Языки программирования

Лекция 10

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2023

Работа с бинарными файлами в стиле C++

Бинарные (двоичные) файлы можно использовать
для организации внешних таблиц (массивов структур во внешней памяти).

Режимы открытия файлов

Чтобы сохранять данные в двоичном формате, при создании потокового объекта (или при открытии) необходимо указать режим **ios::binary**.

В отличие от текстового режима, этот режим должен быть задан явно.
При явном указании режима требуется определить все режимы открытия файла, соединив их поразрядной операцией **|** (или).

```
ifstream fin(nameF, ios::in | ios::binary);

ofstream fout(nameF, ios::app |ios::binary);

fstream finout(nameF, ios::in | ios::out | ios::binary);
```

Запись в файл

Для записи данных в двоичном формате используется метод **write()**

fout.write(reinterpret_cast <char*>(&p),sizeof p);

Этот метод копирует определенное число байтов из памяти в файл.

Количество байтов, которое должно быть скопировано, задается вторым параметром.

Первый параметр определяет адрес, где расположены данные, которые необходимо скопировать.

Особенностью метода является то, что адрес должен быть преобразован к типу «указатель на char».

Чтение из файла

Для чтения из двоичного файла используется метод **read()**, имеющий такой же список параметров, как и метод write()

fin.read(reinterpret_cast <char*>(&p),sizeof p);

Данный метод возвращает
значение true, если операция чтения завершилась нормально, и
false — в случае возникновения ошибки, например, при достижении конца файла.

Доступ к записи с определенным номером

Поскольку при организации внешней таблицы файл состоит из записей одинакового размера, легко обеспечить доступ к записи с определенным номером.

Для этого используются методы:
seekg() — с объектами классов ifstream и fstream, и
seekp() — с объектами классов ofstream и fstream

Для обоих методов существуют два варианта перегрузок с одним параметром и с двумя.
В первой версии функции позиция задается абсолютным значением,
во второй — через смещение от одной из позиций (начало, текущая, конец)

Пример. Написать программу для организации работы с таблицей, хранящей данные о студентах (имя, год рождения, средний балл по итогам последней сессии).

Для этого реализовать следующие функции: создание файла, вывод его содержимого на экран, выдача записи по номеру.

```
struct dates {
  char name[20];
  int year;
  double rate;
};


inline void end_of_line() {cin.ignore(1,'\n');}
```

Создание файла

```
void toFile(char* nameF) {
  dates p;
  ofstream fout(nameF, ios::app |ios::binary);
  if (!fout.is_open()) {
    cerr<<"Can't open "<<nameF<<"\n";
    exit(1);
  }
  cout<<"Enter name \n";
  cin.get(p.name,20);
  while (p.name[0]!='\0'){
    end_of_line();
    //inline void end_of_line()
    //{cin.ignore(1,'\n');}
    cout<<" Enter year ";
    cin>>p.year;
    cout<<"Enter rate ";
    cin>>p.rate;
    …
```

get() считывает символы в массив, на который ссылается указатель p.name, пока не будет считано 19 символов, или обнаружен символ перехода на следующую строку, или достигнут конец файла.
Эта функция записывает нулевой символ в конец массива, на который ссылается указатель p.name.
Символ перехода на новую строку не считывается (!). Он остается в потоке, пока не будет выполнена следующая операция ввода

Создание файла

```
    …
    fout.write(reinterpret_cast <char*>(&p),sizeof p);
    end_of_line(); //inline void end_of_line() {cin.ignore(1,'\n');}

    cout<<"Enter name (blank line to quit) \n";
    cin.get(p.name,20);
  }
  fout.close();
}
```

Вывод содержимого файла на экран

```
void echoFile(char* nameF) {
  dates p;
  ifstream fin(nameF, ios::in | ios::binary);
  if (fin.is_open()) {
    while (fin.read(reinterpret_cast <char*>(&p),sizeof p)) {
      cout<< setw(20)<<p.name<<" : "
          <<setw(10)<<p.year<<setprecision(2)
          <<setw(15)<<p.rate<<"\n";
    }
  }
  fin.close();
}
```

Выдача записи по номеру

```
void numbDates (char* nameF, int n) {
  dates p;
  ifstream fin(nameF, ios::in | ios::binary);
  streampos place = n * sizeof p;
  fin.seekg(place);
  if (fin.fail())
    exit(1);
  fin.read(reinterpret_cast <char*>(&p),sizeof p);
  cout<< setw(20)<<p.name<<" : "<<setw(10)<<p.year<<setprecision(2)<<setw(15)<<p.rate<<"\n";
  fin.close();
}
```

Ошибки состояния

```
if (file.bad())
   std::cout << "Ошибка ввода-вывода при чтении\n";
else if (file.eof())
   std::cout << "Достигнут конец файла\n";
else if (file.fail())
   std::cout << “Предполагаемая позиция за границей файла и т.п.\n";
```

Если мы достигли конца файла и хотим установить курсор на начало **file.seekg(0, file.beg),** то ничего не прочитаем,
т.к. флаг fail не сброшен.

Сбросить можно с помощью **file.clear()**.

Текущее состояние можно вывести **file.rdstate()**

Пример. Дан файл вещественных чисел. Обнулить элементы файла, значения которых меньше среднего арифметического всех чисел в файле.

```
int t=sizeof(double);

void toFile(char* nameF) {
  ofstream fout(nameF, ios::out | ios::binary);
  if (!fout.is_open()) {
    cerr << "Can't open " << nameF << "\n";
    exit(1);
  }
  double val;
  cout << "Enter value ";
  while (cin >> val){
    fout.write(reinterpret_cast <char*>(&val), t);
  }
  fout.close();
}
```

При вводе данных в неподходящем формате (ввод символа вместо ожидаемого числа) значением выражения **cin>>val** является **false**

```
void echoFile(char* nameF) {
  double val;
  ifstream fin(nameF, ios::in | ios::binary);
  if (fin.is_open()) {
    while (fin.read(reinterpret_cast <char*>(&val),t))
      cout << val << " ";
    cout << endl;
  }
  fin.close();
}
```

Следует обратить внимание на то что, функции **average** и **smooth** в качестве параметра получают потоковую переменную, передаваемую по ссылке

double average(fstream &f)
void smooth(fstream &f, double avg)

Потоковые переменные всегда передаются в функции по ссылке

```
double average(fstream &f) {
  double p,s=0;
  int k=0;
  streampos posg = f.tellg();
  f.seekg(0, ios_base::beg);
  f.read(reinterpret_cast <char*>(&p), t);
  while (!f.eof()){
    s += p;
    k++;
    f.read(reinterpret_cast <char*>(&p), t);
  }
  f.clear();
  f.seekg(posg, ios_base::beg);
  if (k) return s / k;
  return 0;
}
```

Позиция указателя является объектом класса **streampos**.
Определение позиции через смещение относительно начала файла:
f.seekg(0, ios_base::beg);

В функциях, в которых файловый поток является параметром, рекомендуется в начале функции сохранять позицию указателя, а в конце — ее восстанавливать.
Для запоминания позиции потокового файл типа fstream можно использовать одну из функций **tellg/tellp**, для установки позиции — **seekg/seekp**.

```
streampos posg = f.tellg();
...
f.seekg(posg, ios_base::beg);
```

```
void smooth(fstream &f, double avg){
  double p;
  streampos posg = f.tellg();
  streampos posp = f.tellp();
  f.seekg(0, ios_base::beg);
  streampos pos=f.tellg();
  f.read(reinterpret_cast <char*>(&p), t);
  while (!f.eof()){
    if (p < avg){
      f.seekp(pos, ios_base::beg);
      p = 0.0;
      f.write(reinterpret_cast <char*>(&p), t);
    }
    pos = f.tellg();
    f.seekg(pos, ios_base::beg);
    f.read(reinterpret_cast <char*>(&p), t);
  }
  f.clear();
  f.seekg(posg, ios_base::beg);
  f.seekp(posp, ios_base::beg);
}
```

Если предполагается использовать потоковый файл типа fstream и для выполнения операции чтения (read) и для выполнения операции записи (write), то перед каждой операцией чтения/записи следует явно устанавливать указатель файлового потока в нужную позицию.

Поскольку организация цикла использует проверку состояния файлового потока eof, то после завершения цикла необходимо очистить битовую маску состояния потока:
f.clear();

```
int main() {
  char nameF[30] = "inf.dat";
  toFile(nameF);
  echoFile(nameF);
  fstream finout(nameF, ios::in | ios::out | ios::binary);
  if (!finout.is_open()) {
    cerr << "Can't open " << nameF << "\n";
    exit(1);
  }
  double d= average(finout);
  smooth(finout, d);
  finout.close();
  echoFile(nameF);
  return 0;
}
```

Приведение типов данных

Приведение типа в стиле C

Приведение типов в стиле C++

Приведение типов часто становится источником всевозможных проблем.

В общем случае количество таких операций **должно быть минимальным**

Приведение типа в стиле С

```
int b = 200;
//long c = (long) b;
long c = b;
unsigned long a = (unsigned long) b;
```

Приведение типа в стиле C++

xxx_cast< type_to >(expression_from)

Например:

```
int b = 200;
unsigned long a = static_cast< unsigned long >( b );

string s = static_cast< string >("Hello!");

ofstream fout("nameF",ios::app |ios::binary);
fout.write(reinterpret_cast <char*>(&b),sizeof b);
```

Операторы приведения типов в стиле C++

Оператор	Описание
static_cast	Для «безопасного» и «более или менее безопасного» приведения, включая то, которое может быть выполнено неявно. Например, автоматическое приведение типа.
const_cast	Изменение статуса объявлений volatile и/или const.
reinterpret_cast	Приведение к типу с совершенно иной интерпретацией. Принципиальная особенность этого приведения заключается в том, что для надежного использования значение должно быть приведено обратно к исходному типу. Тип, к которому выполняется приведение, обычно задействуется для манипуляций с битами или других неочевидных целей. **Это самый опасный из всех видов приведения**.
dynamic_cast	Понижающее приведение, безопасное по отношению к типам.

Сравнение

Если неправильно используются операторы приведения в стиле C++, то компилятор сообщит об ошибке.
Приведение в стиле C не обладает такой возможностью.

O - нотация оценки сложности алгоритмов

Для оценивания трудоемкости алгоритмов была введена специальная система обозначений – так называемая O-нотация.

Эта нотация позволяет учитывать в функции f (n) лишь наиболее значимые элементы, отбрасывая второстепенные.

Например, в функции **$f(n) = 2n^2 + n - 5$** при достаточно больших n компонента **n^2** будет значительно превосходить остальные слагаемые, и поэтому характерное поведение этой функции определяется именно этой компонентой.

Остальные компоненты можно отбросить и условно записать, что данная функция имеет оценку поведения (в смысле скорости роста ее значений) вида **$O(n^2)$**.

Важность О-оценивания

Состоит в том, что оно позволяет описывать **характер поведения** функции f(n) с ростом n:
насколько быстро или медленно растет эта **функция**.

О-оценка позволяет разбить все основные функции на ряд групп в зависимости от скорости их роста

Группы функций

1. **постоянные** функции типа **O(1)**, которые с ростом n НЕ растут (в оценивании алгоритмов этот случай встречается крайне редко, но все-таки встречается!)
2. функции с **логарифмической** скоростью роста **$O(\log_2 n)$**
3. функции с **линейной** скоростью роста **O(n)**
4. функции с **линейно–логарифмической** скоростью роста **$O(n*\log_2 n)$**
5. функции с **квадратичной** скоростью роста **$O(n^2)$**
6. функции со **степенной** скоростью роста **$O(n^a)$** при a>2
7. функции с **показательной** или **экспоненциальной** скоростью роста **$O(2^n)$**
8. функции с **факториальной** степенью роста **O(n!)**

Время выполнения алгоритма

размер сложность	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00005 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 минут	5,2 минут	13 минут
2^n	0,0001 сек.	1 сек.	17,9 минут	12,7 дней	35,7 веков	366 веков
3^n	0,059 сек.	58 минут	6,5 лет	3855 веков	2×10^8 веков	$1{,}3 \times 10^{13}$ веков

Время выполнения алгоритма с определённой сложностью в зависимости от размера входных данных при скорости 10^6 операций в секунду