp*. Первая функция перемещает указатель в позицию *pos* относительно начала потока. Вторая перемещает соответствующий указатель на off(целое число) байт в трех возможных направлениях : *ios::beg*(от начала потока), *ios::cur*(от текущей позиции) и *ios::end*(от конца потока). Кроме рассмотренных функций в этих классах имеются еще функции *tellg* и *tellp*, возвращающие текущее положение указателей *get* и *put* соответственно

```
streampos tellg();
streampos tellp();
```

Рассмотрим еще одну программу, использующую функции *seekg*, *seekp* и *tellg*, *tellp* выполняющую перезапись двух упорядоченных файлов (один по возрастанию, другой по убыванию) в один, например по возрастанию.

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
using namespace std;

void error(char *s1, char *s2 = "")  // вывод сообщения об ошибке
{
  cout << s1 << "  " << s2 << endl;  // при открытии потока для файла
  exit(1);
}

int main()
{
  int i1, i2, n1;
  ifstream f1, f2;                  // входной поток
  ofstream f3;                     // выходные потоки
  streampos n2;
  f1.open("f1.txt", ios::in);       // открытие исходного файла
  if (!f1)                          // проверка состояния потока
  error("ошибка открытия файла", "f1.txt")
  f2.open("f2.txt", ios::in);     // открытие 1 выходного файла
```

```
  if (!f2) error("ошибка открытия файла", "f2.txt");
  f3.open("f3.txt", ios::out);     // открытие 2 выходного файла
  if (!f3) error("ошибка открытия файла", "f3.txt");
  f1 >> i1;
  n1 = f1.rdstate();
  f2.seekg(-3, ios::end);
  n2 = f2.tellg();
  f2 >> i2;
  while (!n1 && n2 >= 0)
  {
    while (i1 < i2 && !n1)  // считывание в буфер до 20 символов
    {
      f3 << i1 << ' ';
      f1 >> i1;
      n1 = f1.rdstate();
    }
    while (i1 >= i2 && n2 >= 0)  // считывание в буфер до 20 символов
    {
      f3 << i2 << ' ';
      n2 -= 3;
      f2.seekg(n2);
      f2 >> i2;
    }
  }
  while (!n1)  // считывание в буфер до 20 символов
  {
    f3 << i1 << ' ';
    f1 >> i1;
    n1 = f1.rdstate();
  }
  while (n2 >= 0)  // считывание в буфер до 20 символов
  {
    f3 << i2 << ' ';
    n2 -= 3;
    f2.seekg(n2);
    f2 >> i2;
    n2 = f2.rdstate()
  }
  f1.close();                              // закрытие файлов
  f2.close();
  f3.close();
}
```

Файлами произвольного доступа

Организация хранения информации в файле прямого доступа предполагает доступ к ней не последовательно от начала файла по некоторому ключу, а непосредственно, например, по их порядковому номеру. Для этого требуется, чтобы все записи в файле были одинаковой длины.

Наиболее удобными для организации произвольного доступа при вводе–выводе информации являются компоненты - функции *istream::read* и *ostream::write*. При этом, так как функция *write(read)* ожидает первый аргумент типа *const char* (char*)*, то для требуемого приведения типов используется оператор явного преобразования типов:

```
istream & istream::read(reinterpret_cast<char *>(&s), streamsize n);
ostream& ostream::write(reinterpret_cast<const char *>(&s), streamsize n);
```

Ниже приведены тексты нескольких программ организации работы с файлом произвольного доступа. Вначале рассмотрим несложный пример программы выполняющей объединение двух отсортированных файлов(один по возрастанию, второй по убыванию) в третий по возрастанию.

```
#include <fstream>
#include <iostream>
using namespace std;

void error(char *s1, char *s2 = "")  // вывод сообщения об ошибке
{
  cout << s1 << "  " << s2 << endl;  // при открытии потока для файла
  exit(1);
}

int main()
{
  int i1, i2, n1, n2;
  fstream f1, f2;                  // входные потоки
  fstream f3;                      // выходной поток
                                   //streampos n2;
  f1.open("f1.txt", ios::out);       // открытие исходного файла
  if (!f1)                           // проверка состояния потока
  error("ошибка открытия файла", "f1.txt");
  f2.open("f2.txt", ios::out);      // открытие 1 выходного файла
  if (!f2) error("ошибка открытия файла", "f2.txt");
  while (1)
  {
    cin >> n1;
    if (n1 == 100) break;
    f1.write(reinterpret_cast<char *>(&n1), sizeof(int));
    cout << f1.rdstate() << endl;
  }
  while (1)
  {
    cin >> n1;
    if (n1 == 100) break;
    f2.write(reinterpret_cast<char *>(&n1), sizeof(int));
    cout << f2.rdstate() << endl;
  }
  f1.close();
  f2.close();
  f1.open("f1.txt", ios::in);     // открытие 1  файла
  f2.open("f2.txt", ios::in);     // открытие 2  файла
                                  // открытие выходного файла
  f3.open("f3.txt", ios::in | ios::out | ios::trunc);
  if (!f3) error("ошибка открытия файла", "f3.txt");
  f1.read(reinterpret_cast<char *>(&i1), sizeof(int));
  n1 = f1.rdstate();
  f2.seekg(-sizeof(int), ios::end);
  n2 = f2.tellg();
  f2.read(reinterpret_cast<char *>(&i2), sizeof(int));
  while (!n1 && n2 >= 0)
  {
    while (i1 < i2 && !n1)  // перезапись из первого файла
    {
      f3.write(reinterpret_cast<char *>(&i1), sizeof(int));
      f1.read(reinterpret_cast<char *>(&i1), sizeof(int));
      n1 = f1.rdstate();
```

```
    }
    while (i1 >= i2 && n2 >= 0)  // перезапись из ворого файла
    {
      f3.write(reinterpret_cast<char *>(&i2), sizeof(int));
      n2 -= sizeof(int);
      f2.seekp(n2);
      f2.read(reinterpret_cast<char *>(&i2), sizeof(int));
    }
  }
  while (!n1)  // считывание в буфер до 20 символов
  {
    f3.write(reinterpret_cast<char *>(&i1), sizeof(int));
    f1.read(reinterpret_cast<char *>(&i1), sizeof(int));
    n1 = f1.rdstate();
  }
  while (n2 >= 0)  // считывание в буфер до 20 символов
  {
    f3.write(reinterpret_cast<char *>(&i2), sizeof(int));
    n2 -= sizeof(int);
    f2.seekp(n2);
    f2.read(reinterpret_cast<char *>(&i2), sizeof(int));
  }
  f3.seekg(0);
  while (1)
  {
    f3.read(reinterpret_cast<char *>(&i1), sizeof(int));
    if (f3.rdstate()) break;
    cout << i1 << ' ';
  }
  cout << endl;
  f1.close();                              // закрытие файлов
  f2.close();
  f3.close();
}
```

Далее рассмотрим пример удаления и добавления в файл информации о банковских реквизитах клиента (структура inf).

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

struct inf
{
  char cl[10];    // клиент
  int pk;             // пин-код
  double sm;      // сумма на счете
} cldata;

class File
{
  char filename[80];       // имя файла
  fstream *fstr;                // указатель на поток
  int maxpos;                  // число записей в файле
public:
  File(char *filename);
  ~File();
  int Open();                     // открытие нового файла
  const char* GetName();
  int Read(inf &);              // считывание записи из файла
  void Remote();               // переход в начало файла
```

```
  void Add(inf);                // добавление записи в файл
  int Del(int pos);              // удаление записи из файла
};

ostream& operator << (ostream &out, File &obj)
{
  inf p;
  out << "File " << obj.GetName() << endl;
  obj.Remote();
  while (obj.Read(p))
  out << "\nКлиент -> " << p.cl << ' ' << p.pk << ' ' << p.sm;
  return out;
}

File::File(char *_filename)              // конструктор
{
  strncpy(filename, _filename, 80);
  fstr = new fstream();
}

File::~File()                                    // деструктор
{
  fstr->close();
  delete fstr;
}

int File::Open()         // функция открывает новый файл для ввода-вывода
{
fstr->open(filename, ios::in | ios::out | ios::binary | ios::trunc); //
бинарный
  if (!fstr->is_open()) return -1;
  return 0;
}

int File::Read(inf &p)     // функция чтения из потока fstr в объект   p
{
  if (!fstr->eof() &&                // если не конец файла
    fstr->read(reinterpret_cast<char*>(&p), sizeof(inf)))
  return 1;                                //
  fstr->clear();
  return 0;
}

void File::Remote()      // сдвиг указателей get и put в начало потока
{
  fstr->seekg(0, ios_base::beg); // сдвиг указателя на get на начало
  fstr->seekp(0, ios_base::beg); // сдвиг указателя на put на начало
  fstr->clear();                          // чистка состояния потока
}

const char* File::GetName()      // функция возвращает имя файла
{
  return this->filename;
}

void File::Add(inf cldata)
{
  fstr->seekp(0, ios_base::end);
  fstr->write(reinterpret_cast<char*>(&cldata), sizeof(inf));
  fstr->flush();
}
```

```
int File::Del(int pos) // функция удаления из потока записи с номером pos
{
  Remote();
  fstr->seekp(0, ios_base::end);  // для вычисления maxpos
  maxpos = fstr->tellp();            // позиция указателя put
  maxpos /= sizeof(inf);
  if (maxpos < pos) return -1;
  fstr->seekg(pos * sizeof(inf), ios::beg); // сдвиг на позици
                        // следующую за pos
  while (pos < maxpos)
  {
    fstr->read(reinterpret_cast<char*>(&cldata), sizeof(inf));
    fstr->seekp(-2 * sizeof(inf), ios_base::cur);
    fstr->write(reinterpret_cast<char*>(&cldata), sizeof(inf));
    fstr->seekg(sizeof(inf), ios_base::cur);
    pos++;
  }
  strcpy(cldata.cl, "");  // для занесения пустой записи в
  cldata.pk = 0;           // конец файла
  cldata.sm = 0;
  fstr->seekp(-sizeof(inf), ios_base::end);
  fstr->write(reinterpret_cast<char*>(&cldata), sizeof(inf));
  fstr->flush();         // выгрузка выходного потока в файл
}
int main(void)
{
  int n;
  File myfile("file");
  if (myfile.Open() == -1)
  {
    cout << "Can't open the file\n";
    return -1;
  }
  cin >> cldata.cl >> cldata.pk >> cldata.sm;
  myfile.Add(cldata);
  cout << myfile << endl;      // просмотр файла
  cout << "Введите номер клиента для удаления ";
  cin >> n;
  if (myfile.Del(n) == -1)
  cout << "Клиент с номером " << n << " вне файла\n";
  cout << myfile << endl;      // просмотр файла
}
```

Вопросы:

- Организация работы с файлами.
- Организация работы с файлами последовательного доступа.
- Организация работы с файлами произвольного доступа.
- Организация ввода и вывода объектов

Задание

Задание выполнить на основе предыдущих лабораторных работ или показать функции из курсовой работы если реализован схожий функционал (текстовые/бинарные файлы, запись/чтение объектов в файл).

Расшить функционал предыдущих работ. Написать функции для чтения/записи в файл в двух видах: текстовом и бинарном. В контейнер (разработанный в лабораторной работе №4) поместить несколько объектов из лабораторной работы №3, записать содержимое контейнера в файл и прочитать записанные данные обратно в контейнер. Для записи в бинарный файл следует использовать промежуточную структуру с фиксированным размером полей, если в исходном классе содержатся динамические поля.

Продемонстрировать работу с текстовыми и бинарными файлами.

Литература

1. Бьерн Страуструп. Язык программирования C++. Пер. с англ.- М.: «Издательство БИНОМ», 2004. – 1098 с.

2. Скляров В.А. Язык C++ и объектно-ориентированное программирование. Мн.: Выш.шк., 1997г. – 478 с.: ил.

3. Дейтел Х., Дейтел П. Как программировать на C++. Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2001. – 1152 с.: ил.

4. Лафоре Р. Объектно-ориентированное программирование в C++.– «Питер», 2003. – 923 с.

5. Луцик Ю.А., Ковальчук А.М., Лукьянова И.В. Объектно-ориентированное программирование на языке C++. Мн.: БГУИР, 2003. – 203 с.: ил.

6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – «Питер», Addison-Wesley, 2009. – 366 с.

7. Фримен Э., Фримен Э., Сьерра К., Бейтс Б. Паттерны проектирования. – «Питер», 2011 г. – 656 с.