Лекция 5 Особенности потокового ввода-вывода. Файлы.

О.С. Трушин
Зав. лаб. ЯФ ФТИАН РАН,
Доцент кафедры нанотехнологии в электронике

План

- Потоки
- Классы и объекты потоков
- Ввод символов и строк с помощью функций-элементов
- Манипуляторы потоков
- Работа с файлами
- Файлы последовательного доступа
- Файлы прямого доступа

Потоки

В языке C++ производится ввод-вывод потоков байтов.

Поток – последовательность байтов.

В операциях **ввода** байты пересылаются **от устройств** (клавиатура, дисовод и т.д.) **в оперативную память**

При **выводе** байты пересылаются **из оперативной памяти на устройства** (экран дисплея, принтер, дисковод)

Язык C++ предоставляет возможности для ввода-вывода:

1) на низком уровне (неформатированный ввод-вывод)
2) на высоком уровне (форматированный ввод-вывод)

Библиотека IOSTREAM

Представляет множество возможностей для выполнения операций ВВОДА-ВЫВОДА

Интерфейс разбит на несколько заголовочных файлов:

<iostream.h> - стандартный ввод с клавиатуры и вывод на экран

<iomanip.h> - форматированный ввод-вывод с использованием манипуляторов потока

<fstream.h> - операции ввода-вывода с использованием файлов

<strstream.h> - ввод-вывод в ПАМЯТЬ (операции с символьными массивами)

Классы и объекты потоков ввода-вывода

<< - операция поместить в поток

>> - операция взять из потока

cin – объект стандартного потока ввода «присоединен» к клавиатуре

Пример: **cin >> x;**

cout – объект стандартного потока вывода «присоединен» к дисплею

Пример: **cout << x;**

Сцепление операций поместить в поток и взять из потока

cout << “ x=“ << x << endl;

cin >> x >> y;

Выполнение операций осуществляется

слева направо !!!

Вывод переменных типа char*

Пример:

```
#include <iostream>

void main()
{

char* stroka=“Проверка”;

cout  <<  “ Значение строки равно “ << stroka << endl
        << “ Значение адреса строки равно ” << (void*) stroka << endl;

system(“pause”);
}
```

Вывод символов с помощью функции-элемента put

Вместо операций поместить в поток **(<<)** можно использовать функции-элементы

cout.put(‘A’);

Возможно сцепление операций:

cout.put(‘A’).put(‘B’);

Ввод символов с помощью функции-элемента get

Вместо операций взять из потока **(>>)** можно использовать функции-элементы

c=cin.get();

Пример:

```
#include <iostream>

void main()
{

   char c=cin.get();

   cout.put(c);

}
```

Ввод строки с помощью функции-элемента getline

Вместо операций взять из потока **(>>)** можно использовать функции-элементы

Пример:

```
#include <iostream>

void main(){

int size=50;

char stroka[size];

cin.getline(stroka,size);

}
```

cin.getline(stroka, size);

cin.getline(stroka, size, ‘\0’);

Манипуляторы потоков

Для форматирования ввода-вывода в C++
можно использовать манипуляторы потоков:

#include <iomanip.h>

- Задание ширины полей
- Задание точности
- Установка и сброс флагов формата
- Задание заполняющего символа полей
- Сброс потоков
- Вставка в выходной поток символа новой строки
- Вставка нулевого символа
- Пропуск символов разделителей во входном потоке

Манипуляторы для задания основания чисел dec,oct,hex

Пример:

```
#include <iostream>

void main()
{

int x=20;

cout   <<  “n=“ << n << endl
       << hex   << n << endl
       << oct    << n << endl;

}
```

Ответ:

```
n= 20
   14
   24
```

Манипуляторы для задания точности представления чисел

В C++ возможно управлять точностью представления чисел с помощью манипулятора потока **setprecision** или функции-элемента **precision**

Они задают число значащих цифр

Пример:

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>

void main(){
double root=sqrt(2.0);

cout.precision(2);
cout << root;

cout << setprecision(3) << root;
}
```

Ответ:

1.4

1.41

Манипуляторы для задания ширины поля ввода-вывода

Пример:

```
#include <iostream>

void main(){
int w=4;
char stroka[10];

cout << " Введите строку" << endl;
cin.width(5);

while( cin >> stroka) {
cout.width(w++);
cout << stroka << endl;
cin.width(5);
}

}
```

Функция-элемент **width()**
или
Манипулятор **setw()**

Работа с файлами

Файлы предназначены для постоянного хранения больших объемов данных

Компьютеры сохраняют файлы на вспомогательных запоминающих устройствах: магнитных дисках, оптических дисках, флэшках и т.д.

Файлы – представляют собой группы связных записей.

Существует множество способов организации записей в файле.

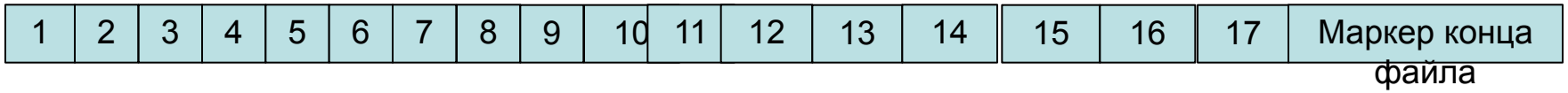
Наиболее распространенный тип – последовательный файл.

Различают **файлы последовательного доступа и файлы произвольного доступа**

Группы связанных файлов называют **базой данных**

Файлы и потоки

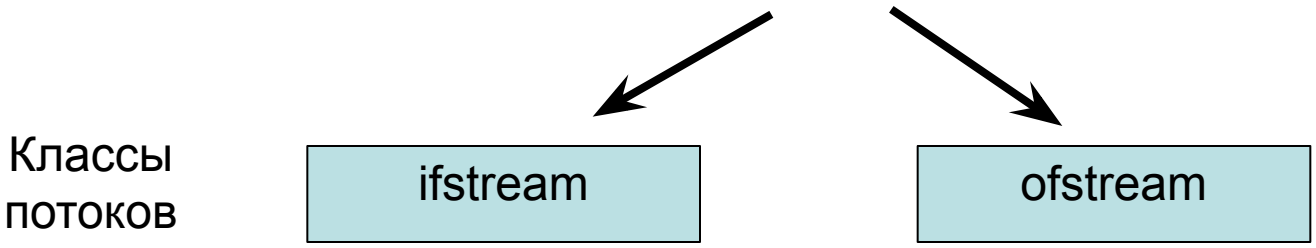
В C++ каждый файл рассматривается как последовательный поток байтов.



Маркер конца файла – EOF (End Of File marker)

Когда файл открывается , то создается объект и с этим объектом связывается поток

Для обработки файлов в C++ необходимо подключать заголовочный файл **fstream.h**



Все операции, функции-элементы и манипуляторы описанные для потоков работают и при обработке файлов

Создание файла последовательного доступа

Пример:

```
#include <iostream>
#include <fstream>

void main(){

ofstream outfile(“clients.dat”);

int account;  char name[10]; float balance;
while(cin >> account >> name >> balance)
{
  outfile << account <<“  “<<name<<“  “<<balance << endl;
}

}
```

В результате в файле создается последовательность записей, отсортированных по номеру счета

Режимы записи в файл. Функция-элемент open.

ofstream outfile(“ clients.dat”, ios::out);
Если файл существует , то стирает предыдущее содержимое

ofstream outfile(“ clients.dat”, ios::app);
Если файл существует, то дописывает в конец

Создание объекта без открытия файла
ofstream outfile;

Последующее открытие файла с помощью функции-элемента
outfile.open(“ clients.dat”, ios::out);

Чтение данных из файла последовательного доступа

Пример:

```
#include <iostream>
#include <fstream>

void main(){

ifstream infile(“clients.dat”, ios::in);

int account;  char name[10]; float balance;
while(infile >> account >> name >> balance)
{
  cout << account <<“  “<<name<<“  “<<balance << endl;
}

}
```

Файлы произвольного доступа

Для обновления файла последовательного доступа приходится перезаписывать весь файл !!!

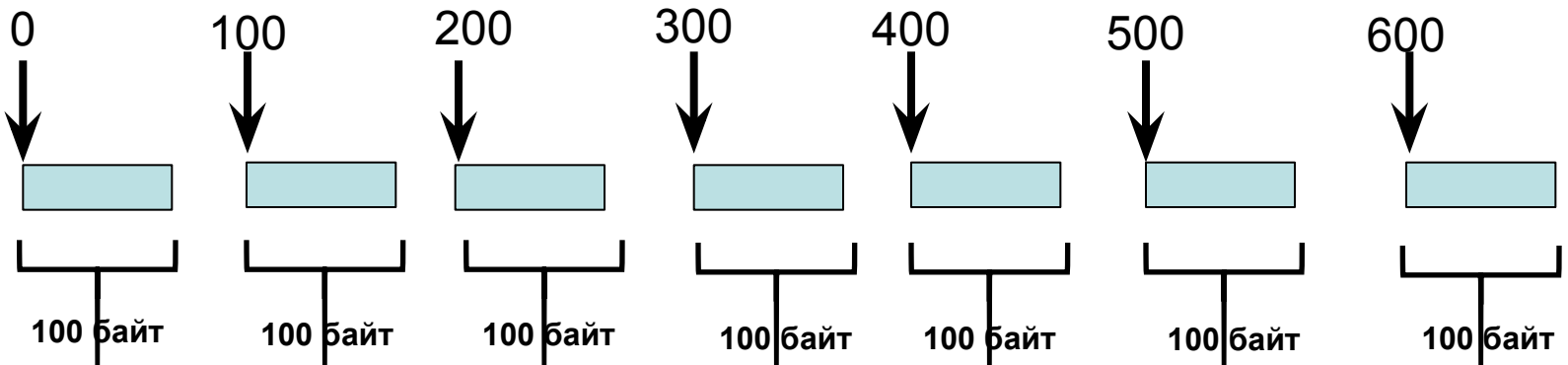
Для быстрой перезаписи отдельных записей лучше использовать файл произвольного доступа !!!

Пример: обновление состояния счета в банке отдельного лица.
В банке имеются тысячи счетов, но править нужно только один!!!

Отдельные записи файла произвольного доступа могут быть доступны непосредственно (и быстро) без поиска среди других записей.

При создании файла произвольного доступа накладывается требование чтобы все записи были **одинаковой фиксированной длины**

Структура файла произвольного доступа



Файл произвольного доступа подобен железнодорожному поезду со многими вагонами, одни из которых заняты, а другие свободны.

Данные могут быть вставлены в файл прямого доступа без разрушения других данных файла.
Данные, которые уже в нем хранятся могут быть изменены или удалены без перезаписи всего файла.

Создание файла прямого доступа

Пример:

```
#include <fstream>

//------------STRUCTURE----CLIENTDATA-------------
Struct clientData {
int accNum;
char lastName[15];
Char firstName[10];
float balance;
};
//-------------------------------------------------------------
void main(){
ofstream  outCredit(“credit.dat”, ios::out);

clientData blankClient = {0, “ “, “ “, 0.0);

for(int i=1; i<=100; i++)
outCredit.write( (char*)&blankClient, sizeof(blankClient));
}
```

Запись данных в файл прямого доступа

Пример:

```
#include <fstream>

//-----------STRUCTURE----CLIENTDATA-------------
Struct clientData {
int accNum;
char lastName[15];
Char firstName[10];
float balance;
};
//-------------------------------------------------------------
void main(){
ofstream  outCredit(“credit.dat”, ios::ate);

clientData client;
cin >> client.accNum >> client.lastName >> client.firtsName >> client.balance;
outCredit.seekp((client.accNum-1)*sizeof(client));
outCredit.write( (char*)&blankClient, sizeof(blankClient));
}
```

Чтение из файла прямого доступа

Пример:

```
#include <fstream>

//------------STRUCTURE----CLIENTDATA-------------
Struct clientData {
int accNum;
char lastName[15];
Char firstName[10];
float balance;
};
//-------------------------------------------------------------
void main(){
ifstream  inCredit(“credit.dat”, ios::in);

clientData client;

inCredit.read((char*)&client,sizeof(clientData));

cout << client.accNum << client.lastName;
}
```