Ввод/вывод в C++

Звонарев Никита

СПбГУ, СтатМод

27 октября 2017 г.

Ввод-вывод в стиле C++?

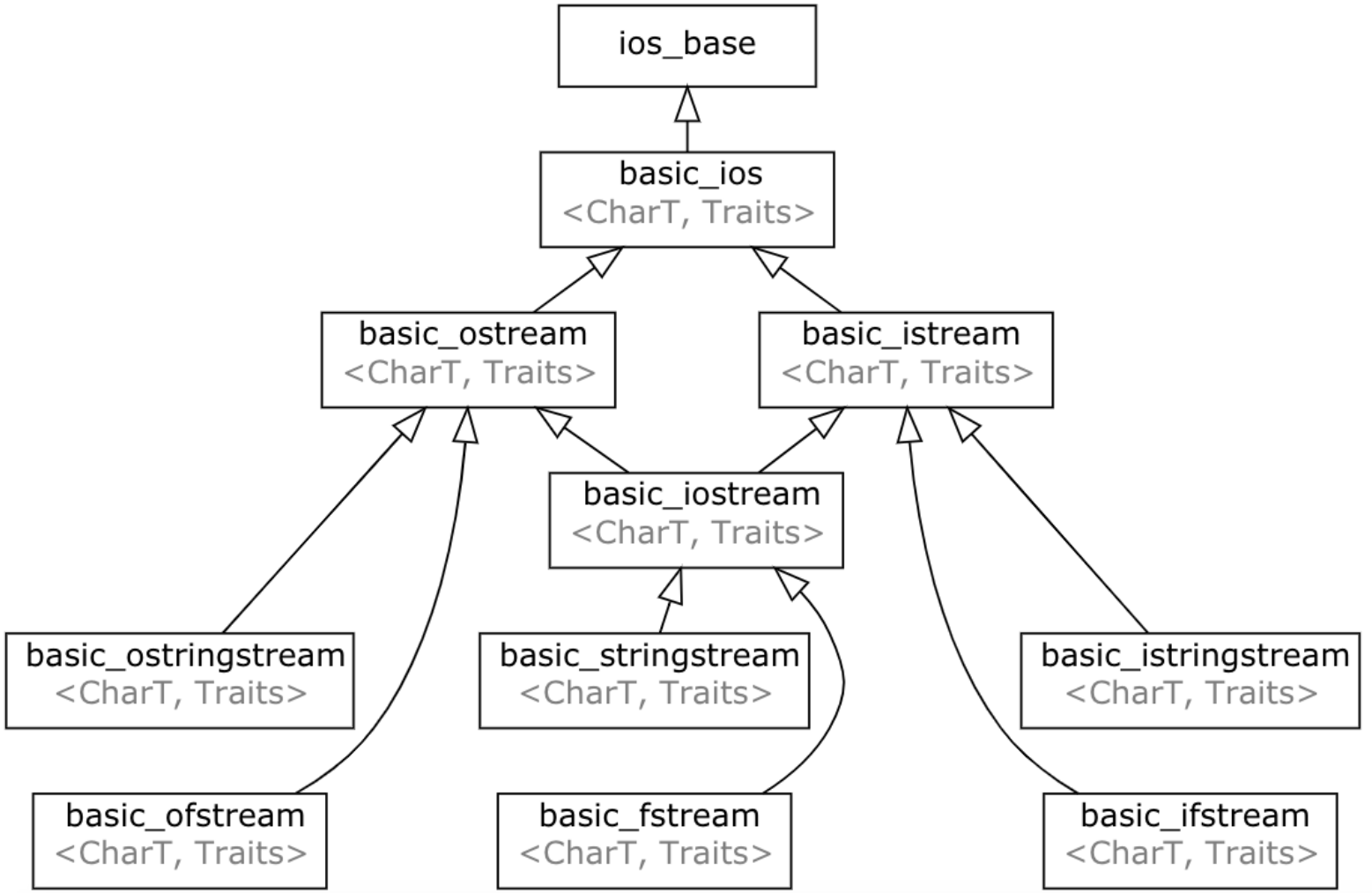
Вопрос: у нас доступен сишный ввод-вывод через `fprintf()` и `fscanf()`, зачем нужно что-то ещё?

У него есть свои минусы: он небезопасный, так как работает с сишными строчками (“сломать” программу с помощью переполнения буфера — классика жанра), недостаточно гибок, требует обязательную обвязку в виде объекта в любом более-менее крупном проекте.

Есть и плюсы: очень быстрый.

C++ представляет более гибкий ООП-подход к файловому вводу-выводу. Тоже есть свои плюсы-минусы: медленнее сишного подхода (хотя целиком зависит от реализации, иногда он просто реализован через `FILE`), исключения опциональны, использует RAII (в каком-то смысле).

Диаграмма наследования



Классы

Defined in header <ios>:

- ios_base — manages formatting flags and input/output exceptions (class)
- basic_ios — provides facilities for interfacing with objects that have std::basic_streambuf interface. (class template)

Defined in header <streambuf>:

- basic_streambuf — manages an arbitrary stream buffer (class template)

Defined in header <ostream>:

- basic_ostream — wraps a given abstract device (std::basic_streambuf) and provides high-level output interface (class template)

Defined in header <iostream>:

- basic_istream — wraps a given abstract device and provides high-level input interface (class template)
- basic_iostream

Реализации интерфейсов

File I/O implementation. Defined in header <fstream>:

- basic_filebuf — implements raw file device (class template)
- basic_ifstream — implements high-level file stream input operations (class template)
- basic_ofstream
- basic_fstream

String I/O implementation. Defined in header <sstream>:

- basic_stringbuf — implements raw string device (class template)
- basic_istringstream — implements high-level string stream input operations (class template)
- basic_ostringstream
- basic_stringstream

Typedefs

```
typedef basic_ios<char>                ios;
typedef basic_ios<wchar_t>             wios;

typedef basic_streambuf<char>      streambuf;
typedef basic_filebuf<char>          filebuf;
typedef basic_stringbuf<char>      stringbuf;

typedef basic_istream<char>          istream;
typedef basic_ostream<char>          ostream;
typedef basic_iostream<char>        iostream;

typedef basic_ifstream<char>         ifstream;
typedef basic_ofstream<char>         ofstream;

typedef basic_istringstream<char> istringstream;
typedef basic_ostringstream<char> ostringstream;
```

std::ios_base и std::basic_ios: флаги состояния

`goodbit` — no error.
`badbit` — irrecoverable stream error.
`failbit` — input/output operation failed (formatting or extraction error).
`eofbit` — associated input sequence has reached end-of-file
Можно устанавливать комбинацию флагов с помощью | . `rdstate()` — возвращает состояние флагов, `clear()` устанавливает состояние, `setstate()` — устанавливает нужное состояние в дополнение к текущим (= `clear(rdstate() | state)`).

Также состояние можно получать с помощью следующим способом:

ios_base::iostate flags			basic_ios accessors					
eofbit	failbit	badbit	good()	fail()	bad()	eof()	operator bool	operator!
false	false	false	true	false	false	false	true	false
false	false	true	false	true	true	false	false	true
false	true	false	false	true	false	false	false	true
false	true	true	false	true	true	false	false	true
true	false	false	false	false	false	true	true	false
true	false	true	false	true	true	true	false	true
true	true	false	false	true	false	true	false	true
true	true	true	false	true	true	true	false	true

Пример кода с флагами состояния

```
#include <iostream>
#include <string>

int main() {
    double n;
    while( std::cout << "Please, enter a number\n"
    && ! (std::cin >> n) ) {
        std::cin.clear();
        std::string line;
        std::getline(std::cin, line);
        std::cout << "I am sorry, but '" << line
        << "' is not a number\n";
    }
    std::cout << "TY for entering the number "
    << n << '\n';
}
```

std::ios_base : флаги форматирования

Устанавливаются с помощью `setf(int flag)` и `setf(flag, mask)` — установить конкретный флаг форматирования, `unsetf()` — убрать, `flags()` и `flags(fmtflags flags)` — для чтения/установки флагов. Ниже список:

Constant	Explanation
`dec`	use decimal base for integer I/O: see `std::dec`
`oct`	use octal base for integer I/O: see `std::oct`
`hex`	use hexadecimal base for integer I/O: see `std::hex`
`basefield`	`dec|oct|hex|0`. Useful for masking operations
`left`	left adjustment (adds fill characters to the right): see `std::left`
`right`	right adjustment (adds fill characters to the left): see `std::right`
`internal`	internal adjustment (adds fill characters to the internal designated point): see `std::internal`
`adjustfield`	`left|right|internal`. Useful for masking operations

std::ios_base : флаги форматирования, продолжение

`scientific`	generate floating point types using scientific notation, or hex notation if combined with fixed: see `std::scientific`
`fixed`	generate floating point types using fixed notation, or hex notation if combined with scientific: see `std::fixed`
`floatfield`	`scientific|fixed|(scientific|fixed)|0`. Useful for masking operations
`boolalpha`	insert and extract `bool` type in alphanumeric format: see `std::boolalpha`
`showbase`	generate a prefix indicating the numeric base for integer output, require the currency indicator in monetary I/O: see `std::showbase`
`showpoint`	generate a decimal-point character unconditionally for floating-point number output: see `std::showpoint`
`showpos`	generate a `+` character for non-negative numeric output: see `std::showpos`
`skipws`	skip leading whitespace before certain input operations: see `std::skipws`
`unitbuf`	flush the output after each output operation: see `std::unitbuf`
`uppercase`	replace certain lowercase letters with their uppercase equivalents in certain output operations: see `std::uppercase`

Плюс есть следующие методы, используемые при форматировании:
`precision()` и `precision(int)` — установить число знаков для вещественных чисел.
`width()` и `width(int)` — установить ширину поля.

<iomanip> и не только: манипуляторы

Манипулятор — вспомогательная функция, которая позволяет управлять состоянием потока ввода (вывода, ввода/вывода и т.д.) с помощью operator« и operator».

Что это означает? Например, у объекта типа basic_ostream определено следующее:

```
basic_ostream& operator<<(
std::ios_base& (*func)(std::ios_base&) );
```

Этот оператор принимает указатель на функцию типа указанной ниже и запускает её с нужным параметром:

```
std::ios_base& p5(std::ios_base& os) {
os.precision(5);
return os; }
```

Т.о., мы можем писать std::cout « p5; вместо std::cout.precision(5);. Манипулятор может принимать и ostream.

Манипуляторы с параметром

Также бывают манипуляторы с параметром.
Как сделать такой самому? Можно через объект, у которого определено

```
class mysetprecision {
int prec;
public mysetprecision(int prec) : prec(prec) { }

friend ostream&
operator<<(ostream& os, mysetprecision& msp) {
os.precision(msp.prec);
return os;
}

};
```

Тогда можно делать `std::cout « mysetprecision(10);` Получили т.н. *эффектор*.

Список манипуляторов

Defined in header <ios>

boolalpha **noboolalpha**	switches between textual and numeric representation of booleans (function)
showbase **noshowbase**	controls whether prefix is used to indicate numeric base (function)
showpoint **noshowpoint**	controls whether decimal point is always included in floating-point representation (function)
showpos **noshowpos**	controls whether the + sign used with non-negative numbers (function)
skipws **noskipws**	controls whether leading whitespace is skipped on input (function)
uppercase **nouppercase**	controls whether uppercase characters are used with some output formats (function)
unitbuf **nounitbuf**	controls whether output is flushed after each operation (function)
internal **left** **right**	sets the placement of fill characters (function)
dec **hex** **oct**	changes the base used for integer I/O (function)
fixed **scientific** **hexfloat** (C++11) **defaultfloat** (C++11)	changes formatting used for floating-point I/O (function)

Список манипуляторов: продолжение

Defined in header <istream>	
ws	consumes whitespace (function template)
Defined in header <ostream>	
ends	outputs '\0' (function template)
flush	flushes the output stream (function template)
endl	outputs '\n' and flushes the output stream (function template)
Defined in header <iomanip>	
resetiosflags	clears the specified ios_base flags (function)
setiosflags	sets the specified ios_base flags (function)
setbase	changes the base used for integer I/O (function)
setfill	changes the fill character (function template)
setprecision	changes floating-point precision (function)
setw	changes the width of the next input/output field (function)

А теперь посмотрим в пример на бумажке.

fstream: режимы

Реализует ввод-вывод файл. Конструктор:

```
basic_fstream( const char* filename,
std::ios_base::openmode mode =
ios_base::in|ios_base::out );
```

Что такое `mode`? Битовая маска.

1. `app` — seek to the end of stream before each write
2. `binary` — open in binary mode
3. `in` — open for reading
4. `out` — open for writing
5. `trunc` — discard the contents of the stream when opening
6. `ate` — seek to the end of stream immediately after open

Аналогично — запустить конструктор по-умолчанию, после чего сделать вызов `open()` с указанными параметрами.
C++11 — filename может быть `std::string`.

fstream: чтение-запись

Открыть поток — `open()`, закрыть — `close()`, сбросить данные на диск — `flush()`.
Замечание: `os « std::endl` автоматически делает сброс.
Как читать/писать?

- `operator»` и `operator«`, если они определены для нужных типов
- `put()` и `get()` — читать/писать по одному / *n* символам до разделителя (по-умолчанию до конца строки) / в буфер
- `read()` и `write()` — читать/писать *n* символов, `char *`
- метод `getline()` — читать *n* символов в `char *`
- функция `getline()` — читать *n* символов в `std:string`
- Через обращение к буферу `rdbuf`.

Теперь смотрим в листочки на пример.

fstream: различное

- `peek()` — прочитать символ, не вынимая из потока
- `gcount()` — количество символов, вытащенных последней операцией ввода без форматирования
- `tellg()` и `seekg()` — получить/установить позицию для чтения
- `tellp()` и `seekp()` — получить/установить позицию для записи

Позицию можно устанавливать абсолютно (`seekp(7)`) и относительно (`seekp(0, std::ios_base::end)`):

- `beg` — начало потока
- `end` — конец потока
- `cur` — относительно текущей позиции

<sstream>: stringstream

Реализует ввод/вывод в строчку внутри памяти. Очень полезен при форматировании.
Абсолютно аналогичный `fstream` интерфейс, кроме того, что не требуется имя файла.

Плюс самое главное: метод `str()` — возвращает копию лежащей внутри строки, `str(new_str)` — меняет лежащую внутри строку.

За примерами смотрим в бумажку.

Исключения

Можно установить у `basic_ios` маску, при которой будет кидаться исключение с помощью метода `exceptions(mask)`. Без параметров этот же метод вернёт маску. Пример кода:

```
#include <iostream>
#include <fstream>
int main()
{
   int ivalue;
   try {
         std::ifstream in("in.txt");
         in.exceptions(std::ifstream::failbit);
         in >> ivalue;
   } catch (std::ios_base::failure &fail) {
         // handle exception here
   }
}
```

Домашнее задание

Дан tab (или comma) separated file. В первой строке — разделенные табуляцией названия столбцов. Дальше данные — вещественные числа, разделенные табуляциями и переносами строк.

Требуется разобрать исходный файл, вывести содержимое разбора на экран. В случае ошибки при парсинге — кинуть исключение и показать, в чём проблема.

Подсказка: хранить названия можно с помощью `vector<string>`, данные — `vector<vector<double> >`.

Ориентировочный дедлайн — через неделю.