**Тема Ввод/вывод данных в C++**

**Понятие потока. Стандартные потоки ввода/вывода. Форматирование данных при вводе/выводе – флаги, функции, манипуляторы*.***

**Понятие потока (***stream***).** Функционал потоков ввода/вывода не определен как часть языка **C++**, а предоставляется стандартной библиотекой **C++** (и, следовательно, находится в [**пространстве имён std**](https://ravesli.com/urok-53-prostranstva-imen/)).

Потоки в **С++** отличаются от функций ввода/вывода в **C**, обеспечивая работу как со [стандартными потоками](javascript:termInfo(%22стандартными%20потоками%22)) данных, так и с типами данных, определяемыми пользователем, а также обеспечивая единообразный и понятный синтаксис. Обмен с потоком для увеличения скорости передачи данных производится через специальную область памяти – ***буфер***. Передача данных выполняется при выводе после заполнения буфера и при вводе, если буфер исчерпан.

**Поток** в **С++–** это *последовательность байтов*, независимых от конкретного устройства, с которого производится ввод/вывод данных. Поэтому практически ко всем типам устройств можно применить *одни и те же функции и операции ввода/вывода*. Поток связан с внешним устройством с помощью *системы* *ввода/вывода* **С++**.

По ***направлению обмена*** данных потоки делят на:

* [входные потоки](javascript:termInfo(%22входные%20потоки%22)) (позволяющие вводить данные в память);

* [выходные потоки](javascript:termInfo(%22выходные%20потоки%22)) (осуществляющие вывод данных из памяти);

* [двунаправленные потоки](javascript:termInfo(%22двунаправленные%20потоки%22)) (ввода/вывода данных).

В зависимости ***от вида устройства***, с которым работает поток, их делят на:

* стандартные потоки;
* файловые потоки;
* строковые потоки.

***Стандартные потоки*** – обеспечивают *консольный***в/ы** (события, возникающие от *клавиатуры* или на э*кране*). ***Файловые потоки*** – для обмена информацией с файлами. [***Строковые потоки***](javascript:termInfo(%22Строковые%20потоки%22)) – для работы с массивами символов в оперативной памяти.

**Различают два типа потоков –** *текстовый* и *двоичный* (бинарный)*.*

***Текстовые потоки*** – это *последовательность символов*. В текстовых потоках некоторые символы могут быть *преобразованы* согласно требованиям среды. Например, *символ новой строки* может быть преобразован в пару символов *«возврат каретки – перевод строки»*. Следовательно, может не быть однозначного соответствия между количеством прочитанных (записанных) байтов в памяти с количеством символов на внешнем устройстве.

Текстовые потоки обеспечивают ***форматированный ввод–вывод*** (*в/ы на высоком уровне*).

***Двоичные потоки*** – *последовательность байтов*, имеющих однозначное соответствие с байтами в памяти и на внешнем устройстве. Никакого преобразования символов не выполняется. Поэтому количество прочитанных (записанных) байтов в точности совпадает с количеством байтов на внешнем устройстве.

Двоичные потокиобеспечивают ***неформатированный ввод/вывод*** (*в/ы на низком уровне*). Они позволяет осуществить пересылку больших по объёму данных с высокой скоростью.

В **С++** при подключении заголовочного файла <**iostream**> программа получает доступ ко всей иерархии [**классов**](https://ravesli.com/urok-113-klassy-obekty-i-metody-klassov/) библиотеки **iostream**, в основе которой находится класс **ios**, содержащий множество *функций* и *переменных* *для управления основными* ***операциями*** *над потоками*.

Класс **ios** является базовым по отношению к классам **istream, ostream** и **iostream,** которыепредставляют собой **классы потоков*****ввода, вывода и ввода–вывода***соответственно (*потоковые классы*).

В языке **C++** ***поток*** представляет собой ***объект*** некоторого потокового класса.

**Стандартные потоки ввода/вывода.** По умолчанию стандартная библиотека **C++** предоставляет объект **cout** типа **ostream,** который позволяет выводить данные на консоль. Для чтения с консоли применяется объект **cin** типа **istream,** которыйпозволяет вводить данные с консоли.

Для записи данных в поток **ostream** применяется операция **<<** – **вставки в поток,** которая получает два операнда. Левый операнд представляет объект типа **ostream**, в который будет производиться вывод, а правый операнд – объект, данные которого выводятся в **поток**.

Для чтения данных из потока применяется операция **>>** – **извлечения из потока**, которая принимает два операнда. Левый операнд представляет объект типа **istream**, с которого производится считывание, а правый операнд – объект, в который считываются данные. Например:

int a;

cin>>a; // ввод с клавиатуры

cout<<"a = "<<a<<endl; // вывод на экран

Операции ***извлечения***из потокаи***вставки*** в поток в качестве результата своего выполнения формируют соответственно ***ссылки*** на объект типа **istream** или **ostream**, что позволяет создавать *цепочки операций*:

int a, b, c;

cin>>a>>b>>c; // **5 6 7 или 5 6 7**

cout<<a<<b<<c<<endl;

Данные *при вводе* из потока должны разделяться ***пробелами*** (или другими пробельными символами). Ввод прекращается, если очередной символ оказался недопустимым.

Символы *пробел, табуляция, перевод строки, возврат каретки, новая страница, вертикальная табуляция* и *новая строка* называются **пробельными.** Компилятор языка С++ игнорирует пробельные символы, если они используются не как компоненты символьных констант или символьных строк.

int **main(){**

char ch;

while (cin >> ch) cout << ch;

}

*Числовые значения* можно вводить в **10**-ой или **16**-ой системе счисления (с префиксом **0х**), со знаком или без знака:

int n;

cout<<"Enter number > 0 – **10** c/c: "; cin>>n; // вводим **26**

cout<<"Enter number > 0 – **16** c/c: ";

cin>>hex>>n; // вводим, например, **0X1A** или **1a**

Поскольку ввод буферизирован, есть возможность исправлять введённые символы до того, как нажата клавиша ***Enter***.

*Операции << и >>* перегружены для всех встроенных типов данных, строк и значений указателей, что позволяет автоматически выполнять ввод/вывод в соответствии с типом величин.

Как и для других ***перегруженных операций***, для *операций извлечения* и *вставки* невозможно определить приоритеты, поэтому в необходимых случаях используются скобки:

cout<<(i << j); // *операция* << в скобках означает сдвиг влево

Под любую величину *при выводе* отводится столько позиций, сколько требуется для её представления: cout<<i<<' '<<j<<" "<<k<<endl;

При *вводе строк* извлечение символов из потока происходит до ближайшего, например, пробела. Вместо него в строку заносится *нуль-символ* – '**\0**', который является признаком конца строки.

**Форматирование данных.** До сих пор при *вводе* или *выводе* информации в наших примерах программ действовали параметры форматирования, которые ***по умолчанию*** использует система ввода/вывода **С++**. И если они не устраивают программиста, то он может сам управлять форматом представления данных, причём разными способами.

В потоковых классах форматирование можно выполнить с помощью *флагов, специальных функций форматирования и манипуляторов*.

***Флаги форматирования***. В **С++** каждый поток связан с набором *флагов форматирования*, которые управляют процессом форматирования. Если флаг формата установлен, реализуется соответствующая ему *функция*.

Флаги устанавливаются функцией **setf(),** а сбрасываются функцией **unsetf()**. Функции являются членами класса **ios.** Вызов функций происходит относительно конкретного потока, поэтому каждый поток отдельно поддерживает своё собственное состояние формата. Функции могут устанавливать или сбрасывать сразу несколько флагов. При сбросе флага используется ***формат по умолчанию***:

поток.**setf**(ios::флаг1 | ios::флаг2 | ios::флаг3 …); //установка флагов

поток. **unsetf**(ios::флаг); // сброс флага

Наиболее часто используемые **флаги форматирования**:

**skipws** при вводе пробельные символы игнорируются;

**left** при выводе выравнивание по левому краю поля;

**rigth** при выводе выравнивание по правому краю поля;

**dec**  десятичная система счисления (по умолчанию);

**oct** восьмеричная система счисления;

**hex**  шестнадцатеричная система счисления;

**scientific** вывод вещественных чисел в форме мантиссы и порядка с

**6** знаками после точки;

**fixed** вывод вещественных чисел в обычной форме с **6**

десятичными знаками после точки;

**showpoint** вывод вещественных чисел с десятичной точкой и

дробной частью, по умолчанию всего **6** знаков;

Так как флаги форматирования являются определениями класса **ios**, обычно к ним обращаются через *операцию разрешения области видимости класса* **(::)** например, **ios::skipws**:

char str[20];

cout << "Enter stroku: "; cin.setf(ios::skipws);

cin >> str; // введём, например,111

cout<<"str = "<<str<<endl; //str = 111 (без начальных пробелов)

Большинство флагов форматирования могут быть установлены с помощью манипуляторов.

***Манипуляторы форматирования*.** В системе *ввода/вывода* языка **С++** имеется ещё один способ изменения *параметров форматирования*, связанных с потоком. Он реализуется с помощью ***специальных функций***, *называемых манипуляторами*, которые включаются в выражения **в/ы**.

Манипуляторы – это инструкции форматирования, которые вставляются прямо в поток. Например, **endl**, который посылает символ разделителя строк в поток **и сбрасывает** буфер:

cout << "\*\*\*\*\*\*" << endl;

При использовании манипуляторов, ***которые принимают аргументы***, необходимо включить в программу заголовочный файл **<iomanip>.**

Наиболее часто используемые **манипуляторы**:

**endl** при выводе перейти на новую строку;

**dec** вывод чисел в *десятичной* системе счисления

**oct** вывод чисел в *восьмеричной* системе счисления;

**hex**  вывод целых чисел в **16**-ой системе счисления;

**setw**(int w) задаёт ширину поля вывода, равную **w**;

**ws** пропуск начальных пробелов;

**left** при выводе выравнивание по левому краю поля;

**rigth** при выводе выравнивание по правому краю поля;

**setfill(**char ch**)** устанавливает символ заполнения **ch**;

**setprecision(**int p**)** задаёт число знаков после десятичной точки,

равное ***p*** позициям, если вывод в форме мантиссы

и порядка. Иначе – общее количество цифр.

Манипулятор **endl** осуществляет перевод на новую строку, но, в тоже время, производит **сброс буфера вывода** (а следовательно и всех форматов при использовании манипуляторов). Поэтому, там где нужно производить переход на новую строку и при этом сохранять форматы, нужно использовать **'\n'**.

Манипуляторы действуют только ***на те данные***, которые следуют ***за ними*** в потоке, а не на те, которые находятся перед ними.

***Функции форматирования****.* Кроме флагов форматирования в классе **ios** определены **функции-члены**, которые устанавливают параметры формата, например:

* **width(**int len) – устанавливает ширину (*width*) поля вывода;
* **precision(**int num**) –** устанавливает точность (*precision*) при выводе, т.е. число цифр после точки, если вывод в форме мантиссы и порядка, иначе – общее количество цифр;
* **fill(**char ch**) –** задаёт символ заполнения поля вывода.

int x = 1234567890; int y = 45;

cout.setf(ios::right);

cout.width(20); // **действует на 1 вывод**

cout.fill(**'\*'**); cout<<x;

cout.width(20); cout<<y<<endl;

**Результат:**

\*\*\*\*\*\*\*\*\*\*1234567890

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*45

**// Пример 1.** Форматирование данных.

#include <iomanip>

int **main()**{

cout<<123.23<<"ABCDEF"<<100<<endl; // установки по умолчанию

// меняем формат с помощью флагов форматирования

cout.unsetf(ios::dec); // требуется не для всех компиляторов

cout.setf(ios::hex|ios::scientific); // вывод в форме мантиссы и порядка

cout<<123.23<<"ABCDEF"<<100<<endl;

// использование функций формата

cout.width(10); // ширина поля **10**

cout<<"ABCDEF"<<endl; // выравнивание вправо

cout.width(10); cout.precision(4);

cout<<123.456789<<endl;

// использование манипуляторов

cout<<hex<<100<<" "; // вывод в **16**-ой с/с

cout<<oct<<10<<endl; // вывод в **8**-ой с/с

cout<<setw(10)<<left<<"ABCDEF"<<endl; // ширина поля **10**

cout<<setprecision(2)<<1234.5678<<endl;

}

**Результат:**

123.23ABCDEF100

1.232300e+002ABCDEF64

ABCDEF

1.2346e+002

64 12

ABCDEF

1.23e+003