ВВОД И ВЫВОД

Механизмы работы с пользовательским вводом в C++. Локализация.

К. Владимиров, Intel, 2017

- Ввод и вывод
- Манипуляторы
- Локализация
- Преобразования строк

Ввод и вывод в C: всё есть FILE

- FILE это implementation-defined структура, поэтому обычно оперируют FILE*
- Всего файлов можно открыть не меньше FOPEN_MAX (не меньше 8)
- Весь ввод/вывод работает только с «файлами»
- «Файлы» можно открывать и закрывать (fopen, fclose), но главное, что в них можно писать (иногда) и из них можно читать (тоже иногда).
- Три стандартных файла не требуется специально открывать:
  - stdin – стандартный поток ввода (обычно смотрит на консоль)
  - stdout – стандартный поток вывода (обычно тоже консоль)
  - stderr – стандартный поток сообщений об ошибках (обычно опять консоль)

Ввод и вывод в C: буферизация

- Буферизованные
  - Построчная буферизация
    - stdout
    - stdin
  - Полная буферизация
- Не буферизованные
    - stderr

- Работа с буфером
  - setvbuf
  - fflush

```
setbuf(stdout, NULL);

setvbuf(fp, // это FILE* fp
   NULL, _IOFBF, BSIZE);

fprintf (stdout, "Hello ");

fprintf (fp, "Hello ");

fflush (fp); // запись в файл
```

Простая задача

Что нужно написать перед следующим кодом, чтобы гарантировать, что вывод произойдет без разрыва одной строчкой?

```
fprintf (stdout, "%s, ", "Hello");

delay(5);

fprintf (stdout, "%s!\n", "world");
```

Простая задача: ответ

Предполагаем, что где-то до этого было выполнено нечто вроде:

setbuf (stdout, NULL);

Что нужно написать перед следующим кодом, чтобы гарантировать, что вывод произойдет без разрыва одной строчкой?

setvbuf (stdout, NULL, _IOFBF, 1024); // не обязательно 1024

fprintf (stdout, "%s, ", "Hello");

delay(5);

fprintf (stdout, "%s!\n", "world");

Ввод и вывод в C: форматирование

- Неформатированный вывод: fputs("Hello, world\n", stdout);
- Форматированный вывод: fprintf(stdout, "%s\n", "Hello, world");

Всё это доступно и в мире C++ через <cstdio> и std::fprintf

- Какие проблемы создаёт C-style IO?

Ввод и вывод в C: форматирование

- Неформатированный вывод: fputs("Hello, world\n", stdout);
- Форматированный вывод: fprintf(stdout, "%s\n", "Hello, world");

Всё это доступно и в мире C++ через <cstdio> и std::fprintf

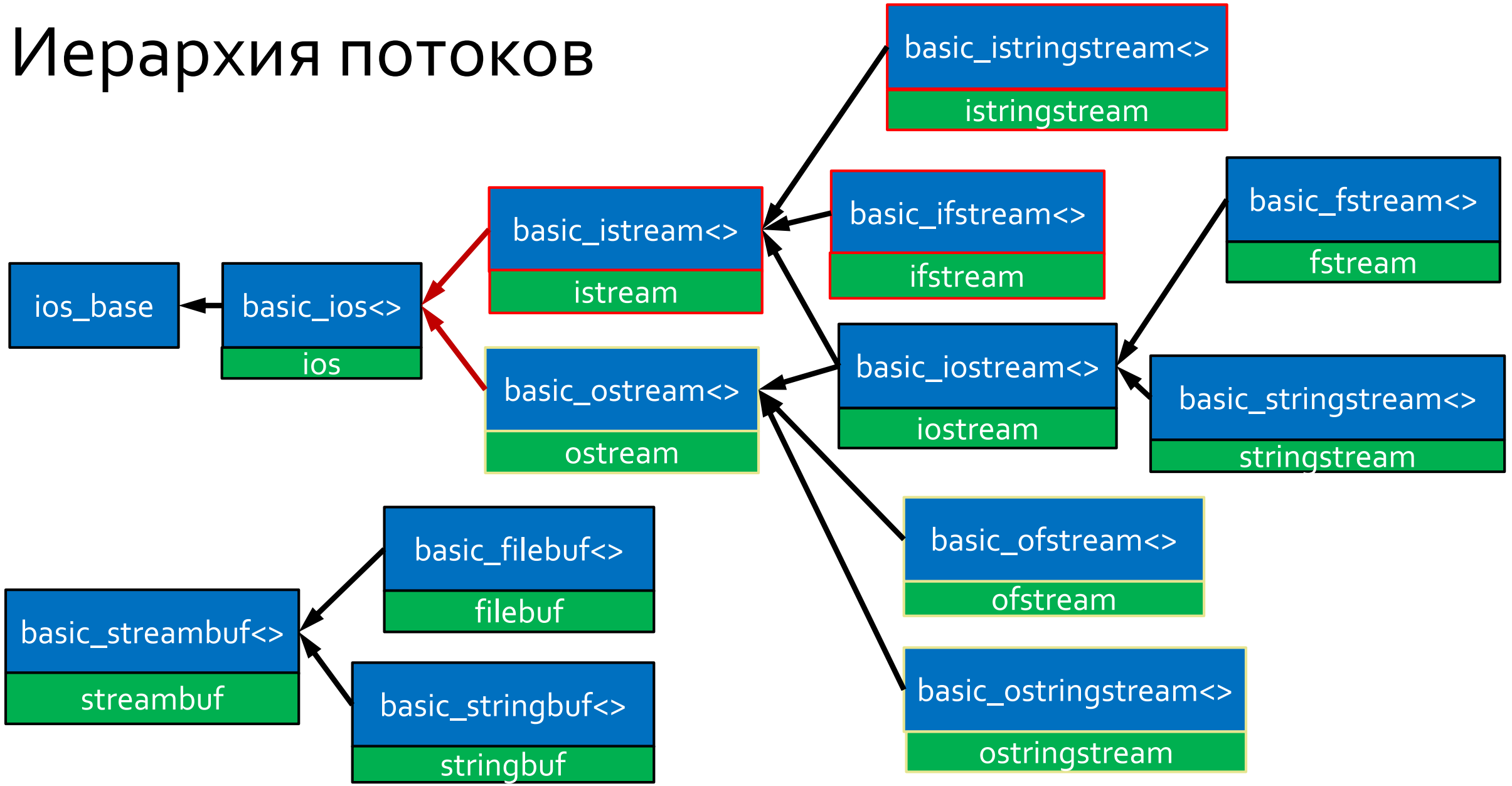
- Какие проблемы создаёт C-style IO?
  - Нерасширяемость. Например как определить новый форматный спецификатор?
  - Неочевидность: выбор спецификатора определяется размером, который может не быть известен. Пример: int64_t требует препроцессора printf("x = %"PRIu64"d", x);
  - Небезопасность относительно типов: printf("%s\n", 1);
  - Небезопасность относительно количества аргументов
  - Дополнительная проблема: нерасширяемость самого механизма FILE* через наследование

Ввод и вывод в C++: потоки

Тип	Файл	Поток
Стандартный ввод	stdin	cin
Стандартный вывод	stdout	cout
Сообщения об ошибках	stderr	cerr
Логгирование	--	clog
Дисковый файл	FILE* f	fstream f
Строка	char buf[N]	sstream sf

- Поток может быть ассоциирован с файлом, но это не файл
- Поток это объект, у него есть методы и состояние
- Не стоит путать stream и thread. Традиционно stream это поток ввода/вывода, а thread это поток (нить) исполнения кода. Очень разные вещи

Иерархия потоков



Ввод и вывод в C++: форматирование

- Форматированный вывод через перегрузку сдвига

```
cout << str;
```

- Форматные спецификаторы

```
int n = 42; cout << n << endl; // на экране 42
cout << hex << n << endl; // на экране 2A
```

- Форматированный ввод тоже через перегрузку сдвига

```
int n; cin >> n; // ожидается десятичное число из cin
```

Ввод сложнее вывода, так как снаружи может придти что угодно.

Неформатированный ввод

Основные средства: get, peek, putback

```
char c = cin.get(); // можно cin.peek() тогда putback ниже не нужен
if ( (c >= '0') && (c <= '9') ) {
  // обработка числа
}  else {
    string str;
    cin.putback (c); // кладём обратно подсмотренный символ
    getline (cin, str); // getline (istream, string) но не cin.getline (char *, int)
    // обработка строки
}
```

Задача: забыть символ

```
cout << "Please enter a number: " << "\n";
cin >> num;
cout << "Your number is: " << num << "\n";
cout << "Please enter your name: \n";
getline (cin, mystr); // упс... тут что-то пошло не так
```

Состояния потоков и обработка ошибок

Буферизация

Сцепленность потоков

```
do {
  ch = cin.get();
  cout.put(ch);
} while (ch!='.');
```

Вывод в файлы

```
outfile.write ("This is an apple",16);
 long pos = outfile.tellp();
 outfile.seekp (pos-7);
 outfile.write (" sam",4);
```

Перенаправление потоков

```
std::ofstream filestr;
 filestr.open ("test.txt");

 backup = std::cout.rdbuf();     // back up cout's streambuf

 psbuf = filestr.rdbuf();       // get file's streambuf
 std::cout.rdbuf(psbuf);        // assign streambuf to cout

 std::cout << "This is written to the file";

 std::cout.rdbuf(backup);       // restore cout's original streambuf
```

- Ввод и вывод
- Манипуляторы
- Локализация
- Преобразования строк

- Ввод и вывод
- Манипуляторы
- Локализация
- Преобразования строк

Кратко о структуре Unicode

- Не-юникодные кодировки (они же системы трансляции)
  - ANSI (7 bytes)
  - ASCII (8 bytes) = ANSI + codepage (например 1251 и 866)
- Юникодные системы кодировки (символ + число)
  - UCS-2 (устаревшая система, 16 бит)
  - UCS-4 (число U+0041 это английское A, а число U+0410 это русское A)
- Юникодные форматы преобразования
  - UTF-8 (от 1 до 6 байт на символ)
    - в ней U+0410 это {0xD0, 0x90}, зато U+0041 это {0x41}
  - UTF-16 (покрывает UCS-2)
    - в UTF16-LE U+0410 это {0x10, 0x04} но и U+0041 это {0x41, 0x00}
  - UTF-32 (покрывает UCS-4)

Символы

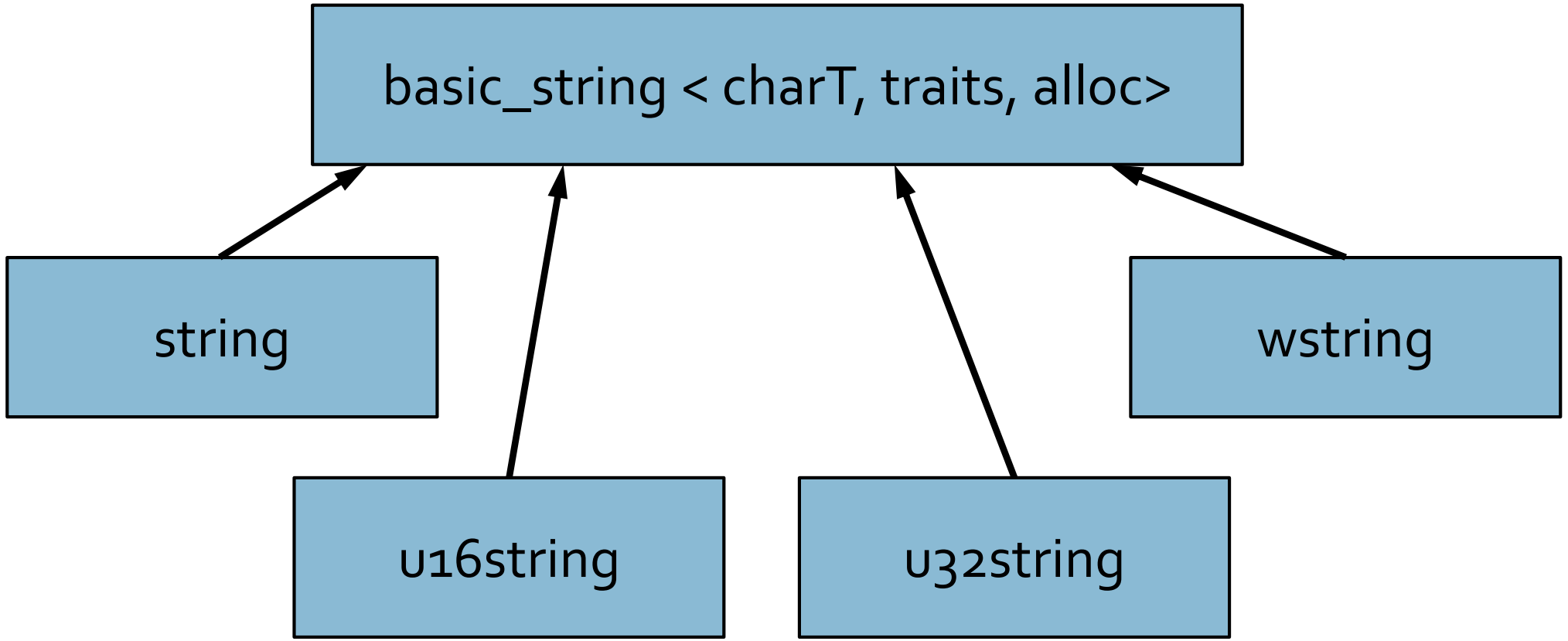
- char – наименьший тип (sizeof(char) == 1)
- char16_t – символ из набора UCS-2
- char32_t – символ из набора UCS-4
- wchar_t – наибольший символьный тип среди всех системных локалей

- Ввод и вывод
- Манипуляторы
- Локализация
- Преобразования строк

Работа со строками может быть сложна

```
std::setlocale(LC_ALL, "");
const std::wstring ws = L"ħëłlö";
const std::locale locale("");
typedef std::codecvt<wchar_t, char, std::mbstate_t> converter_type;
const converter_type& converter = std::use_facet<converter_type>(locale);
std::vector<char> to(ws.length() * converter.max_length());
std::mbstate_t state;
const wchar_t* from_next;
char* to_next;
const converter_type::result result =
  converter.out(state, ws.data(), ws.data() + ws.length(), from_next, &to[0],
&to[0] + to.size(), to_next);
if (result != converter_type::ok && result != converter_type::noconv)
  return false;
const std::string s(&to[0], to_next);
```

Класс basic_string и его наследники



Преобразования строк

- wstring_convert – преобразование из char-string в wchar_t-string

Используемые facets

- codecvt_utf8
- codecvt_utf8_utf16

Литература

- ISO/IEC, "Information technology -- Programming languages – C++", ISO/IEC 14882:2014, 2014
- The C++ Programming Language (4th Edition)
- Nicolai M. Josuttis, The C++ Standard Library - A Tutorial and Reference, 2nd Edition , Addison-Wesley, 2012
- Scott Meyers, Effective STL, 50 specific ways to improve your use of the standard template library, Addison-Wesley, 2001