# **19. Обработка потоков.** Потоки являются абстрактным понятием, введенным в языке C++ для однообразного описания и организации ввода-вывода. С помощью понятия потока можно как организовать ввод-вывод различных типов данных, так и обеспечить работу с различными внешними устройствами.

# Реализация механизма потоков выполнена средствами ООП, а потоковые классы и связанные с ними операции и функции представляют собой часть *стандартной библиотеки классов.*   Чтение данных из потока называется ***извлечением***, а их запись – ***помещением*** в поток. Для увеличения скорости передачи данных используется **механизм буферизации:** данные помещаются или извлекаются в специальную область памяти – буфер, а физический обмен данными происходит по заполнению/опустошению буфера

# Потоки можно классифицировать по следующим критериям: по направлению обмена данными – **на входные, выходные, двунаправленные;** по виду внешних устройств – на **стандартные, файловые, строковые.**

# ***стандартные*** потоки предназначены для передачи данных от клавиатуры и на экран дисплея; ***файловые***– для обмена информацией с файлами на внешних носителях; ***строковые*** потоки предназначены для работы с данными, хранящимися в оперативной памяти. Работа со строковыми потоками будет рассмотрена позднее.

# Ввод-вывод данных с использованием потоков может быть ***форматированным*** или ***неформатированным***. В первом случае используются перегруженные операторы >> и <<, которые позволяют помещать в поток и извлекать из потока данные любых типов, в том числе и пользовательских. Во втором случае (неформатированный ввод-вывод) никакого преобразования данных не происходит, и работа с потоками во многом напоминает уже описанную ранее работу с файлами.

# **18.1 Стандартная библиотека ввода-вывода** В С++ организована как иерархия шаблонов классов. Библиотека включает 2 стандартных множества конкретизирующих классов этой иерархии. **1.** для работы с символами типа char **2.** для работы с символами типа wchar\_t Классы для работы с wchar\_t символами имеют те же имена что и классы для работы с char, но с префиксом "W" . Схема иерархии классов.

# **ios** – базовый класс для всех потоков; **istream, ostream, iostream** – классы для стандартных потоков; **ifstream, ofstream, fstream** – классы для файловых потоков;

# Классы **ifstream** обеспечивает интерфейс для работы с файлами для ввода данных: **ifstream <имя входного потока>; ifstream fin;** Классы **ofstream** обеспечивает интерфейс для работы с файлами для вывода данных: **ofstream <имя входного потока> ; ofstream fout;**

# **18.2 Обработка потоков.** Для открытия потоков используют методы **       open** (<имя файла>**,<**режим открытия>) **        is\_open** () Для закрытия потоков **** метод **close**()

# void ifstream::open(const char \*name, ios::openmode mode = ios::in);  void ofstream::open  (const char \* name, ios::openmode mode = ios::out | ios::trunc);  void fstream::open    (const char \* name, ios::openmode mode = ios::in | ios::out); **name** - имя файла**, mode** - режим открытия файла. Параметр mode является перечислением и может принимать значения,

# **ifstream fin; ofstream fout; fin.open ("input.txt"); if (!fin.is\_open()) { cerr << "Open file failed." << endl; return ; } fout.open ("Output.txt"); if (!fout.is\_open()) { cerr << "Open file failed." << endl; return ; }**

# **fstream file; fin.open ("input.txt",ios::in); fout.open ("output.txt",ios::out | ios::trunc**); **file.open ("output.txt",ios::in | ios::out);** **fin.open ("input.txt",ios::in | ios::binary);**

# По умолчанию режим открытия файла соответствует типу файлового потока. У потока ввода или вывода флаг режима всегда установлен неявно. Например, для потока вывода в режиме добавления файла можно вместо оператора **ofstream fs("FName.txt", ios::out | ios::app); написать      ofstream fs("FName.txt", ios::app);**

# Между режимами открытия ios::ate и ios::app есть разница. Если файл открывается в режиме добавления, вывод в файл будет осуществляться в позицию, начинающуюся с текущего конца файла. В режиме открытия ios::ate (от анг "at end") можно изменить позицию вывода в файл и осуществлять запись, начиная с нее. Для потоков вывода режим открытия эквивалентен ios::out | ios::trunc, то есть можно опустить режим усечения файла. Для потоков ввода-вывода его нужно указывать явно. Файлы, которые открываются для вывода, создаются, если они еще не существуют.

# Если открытие файла завершилось неудачей, объект, соответствующий потоку, будет возвращать 0: **if (!fs){ cout << "Файл не открыт\n";}** Проверить успешность открытия файла можно также с помощью функции **is\_open(),** имеющей следующий прототип: **int is\_open() const;** Функция возвращает 1, если поток удалось связать с открытым файлом. Например, **if (!fs.is\_open()) { cout << "Файл не открыт\n"; return; }**

# Если при открытии файла не указан режим **ios::binary,** файл открывается в текстовом режиме. Для проверки, достигнут ли конец файла, можно использовать функцию **ios::eof(),** имеющую прототип **int eof(); while (!fs.eof()) {……….}** При завершении ввода-вывода, нужно закрыть файл, вызвав **void close():** **fs.close();** Закрытие файла происходит автоматически при выходе потокового объекта из области существования, когда вызывается деструктор потока.

# **18.3. Обработка состояния потока** В процессе работы с потоком могут возникнуть различные ситуации. Для этих целей в классе **ios** определено поле **state**, которое представляет собой комбинацию из следующих битовых масок: **eofbit -** достигнут конец файла **failbit -** ошибка при форматировании **hardfail -** неисправность оборудования **badbit -** прочая серьезная ошибка

# Для обработки состояния потока можно использовать следующие методы: **int eof() -** возвращает ненулевое значение, если установлен флаг **eofbit** **int good() -** возвращает ненулевое значение, если не установлены флаги состояния **void clear(int= 0) -** устанавливает новое значение состояния потока **int fail() -** возвращает ненулевое значение, если последняя операция завершилась неудачно. Возможно восстановление **int bad() -** возвращает ненулевое значение, если последняя операция завершилась неудачно и причины серьезные, восстановление не возможно)

# **Пример 18.1. void main() { double a; ofstream f2; ifstream f1; f1.open("input.txt"); if (!f1.is\_open()) { cerr << "Open file failed." << endl; return ; } f2.open("output.txt"); while (!f1.eof()) { f1>>a; f2<<setprecision(10)<<setw(20)<<a; f2<<setprecision(10)<<setw(20)<<sqrt(a); f2<<endl; } }**

# ***18.4. Форматированный ввод-вывод*** Для форматированного ввода-вывода используются перегруженные операторы >> и <<. Суть работы оператора << состоит в следующем: данные преобразуются в последовательность символов, которая и заносится в выходной файл. Целочисленные данные могут быть преобразованы к одной из систем счисления (10-й, 8-й или 16-ричной). Числа с дробной частью выводятся только в 10 с/с. Символы и строки выводятся в соответствии с принятой кодировкой (в консольных приложениях – это т.н. кодировка OEM, соответствующая кодовой странице 866). **cout << s1 << ’ ’ << s2;**

# Оператор >>, наоборот, анализирует последовательность данных во входном потоке и в случае правильной интерпретации преобразует данные во внутреннее представление того типа, который указан в качестве второго операнда. При этом разделителями между операндами является и пробел, и окончанием строки является не нажатие клавиши Enter, а первый встреченный пробел. Для решения этой проблемы можно использовать неформатированный ввод.

# Для управления вводом-выводом используют флаги и манипуляторы. Флаги можно устанавливать с помощью функции **setf(),** а сбрасывать - с помощью функции **unsetf().**

# ***18.4.1. Флаги форматирования* skipws** **-** при вводе пробельные литеры пропускаются; **left** **-** выводимые данные выравниваются по левому краю с дополнением символами-заполнителями по ширине поля; **right -** выводимые данные выравниваются по правому краю с дополнением символами-заполнителями по ширине поля (установлен по умолчанию); **internal -** знак числа выводится по левому краю, а само число – по правому краю. Промежуток между знаком и цифрами заполняется символом **x\_fill;**

# **dec -** целые числа выводятся по основанию 10 (установлен по умолчанию); устанавливается также манипулятором **dec;** **oct** **-** целые числа выводятся по основанию 8; устанавливается также манипулятором **oct;** **hex -** целые числа выводятся по основанию 16; устанавливается также манипулятором **hex** ; **boolalpha -** перевод логического 0 и 1 соответственно в false или **true; showbase -** при выводе целых чисел отображается префикс, указывающий на основание системы счисления;

# **showpoint -** при выводе чисел с плавающей запятой всегда отображается десятичная точка, а хвостовые нули не отбрасываются; **uppercase -** шестнадцатеричные цифры от А до F, а также символ экспоненты Е отображаются в верхнем регистре; **showpos -** при выводе положительных чисел отображается знак плюс;

# Функция **int width(int);** устанавливает значение ширины поля вывода (т.е. поля **x\_width**), Функция **int precision (int);** устанавливает количество дробных цифр – значение поля **x\_precision**. Функция **char fill(char);** задает значение символа заполнения.

# **// field width { cout << 100 << endl; cout.width(10); cout << 100 << endl; cout.fill('x'); cout.width(15); cout << left << 100 << endl; return 1; } 100 100 100xxxxxxxxxxxx**

# **// using fill character char prev; cout.width (10); cout << 40 << endl; //40 prev = cout.fill ('x'); cout.width (10); cout << 40 << endl; //xxxxxxxx40 cout.fill(prev); return 1;}**

# { int n = 70; cout.width (10); cout << internal << n << endl; 70 cout.width (10); cout << left << n << endl; 70 cout.width (10); cout << right << n << endl; 70 return 1; }

# ***18.4.2. Манипуляторы*** Работа с манипуляторами похожа на исполь-зование форматирующих методов, но более проста и понятна. Манипуляторы делятся на простые и параметризированные. **dec, oct, hex – системы счисления** **endl -** включает в поток при выводе символ новой строки и сбрасывает буфер **ends -** включает в поток при выводе нулевой символ **flush -** сбрасывает буфер

# **setprecision(int) -** устанавливает значение поля **x\_precision** **setw(int) -** устанавливает значение поля **x\_width** **setfill(char) -** устанавливает значение поля **x\_fill** **double x; cout << setprecision(3); cout << setw(9) << x;**

# ***18.5. Методы классов istream* и *ostream*** **get()** извлекает из потока символ и возвращает его код или **EOF**. **getc(ch)** извлекает из потока символ и помещает его в **ch**. Функция возвращает ссылку на текущий поток . **get(buf, num, lim='\n')** считывает **num-1** символов (или пока не встретит символ **lim**) в бу-фер, адрес которого содержится в параметре **buf** типа **char \*,** в конец дописывает нуль, символ **lim** не извлекается, возвращает ссылку на текущий поток .

# **Пример 18.2. Чтение** **символов потока и вывод** #include <iostream> #include <fstream> {char c, str[256]; ifstream is; cout << "Enter name file: "; cin.get (str,256); is.open (str); while (is.good()) { c = is.get(); cout << c; } is.close(); }

# **getline(buf, num, lim='\n')** считывает **num-1** символов (или пока не встретит символ **lim**) в бу-фер, адрес которого содержится в параметре **buf** типа **char \*,** в конец дописывает нуль, символ **lim** **извлекается** из потока**,** возвращает ссылку на текущий поток . **ignore(num=1, lim=EOF)** пропускает либо **num** символов из потока, либо символы до тех пор, пока не встретится символ-ограничитель **lim.**

# **Пример 18.3.** // Использование istream getline char name[256], title[256]; cout << "Enter your name: "; cin.getline (name,256); cout << "Enter your favourite movie: "; cin.getline (title,256); cout << name << "s favourite movie is " << title << endl;

# **peek()** возвращает текущий символ потока (или **EOF**) без его удаления из потока. **putback(ch)** помещает в поток символ **ch**. Этот символ становится текущим. **put(ch)** помещает в поток символ **ch** .

# **Пример 18.4. // Ввод числа или строки используем peek** { **char c; int n; char str[256]; cout << "Enter a number or a word: "; c = cin.peek(); if ( (c >= '0') && (c <= '9') ) { cin >> n; cout << "number " << n << endl; } else { cin >> str; cout << "word " << str << endl; } }**

# **Пример 18.5. // Ввод числа или строки используем putback** { char c; int n; char str[256]; cout << "Enter a number or a word: "; c = cin.get(); if ( (c >= '0') && (c <= '9') ) { cin.putback (c); cin >> n; cout << "number " << n << endl;} else { cin.putback (c); cin >> str; cout << " word " << str << endl; }}

# **flush()** принудительно сбрасывает буфера и выводит их содержимое на физическое устройство. **// Flushing files { ofstream outfile ("test.txt"); for (int n = 0; n < 100; n++) { outfile << n; outfile.flush(); } outfile.close(); }**

# **18.6. Неформатированный потоковый ввод-вывод** В потоковых классах наряду с операциями извлечения из потока и помещения в поток определены методы для неформатированного чтения-записи. Для потока могут быть определены две текущих позиции: одна используется для чтения, другая – для записи в поток.

# Когда файл открывается в текстовом режиме, происходит следующее: при вводе каждая пара символов '\r' +' \n' (возврат каретки + перевод строки) преобразуется в символ перевода строки ('\n'); при выводе каждый символ перевода строки ('\n') преобразуется в пару '\r' +'\n' (возврат каретки + перевод строки). Это не всегда удобно. Если необходимо использовать файл вывода для последующего ввода в программу (возможно, другую), лишние байты информации ни к чему.

# С этой целью система ввода-вывода предоставляет возможность осуществления неформатируемого ввода-вывода, то есть записи и чтения двоичной информации . Для осуществления ввода-вывода в двоичном режиме нужно включить флаг **ios::binary** в параметр **open\_mode** функции open(). Чтение двоичной информации из файла осуществляется функцией **read(),** которая имеет следующие прототипы

# **istream& read(char\* s, int n);** Параметр s задает буфер для считывания данных, а параметр n - число читаемых символов. **Fin\_bin.read (buffer, length);** Запись двоичных данных осуществляет функция write(): **ostream& write(const char \* s, int n); Fout\_bin.write (buffer, length);**

# **18.7. Файлы с произвольным доступом** Произвольный доступ в системе ввода-вывода реализуется с помощью функций **seekg**() и **seekp**(), используемых для позиционирования, соответственно, входного и выходного потока. Каждая из них имеет по два прототипа: **istream& seekg(long pos);      istream& seekg(long pos, seek\_dir dir);      ostream& seekp(long offset);      ostream& seekp(long offset, seek\_dir dir);**

# **Параметр pos** задает абсолютную позицию в файле относительно начала файла. Параметр **offset** задает смещение в файле, а **параметр dir** - направление смещения, которое может принимать значения в соответствии с определением из класса ios: **enum seek\_dir { beg, cur, end };** Константы перечисления определяют: **ios::beg - смещение от начала файла, ios::cur - смещение относительно текущей позиции,  ios::end - смещение от конца файла.**

# С каждым потоком связан указатель позиционирования, который изменяет свое значение в результате операции ввода или вывода. Для выполнения операций произвольного доступа файл должен открываться в двоичном режиме. Получить текущее значение позиции в потоке ввода или вывода можно с помощью функций **tellg() и tellp(),** соответственно. Эти функции имеют следующие прототипы: **long tellg (); long tellp();**

# **Пример 18.6. // Чтение файла в память read и вычисление размера #include <iostream> #include <fstream> using namespace std; int main () { int length; char \* buffer; ifstream is; is.open ("test.txt", ios::binary );** // определяем длину файла: **is.seekg (0, ios::end);**

# **length = is.tellg(); is.seekg (0, ios::beg);** // allocate memory: buffer = new char [length]; // read data as a block: is.read (buffer, length); is.close(); cout.write (buffer, length); }

# **Пример 18.7.** int pos; ofstream outfile; outfile.open("test.txt"); outfile.write("This is an apple", 16); pos = outfile.tellp(); outfile.seekp(pos - 7); outfile.write(" sam", 4); //"This is a sample", outfile.close();

# **// Пример 18.8. Копирование файла { char \* buffer; long size; ifstream infile("test.txt", ifstream::binary); ofstream outfile("new.txt", ofstream::binary); // get size of file infile.seekg(0, ifstream::end); size=infile.tellg(); infile.seekg(0);**

# buffer = new char [size]; // read content of infile infile.read (buffer, size); // write to outfile outfile.write (buffer, size); // release dynamically-allocated memory delete[] buffer; outfile.close(); infile.close(); return 1; }

# **Пример 18.9. Переписать все непустые строки из файла in1.txt, описанного в предыдущем примере, в выходной текстовый файл out1.txt void main() {char \*S = new char[101]; ifstream f1("in1.txt"); ofstream f2("out1.txt"); while (f1.getline(S,100)!= NULL) if (S[0] != '\0') { f2.write(S, strlen(S)); f2.put('\n');} f2.close();**

# **Пример 18.10.** В текстовом файле заданы слова с некоторым набором разделителей. Переписать из этого текстового файла в другой слова палиндромы, разделив их пробелом

# #include <iostream> #include <string> #include <fstream> using namespace std; int poiskPalindrom(string str)//строка палиндром { int i, k=1,n; // если палиндром k=1 нет k=-1 n=str.length(); for (i=0; i<n/2 ; i++) if(str[i]!=str[n-i-1]) k=-1; return k; }

# int main() { setlocale(LC\_ALL,"rus"); ifstream fin; ofstream fout; string rez\_st, slovo, seps(",:.!? "); fin.open("input.txt"); **if (!fin.is\_open())** { cout << "файл не открылся\n"; return 0; } **if (fin.eof())** { cout << "файл пустой\n"; return 0; } **string stt;**

# while(getline(fin,stt)) { if (stt!=""){ int i,k=0,count=0; stt+=" "; while (stt.length()!=0) { **while ((seps.find(stt[0])!=string::npos) &&(stt.length()!=0)) stt.erase(0,1);** //удаление разделителя i=0; while ( sepsg.find(stt[i++])==-1 && i<stt.length()); //поиск конца слова **slovo=stt.substr(0,stt.find(stt[i-1]));** //формирование слова

# //удаление слова из исходной строки stt.erase(0 ,i); if (poiskPalindrom(slovo)==1) // если слово палиндром { rez\_st+=slovo; rez\_st+=" "; //получим строку палиндромов } } if (rez\_st!="") cout << rez\_st<<endl; rez\_st.clear(); } }

# fin.close(); fout.close(); system("pause"); return 0; } **// для повторного чтения** fin.seekg(0, ios::beg); **или** fin.close(); fin.open("input.txt");

# **//нумерация файла –потока с повторным открытием** #include <iostream> #include <fstream> //#include <cstring> #include <string.h> using namespace std; #pragma warning(disable:4996) int main() { setlocale(LC\_ALL, "rus"); int n=0; ifstream fin; ofstream(fout); char st1[100]; string st; fin.open("input.txt"); if (!fin.is\_open()) { cout << "файл не открылся\n"; return 1; } fout.open("output.txt"); if (!fout) { cout << "файл не открылся" << endl; return 1; }

# while( ! fin.eof()) { fin.getline(st1,80); n++; fout <<n<<". "<< st1 << endl; st=st1; } int k=0; /\*fin.close(); fin.open("input.txt");\*/ fin.seekg(0, ios::beg);// для повторного чтения while( ! fin.eof()) { fin.getline(st1,80); n++; fout <<n<<". "<< st1 << endl; st=st1; } fin.close(); fout.close(); system("pause"); return 0; }