Стандартные и нестандартные потоки ввода/вывода

2021

Потоки для форматированного ввода/вывода

Буферы потоков ввода/вывода

Двоичный ввод/вывод

Раздел 1

Потоки для форматированного ввода/вывода

Стандартные потоки

```
namespace std {
    class ios_base {
        streamsize _precision; // количество знаков после запятой
        streamsize _width; // ширина поля вывода
        fmtflags _flags; // флаги форматирования
        iostate _exception; // флаги состояния для исключений
        iostate _streambuf_state; // флаги состояния
    };
    template <class Ch, class Tr>
    class basic_ios: public ios_base {
        Ch _fill; // символ для заполнения пустоты
        basic_streambuf<Ch,Tr>* _streambuf; // буфер ввода/вывода
    };
    template <class Ch, class Tr>
    class basic_ostream: virtual public basic_ios<Ch,Tr> {
        // нет полей
    };
}
```

Стандартные потоки

Стандартная библиотека:

```
namespace std {
    extern istream cin;        // fd=0
    extern ostream cout;       // fd=1
    extern ostream cerr, clog; // fd=2
}
```

Командная строка (перенаправление из файлов):

```
./myprog 0< stdin.txt 1> stdout.txt 2> stderr.txt
./myprog  < stdin.txt  > stdout.txt 2> stderr.txt
```

Пример №1

```
// колонки фиксированной ширины
// ширина поля 20 символов, выравнивание по правому краю
std::cout.width(20);
std::cout.setf(std::ios::right, std::ios::adjustfield);
std::cout << "Word";
std::cout.width(20);
std::cout.setf(std::ios::right, std::ios::adjustfield);
std::cout << "Count";
...
```

```
                Word               Count
                 dog                   1
                 cat                  10
```

Пример №2

```
// генерация имени файла
unsigned int frame_number = 11;
std::stringstream filename;
filename << "frame-";
filename.width(10);
filename.fill('0');
filename << frame_number;
filename << ".jpg";
std::ofstream out{filename.str()}; // frame-0000000011.jpg
...
```

Пример №3: манипуляторы

```
std::cout
    << std::setw(20) << std::right << "Word"
    << std::setw(20) << std::right << "Count"
    << std::endl;
```

```
unsigned int frame_number = 11;
std::stringstream filename;
filename
    << "frame-"
    << std::setw(10) << std::setfill('0') << frame_number
    << ".jpg";
std::ofstream out{filename.str()}; // frame-0000000011.jpg
```

```cpp
// стандартные манипуляторы
namespace std {

    // упрощенная версия
    std::ostream&
    endl(std::ostream& os) {
        os.put('\n'); os.flush(); return os;
    }

    // оператор внутри класса ostream
    ostream&
    ostream::operator<<(ostream& (*manip)(ostream&)) {
        return manip(*this);
    }
}
```

```
namespace std {

    // вспомогательный класс
    struct Setw { int width; };

    // манипулятор с аргументом
    Setw setw(int n) { return {n}; }

    // оператор вывода для вспомогательного класса
    ostream& operator<<(ostream& os, Setw s) {
        os.width(s.width); return os;
    }

}
```

Пример №6: форматирование (C++20)

```
// колонки шириной 20 символов без выделения памяти
std::format_to(std::ostream_iterator<char>(std::cout, ""),
               "{:<20} {:<20}\n", "Word", "Count");

// колонки шириной 20 символов с выделением памяти
std::cout << std::format("{:<20} {:<20}\n", "Word", "Count");

// генерация имени файла
std::ofstream out{std::format("frame-{:0>10}.jpg", frame_number)};
```

Состояние потока

```
std::istream in(...);
if (in.rdstate() & std::ios::eofbit) { ... }
if (in.eof()) { ... }
```

eofbit	failbit	badbit	good()	fail()	bad()	eof()	bool
false	false	false	T	false	false	false	T
false	false	T	false	T	T	false	false
false	T	false	false	T	false	false	false
false	T	T	false	T	T	false	false
T	false	false	false	false	false	T	T
T	false	T	false	T	T	T	false
T	T	false	false	T	false	T	false
T	T	T	false	T	T	T	false

Пример: чтение массива

```
std::istream&
operator>>(std::istream& in, std::vector<float>& arr) {
    float x;
    // попытка №1
    while (in >> x) { arr.push_back(x); }

    // попытка №2
    while (!in.eof()) { in >> x; arr.push_back(x); }

    // попытка №3
    while (!in.eof()) {
        in >> x; if (!in) { break; }
        arr.push_back(x);
    }
    return in;
}
```

Пример: чтение массива

```
std::vector<float> arr;
std::ifstream in:
try {
    in.exceptions(std::ios::failbit | std::ios::badbit);
    in.open("arr.txt");
    in >> arr;
} catch (const std::ios::failure&) {
    std::cerr << "i/o error\n";
} catch (const std::exception& err) {
    std::cerr << err.what() << std::endl;
}
```


Что вы пишете?
программу
библиотеку
Включить исключения в каждом потоке
Ветка кода 1
Исключения включены?
да
нет
Ветка кода 1
Ветка кода 2

Раздел 2

Буферы потоков ввода/вывода

Перенаправление в файл

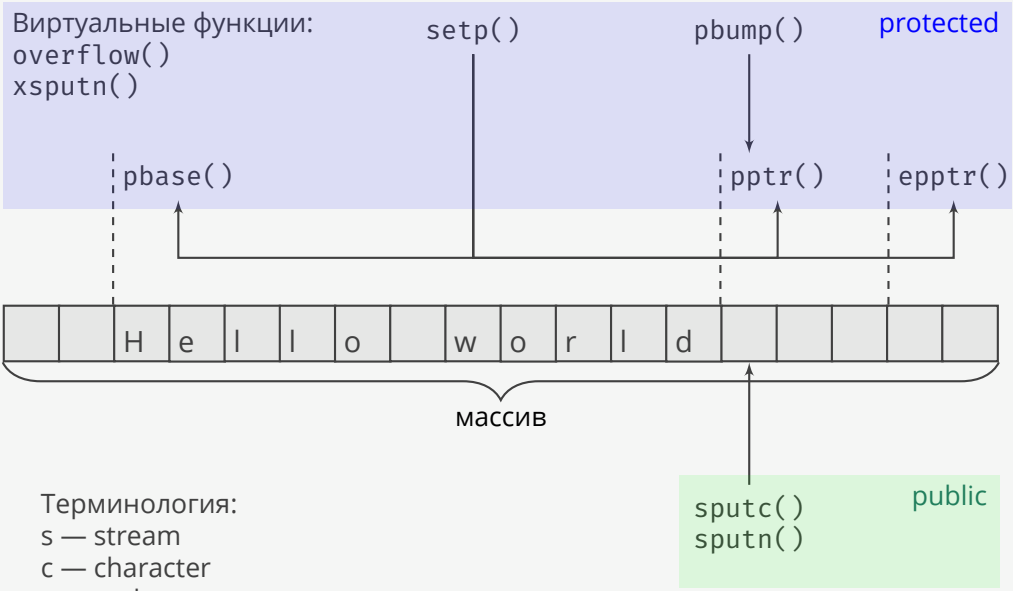
```
// перенаправление в файл
std::ofstream out{"out.txt"};
std::streambuf* oldbuf = std::cout.rdbuf(out.rdbuf());
...
std::cout.rdbuf(oldbuf); // восстанавливаем старый буфер

// копирование файлов
std::ofstream{"dest.txt"} << std::ifstream{"src.txt"}.rdbuf();
```

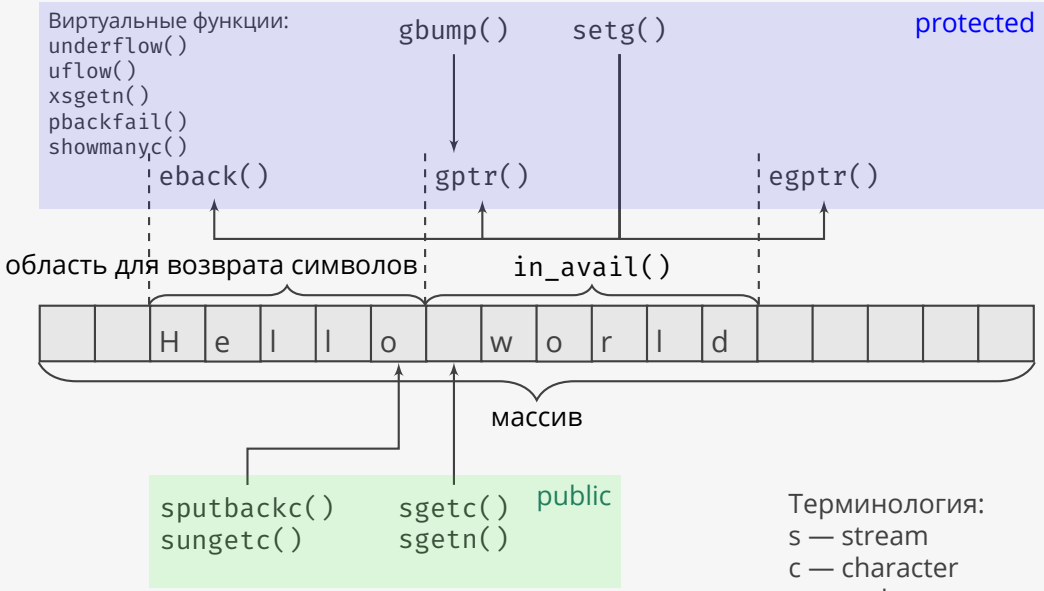
Базовый класс

```
template <class Ch, class Tr=std::char_traits<T>>
class basic_streambuf {
    Ch* _eback; // начало буфера для чтения
    Ch* _gptr;  // текущая позиция для чтения
    Ch* _egptr; // конец буфера для чтения
    Ch* _pbase; // начало буфера для записи
    Ch* _pptr;  // текущая позиция для записи
    Ch* _epptr; // конец буфера для записи
};

// p == put == output == write
// g == get == input == read
```



Терминология:
s — stream
c — character
n — n characters


Виртуальные функции:
underflow()
uflow()
xsgetn()
pbackfail()
showmanyc()
gbump()
setg()
protected
eback()
gptr()
egptr()
область для возврата символов
in_avail()
H e l l o w o r l d
массив
sputbackc()
sungetc()
sgetc()
sgetn()
public
Терминология:
s — stream
c — character
n — n characters

Виртуальные функции

Публичный метод	Какие защищенные методы он вызывает
`sgetc`	`*gptr()` `underflow`
`sgetn`	`xsgetn`
`sputc`	`*pptr()=c` `overflow`
`sputn`	`xsputn`
`in_avail`	`showmanyc`
`sputbackc`	`pbackfail`
`sungetc`	`pbackfail`
`pubseekoff`	`seekoff`
`pubseekpos`	`seekpos`
`pubsync`	`sync`

Пример: буфер-тройник

```
class teebuf: public std::streambuf {
    std::streambuf* buf1, *buf2;
public:
    teebuf(std::streambuf* b1, std::streambuf* b2):
    buf1(b1), buf2(b2) {}
protected:
    int_type overflow(int_type c) override {
        buf1->sputc(c); buf2->sputc(c); return c;
    }
    std::streamsize
    xsputn(const char* s, std::streamsize n) override {
        buf1->sputn(s, n); buf2->sputn(s, n); return n;
    }
};
```

```
// выводим одновременно на консоль и в файл
std::ofstream out{"out.txt"};
std::streambuf* oldbuf = std::cout.rdbuf();
teebuf tee{oldbuf, out.rdbuf()};
std::cout.rdbuf(&tee);
...
std::cout.rdbuf(oldbuf); // восстанавливаем старый буфер
```

Пример: буфер с нижним регистром

```
class LowerCaseBuf: public std::streambuf {
    std::streambuf* old; std::string str;
public:
    explicit LowerCaseBuf(std::streambuf* b): old(b) {}
protected:
    int_type overflow(int_type c) override {
        return old->sputc(std::tolower(c));
    }
    std::streamsize
    xsputn(const char* s, std::streamsize n) override {
        str.assign(s, n);
        for (char& ch : str) { ch = std::tolower(ch); }
        return old->sputn(str.data(), n);
    }
};
```

- Потоки — форматированный вывод (числа, строки, объекты).
- Буферы — посимвольный вывод (1,2,4-байтовые символы).
- Буферы — побайтовый вывод (1-байтовые символы).

Раздел 3

Двоичный ввод/вывод

Выравнивание типов

```
template <class T>
void print() {
    std::cout << sizeof(T) << ' ' << alignof(T) << '\n';
}

// для архитектуры x86_64 и armv6l
print<int>(); // 4 4
print<char>(); // 1 1

struct X { int i; char ch;    }; print<X>(); // 8 4
struct Y { int i; char ch[4]; }; print<Y>(); // 8 4
struct Z { int i; char ch[5]; }; print<Z>(); // 12 4
```

Общие правила выравнивания

- Размер структуры всегда делится без остатка на выравнивание: `sizeof(X)%alignof(X) == 0`.
- Для примитивных типов всегда `sizeof(X) == alignof(X)`.
- Адрес структуры всегда делится без остатка на выравнивание.
- Указатель `X*` можно безопасно привести к указателю `Y*`, только если `alignof(Y) <= alignof(X)`.

Частные правила выравнивания

- `x86_64`: выравнивание требуется только для операций с векторными регистрами.
- `C++`: память всегда выделяется с максимальным выравниванием (как правило, равному размеру векторного регистра).

Пример: контрольная сумма

```
// bytes — часть сетевого пакета
int checksum(const char* bytes, size_t n) {
    const int* x = reinterpret_cast<const int*>(bytes);
    const size_t m = n/sizeof(int);
    int sum = 0;
    for (int i=0; i<m; ++i) {
        sum += x[i];
    }
    return sum;
}
```

Пример: контрольная сумма

```
// bytes — часть сетевого пакета
int checksum(const char* bytes, size_t n) {
    const int* x = reinterpret_cast<const int*>(bytes); // ошибка
    const size_t m = n/sizeof(int);
    int sum = 0;
    for (int i=0; i<m; ++i) {
        sum += x[i];
    }
    return sum;
}
```

Пример: контрольная сумма

```
int checksum(const char* bytes, size_t n) {
    int sum = 0;
    for (size_t i=0; i<n; i+=sizeof(int)) {
        int y = 0;
        static_assert(alignof(char) <= alignof(int), "fail");
        auto ptr = reinterpret_cast<char*>(&y); // ок
        std::copy_n(bytes + i, sizeof(int), ptr);
        sum += y;
    }
    return sum;
}
```

Порядок байт

```
int number = 1;
auto bytes = reinterpret_cast<const char*>(&number);
for (int i=0; i<sizeof(int); ++i) {
    std::cout << int(bytes[i]);
}
std::cout << '\n';
// 1000 — от младшего к старшему (little-endian)
// 0001 — от старшего к младшему (big-endian)
```

- Little-endian: x86, ARM.
- Big-endian: PowerPC, SPARC.
- Динамический: PowerPC, ARM.

Сетевой порядок байт — от старшего к младшему (big-endian).

Изменение порядка байт

```
uint64_t byte_swap(uint64_t n) {
    return ((n & UINT64_C(0xff00000000000000)) >> 56) |
           ((n & UINT64_C(0x00ff000000000000)) >> 40) |
           ((n & UINT64_C(0x0000ff0000000000)) >> 24) |
           ((n & UINT64_C(0x000000ff00000000)) >>  8) |
           ((n & UINT64_C(0x00000000ff000000)) <<  8) |
           ((n & UINT64_C(0x0000000000ff0000)) << 24) |
           ((n & UINT64_C(0x000000000000ff00)) << 40) |
           ((n & UINT64_C(0x00000000000000ff)) << 56);
    // return __builtin_bswap64(n);
}
uint16_t byte_swap(uint16_t n) {
    return ((n & 0xff00) >> 8) | ((n & 0x00ff) << 8);
    // return __builtin_bswap16(n);
}
```

Правила преобразования

- Преобразование массива байт в примитивный тип безопасно только с помощью побайтового копирования.
- Преобразование из примитивного типа в массив байт безопасно даже через указатель (без копирования).
- При передаче данных по сети в двоичном виде порядок байт приводят к единому виду (либо всегда LE, либо всегда BE).
- Порядок байт не связан с порядком бит.
- Порядок байт — только для примитивных типов (int, short, long, float).

Ссылки

- Создание своих буферов для потоков C++.
  `http://wordaligned.org/articles/cpp-streambufs`
- Подробное объяснение невыровненного доступа к памяти от разработчиков ядра Linux.
  `https://www.kernel.org/doc/Documentation/unaligned-memory-access.txt`

Images:

- Quino Al «Teléfono» .