ВВОД И ВЫВОД

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

К. Владимиров, Intel, 2017

- Ввод и вывод
- Файлы и строки
- Буферизация
- 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: форматирование

- Неформатированный ввод и вывод

```
fputs ("Hello, world\n", stdout);
char str[80]; fgets (str, 80, stdin);
```

- Форматированный ввод и вывод

```
fprintf (stdout, "%s\n", "Hello, world");
char str[81]; fscanf (stdin, "%80s", str);
```

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); // допустим эта функция делает 5 секунд задержку

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");

Чего же ещё желать?

- Вопрос: какие проблемы создаёт C-style IO?

Увы, проблемы есть.

- Нерасширяемость. Например как определить новый форматный спецификатор?
- Неочевидность: выбор спецификатора определяется размером, который может не быть известен.
  Пример: `int64_t x = 2; printf("x = %"PRIu64"d", x);`
- Небезопасность относительно типов: `printf("%s\n", x);`
- Небезопасность относительно количества аргументов
- Нерасширяемость самого механизма `FILE*` на строки, память, etc

Обсуждение

- Как могло бы выглядеть решение изложенных проблем?

Решение в стиле C++

- Тип буфера (файл, строка, консоль) отделен от форматирования ввода/вывода
- Форматные спецификаторы в виде отдельных классов
- Пользовательские классы должны иметь возможность переопределить ввод и вывод для себя
- Типизированные аргументы с фиксированным количеством аргументов у каждого оператора

Форматный вывод в C++ необычен

Язык	Форматный вывод
C	`fprintf (stdout, "Kill %x cats", n)`
Python	`print ("Kill {0:x} cats".format(n))`
Java	`System.out.println(String.format("Kill %x cats", n))`
C#	`Console.Write("Kill {0:x} cats", n)`
Rust	`print!("Kill {0:x} cats", n)`
Go	`fmt.Printf("Kill %x cats", n)`

Форматный вывод в C++ **необычен**

Язык	Форматный вывод
C++	`std::cout << "Kill" << std::hex << n << "cats"`
C	`fprintf (stdout, "Kill %x cats", n)`
Python	`print ("Kill {0:x} cats".format(n))`
Java	`System.out.println(String.format("Kill %x cats", n))`
C#	`Console.Write("Kill {0:x} cats", n)`
Rust	`print!("Kill {0:x} cats", n)`
Go	`fmt.Printf("Kill %x cats", n)`

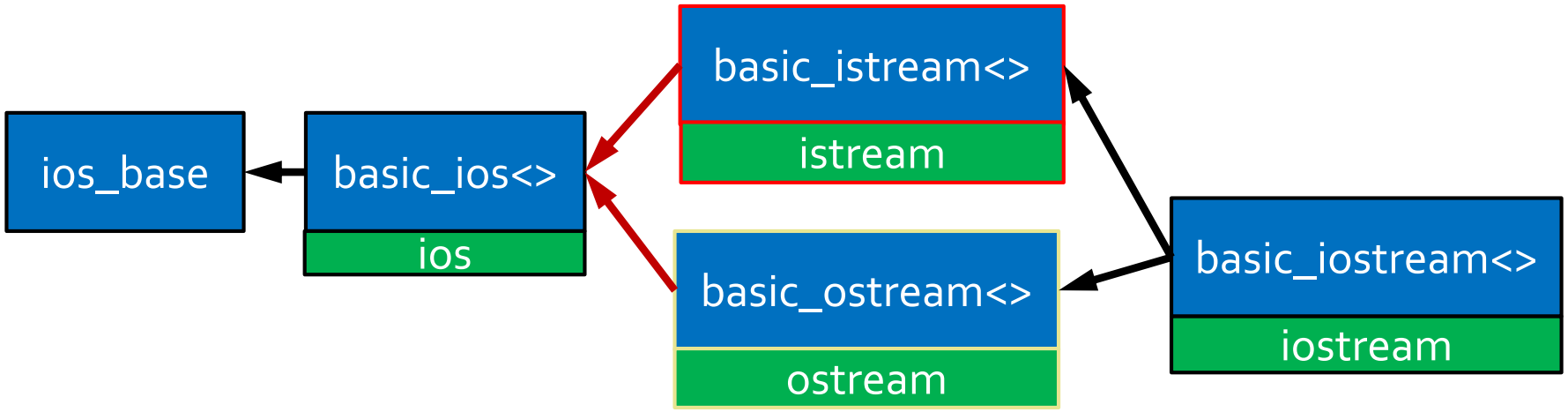

Ты – особенный!

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.

Пример

```
int n;
cin >> hex >> n;
cout << dec << n << endl;
```

1. На входе "2A". На экране "42"
2. На входе " 2A". На экране "42"
3. На входе "2.2A". На экране "2"
4. На входе "AAAAAAAAAAA". На экране "2147483647"
5. На входе "ZZZ". На экране "0".

Перегрузка для своего класса

```
class MyClass {
  //  something private
public:
  // print have access to private data
  void print (std::ostream& stream) const;
};

std::ostream&
operator <<(std::ostream& stream, const MyClass& rhs) {
  rhs.print (stream);
  return stream;
}
```

Обсуждение

- При перегрузке слайдом ранее использован ostream&. Хорошее ли это решение? Есть ли лучшие альтернативы?
  1. ios_base&
  2. basic_ios<charT, traits>&
  3. basic_ostream<charT, traits>&
  4. Оставить ostream&
  5. Ваши варианты?

Использование

```
MyClass obj;
cout << obj << endl;
// Раскрывается в:
1. operator << (operator << (cout, obj), endl);
2. Вызов obj.print()
3. Вызов operator << (cout, endl)
4. Вызов endl (cout)
```

Как устроен манипулятор endl

```
template <typename charT, typename traits>
basic_ostream<charT, traits>&
endl (basic_ostream<charT, traits>& strm)
{
  strm.put(strm.widen('\n'));
  strm.flush();
  return strm;
}
```

- put это специальный метод для неформатированного вывода
- flush это сброс буфера
- метод widen интуитивно понятен, но будет рассмотрен позднее

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
}
```

Внезапная проблема

```
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
```

Внезапная проблема: решение

```
cout << "Please enter a number: " << "\n";
cin >> num; // здесь был нажат Enter и конец строки остался в cin
cout << "Your number is: " << num << "\n";
cout << "Please enter your name: \n";
cin.ignore(); // здесь лишний Enter был забыт
getline (cin, mystr); // теперь всё хорошо
```

Состояния потоков и обработка ошибок

- std::ios_base::eofbit – считан конец файла
- std::ios_base::failbit – восстановимая ошибка (например ошибка форматирования)
- std::ios_base::badbit – серьёзная ошибка (порча потока, потеря данных)

Работа в основном возложена на функции:

- rdstate/clear – прочитать/сбросить флаги
- fail, operator!() – true если failbit || badbit
- operator bool() – true если !fail()

Примеры

1. ```
   int n;
   while (cin >> n) {
     cout << n << endl;
     cin.ignore(); // eating Enter hit
   }
   ```
   Вопрос: что будет на экране, если ввод "1.1"?

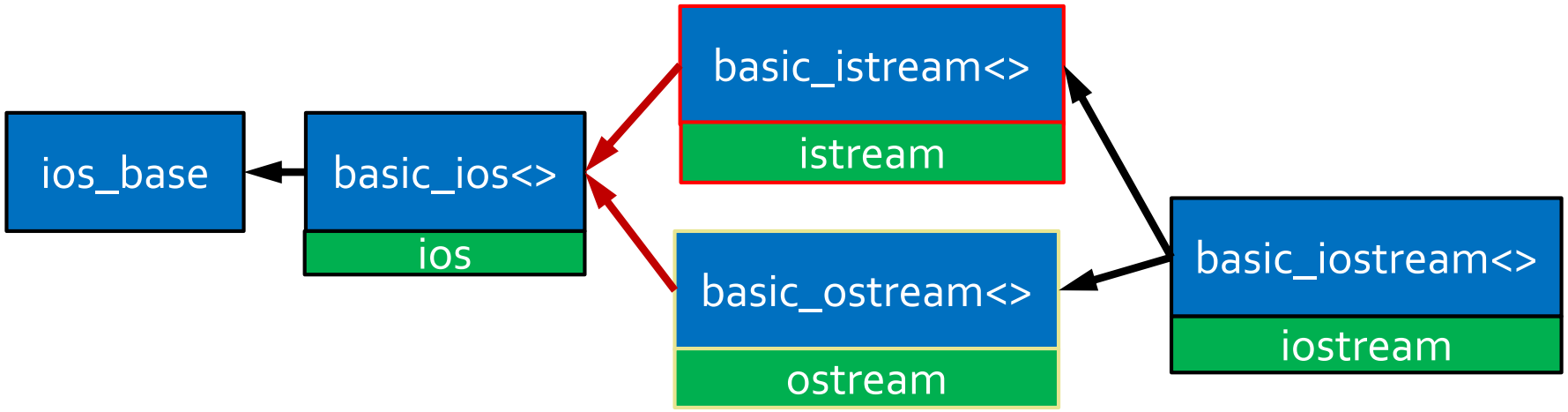
2. ```
   if (!cin >> n){
     // process errors
   }
   ```
   Вопрос: что плохо в таком подходе?

Обсуждение

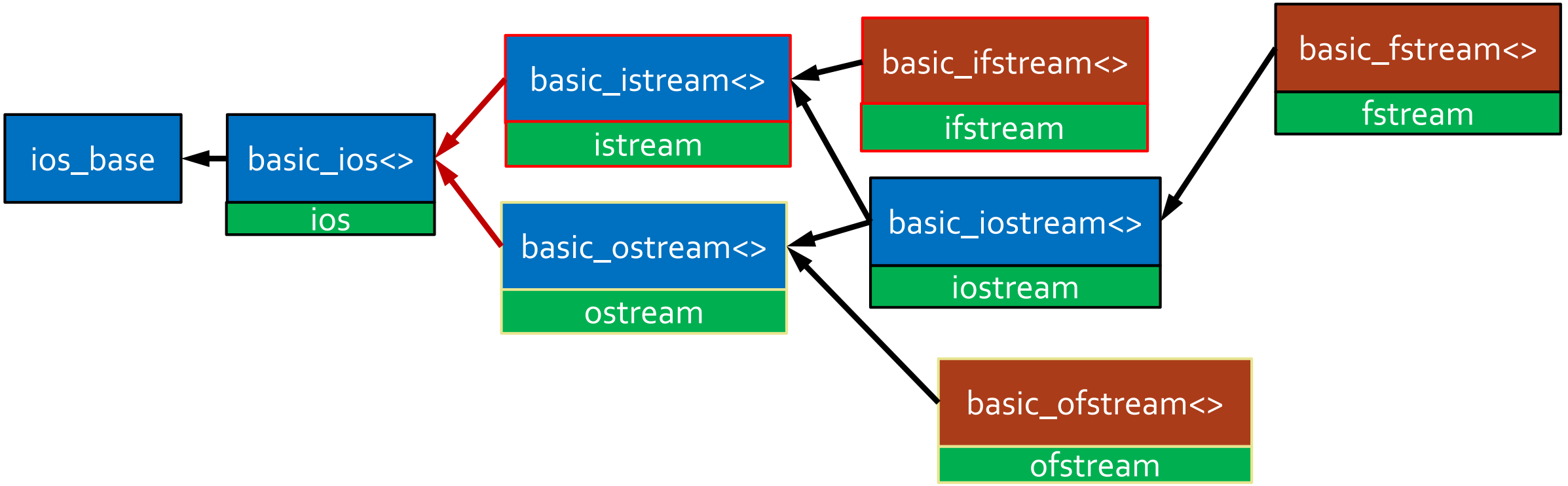
- Вывод чаще нужен форматированный, а ввод – не форматированный. В каких обстоятельствах уместны неформатированный вывод и форматированный ввод?

- Ввод и вывод
- **Файлы** и строки
- Буферизация
- Локализация
- Преобразования строк

Иерархия потоков



Иерархия потоков



Вывод в файл: пример

```
ofstream file("charset.txt");

if (! file)  {   /* process errors */    }

// write character set
for (int i = 32; i != 256; ++i) {

  file << "value: " << setw(3) << i << " "
       << "char: " << static_cast<char>(i) << endl;

}
```

По сути вывод ничем не отличается. Файл будет автоматически открыт и закрыт при выходе из области видимости.

Вывод содержимого файла на экран*

```
{
  ifstream file(filename);
  char c;
  while (file.get(c))
    cout.put(c);
} // здесь файл будет закрыт
```

- Естественные вопросы:
  - Что если файл ещё нужен за пределами области видимости?
  - Что если один и тот же объект хочется использовать дважды?

*Приведенный способ не вполне хорош, но об этом позже

Move-семантика для потоков

- Начиная с C++11 работает следующее:

```
std::ofstream openFile (const std::string& filename){
  std::ofstream file(filename);
  file << "hello, ";
  return file;
}

std::ofstream file;
file = openFile("xyz.tmp"); // move-constructed
file << "world" << std::endl;
```

Многоразовое использование

```
const char *filenames[] = {"testfile1.txt", "testfile2.txt"};

ifstream file;

for (int idx = 0; idx < 2; ++idx) {
  file.open(filenames[idx]);
  char c;

  while (file.get(c))
    cout.put(c);

  file.close();
}
```

Режимы открытия файлов

```
// открыть a.tmp в режиме дозаписи
ofstream file("a.tmp", std::ios_base::out|std::ios_base::app);
```

Режим	Файлы	Потоки
Чтение	"r"	in
Перезапись	"w"	out или out | trunc
Дозапись	"a" или "a+"	app или out | app
Чтение/запись с начала	"r+"	in | out
Чтение/перезапись	"w+"	in | out | trunc

Позиционирование

```
ofstream outfile("a.tmp");
outfile << "This is an apple";
long pos = outfile.tellp(); // текущая позиция
outfile.seekp (pos-7); // перейти к pos-7
outfile << " sam";
```

Вопрос: что в выходном файле?

Позиционирование

```
ofstream outfile("a.tmp");

outfile << "This is an apple";

long pos = outfile.tellp(); // текущая позиция

outfile.seekp (pos - 7); // перейти к -7му символу от pos

outfile << " sam";
```

В выходном файле "This is a sample", но long это не лучшая идея.

Позиционирование

```
ofstream outfile("a.tmp");
outfile << "This is an apple";
auto pos = outfile.tellp();
outfile.seekp (pos - static_cast<decltype(pos)>(7));
outfile << " sam";
```

Можно выкрутится. Но лучше относительное позиционирование.

Позиционирование

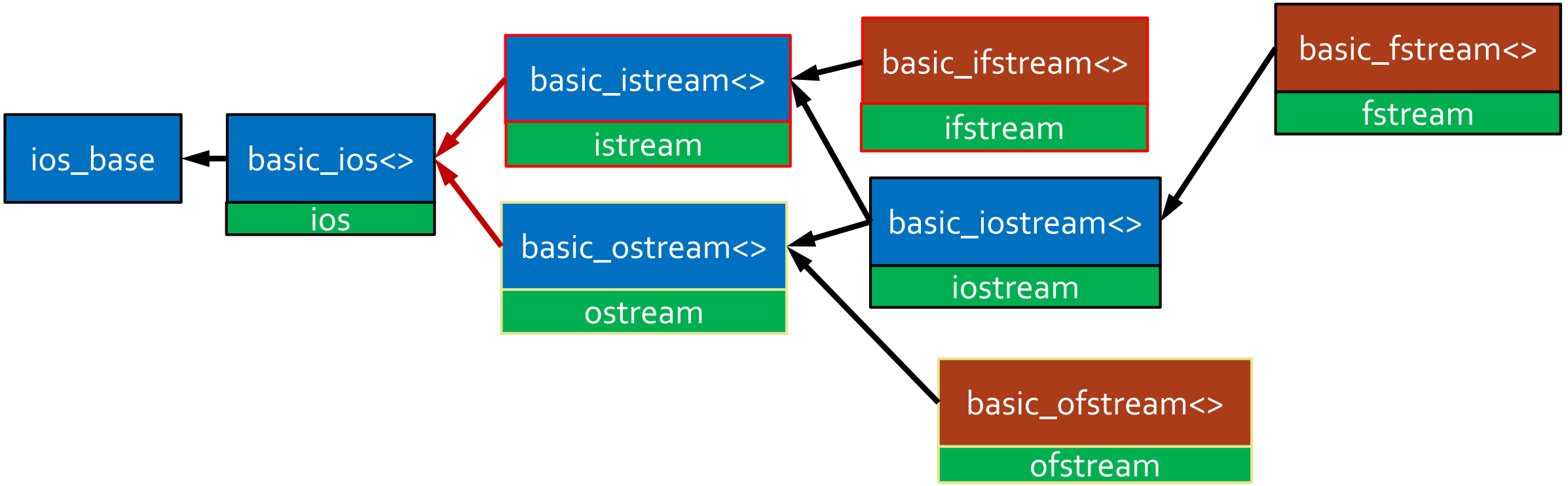
```
ofstream outfile("a.tmp");
outfile << "This is an apple";
outfile.seekp (-7, std::ios::cur); // от cur
outfile << " sam";
```

Относительно позиционировать можно от:

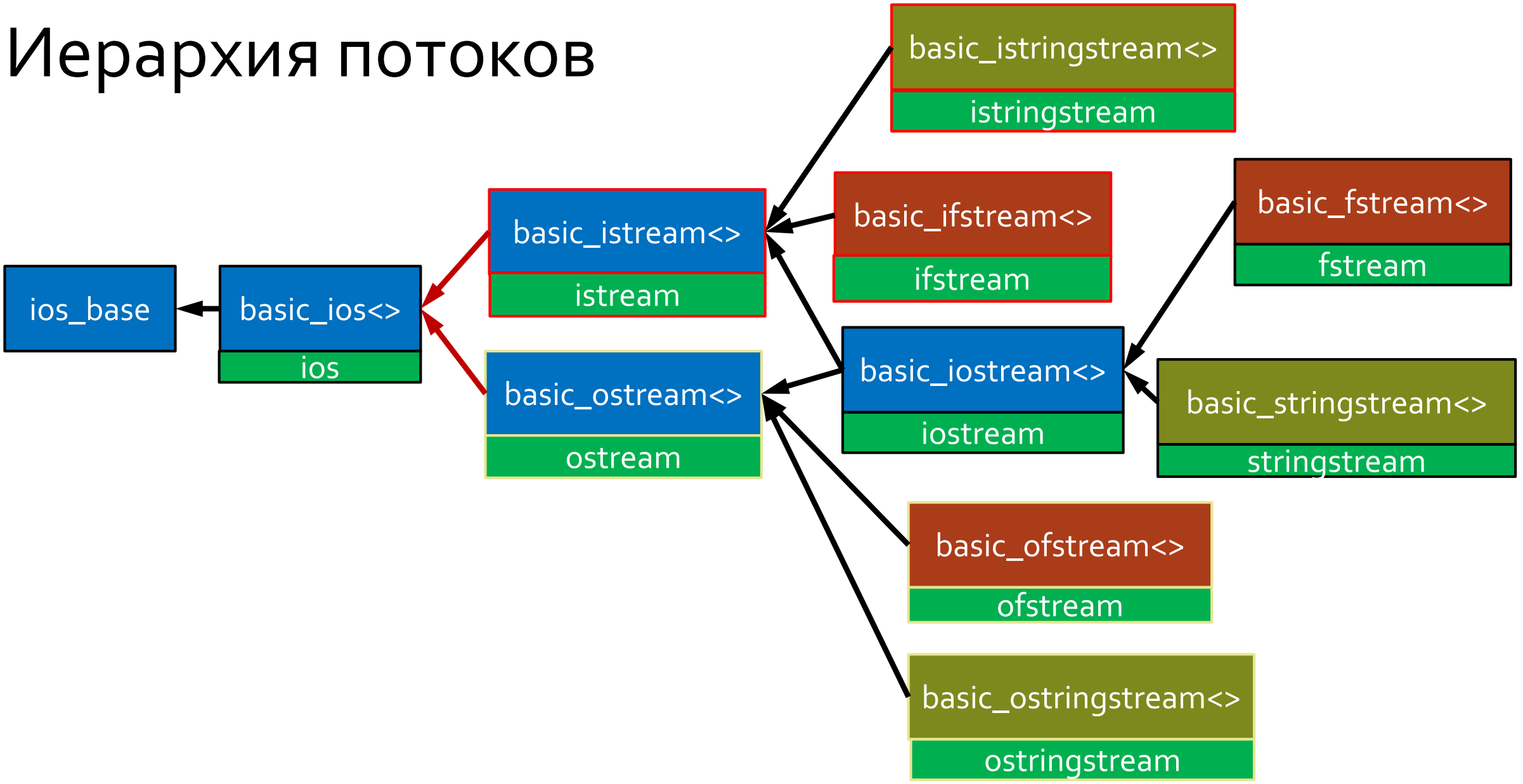
- std::ios::beg
- std::ios::end
- std::ios::cur

- Ввод и вывод
- Файлы и **строки**
- Буферизация
- Локализация
- Преобразования строк

Иерархия потоков



Иерархия потоков



Работа вполне прозрачна

```
std::ostringstream fst; int n; float f;
fst << 42.2442;

std::string s1 = fst.str(); // поток в строку через .str()
std::istringstream iss(s1);
iss >> n >> f;

std::string s2("value: ");

// ate означает "at the end"
std::ostringstream snd(s2, std::ios::out|std::ios::ate);
snd << std::hex << n << " " << f;

std::cout << snd.str() << std::endl;
```

Простой вопрос: что на экране?

Работа вполне прозрачна

```
std::ostringstream fst; int n; float f;
fst << 42.2442;

std::string s1 = fst.str(); // поток в строку через .str()
std::istringstream iss(s1);
iss >> n >> f;

std::string s2("value: ");

// ate означает "at the end"
std::ostringstream snd(s2, std::ios::out|std::ios::ate);
snd << std::hex << n << " " << f;

std::cout << snd.str() << std::endl;

На экране: "value: 2a 0.2442"
```

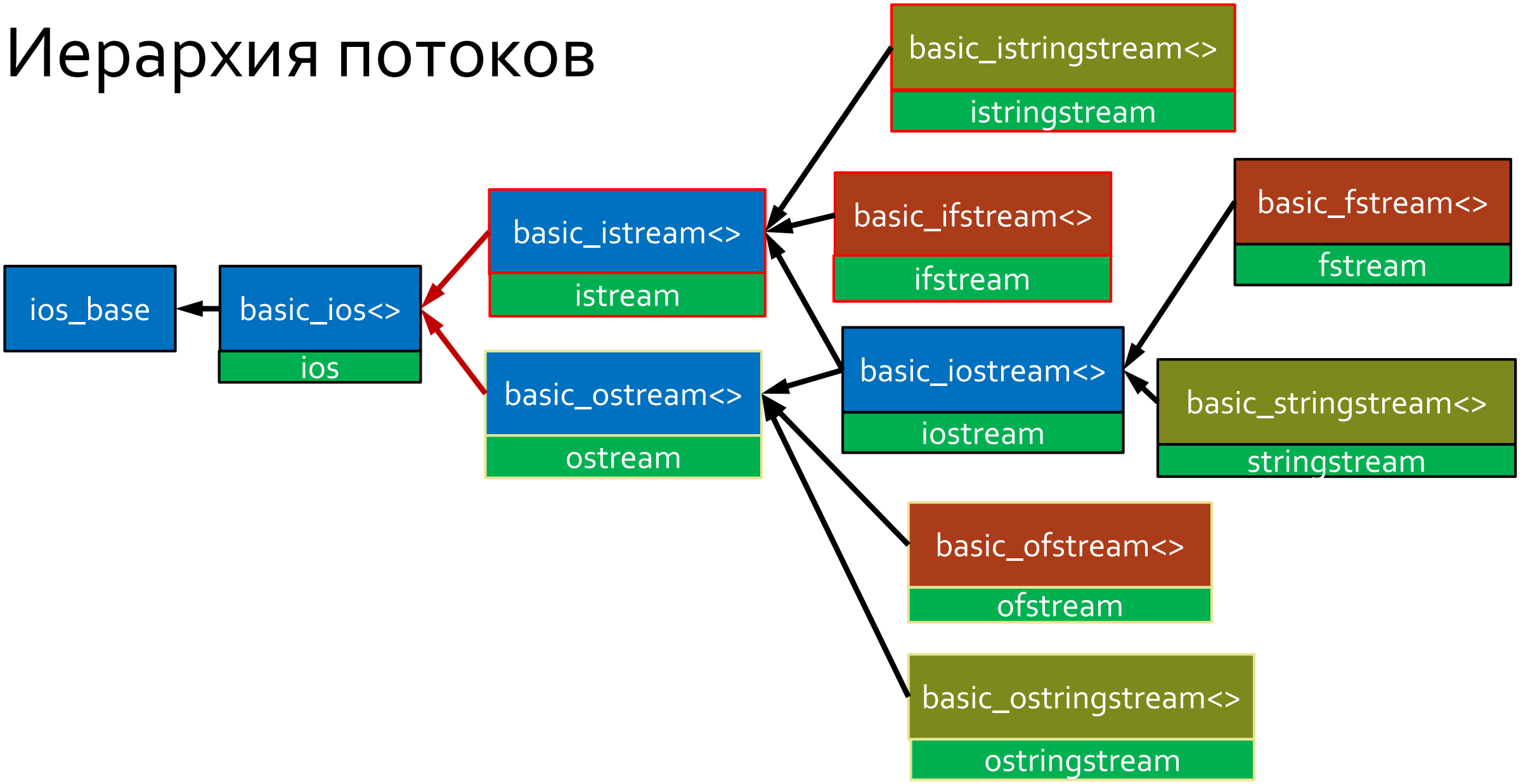
Неименованные потоки (C++11)

```
void parseName(string name)
{
  string s1, s2, s3;
  istringstream(name) >> s1 >> s2 >> s3;
  .....
```

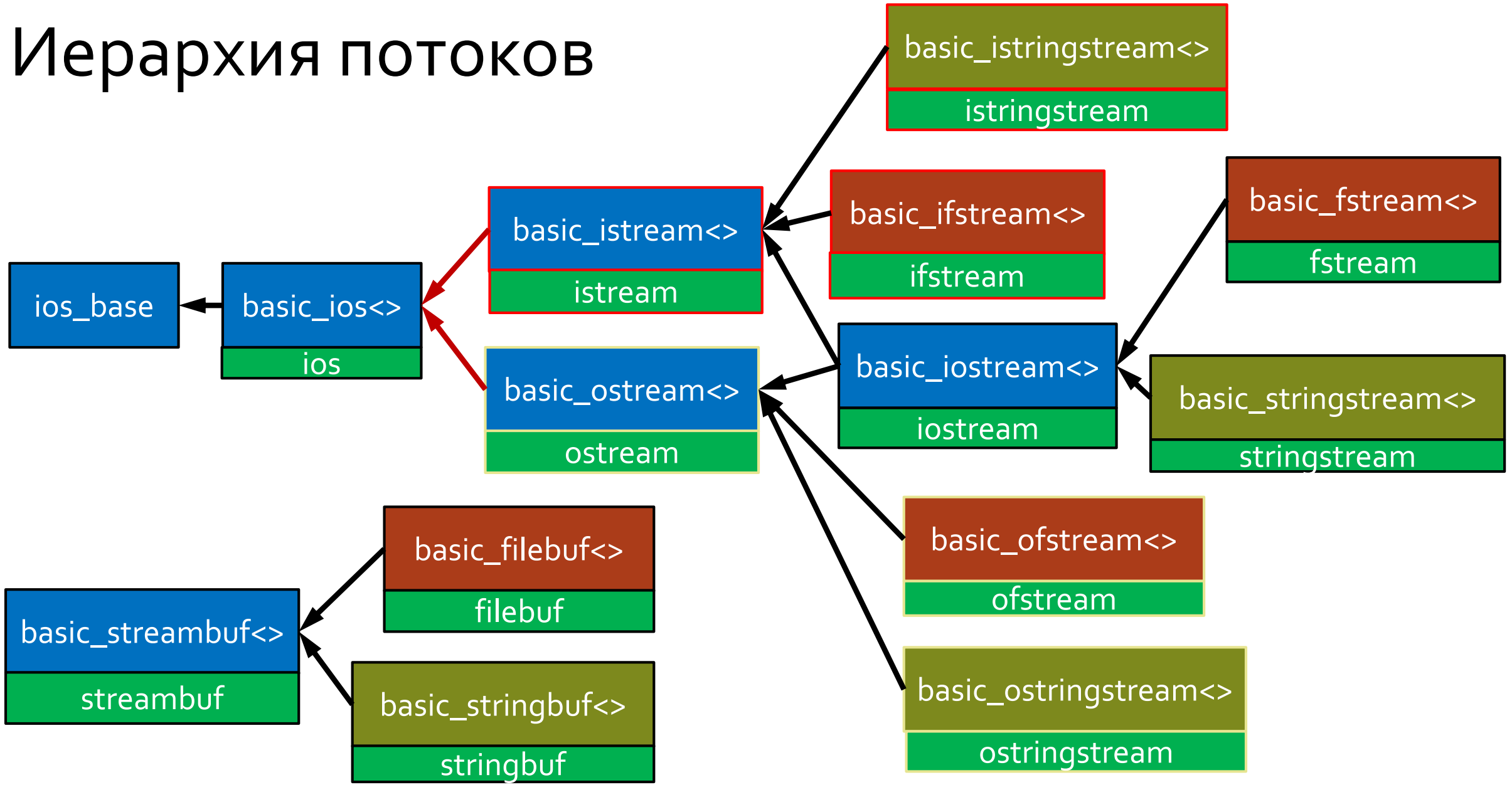
- Неименованный поток живет до конца полного выражения
- Строка будет побита по **разделителям**
- Установить свой разделитель можно, но это сложнее, чем кажется

- Ввод и вывод
- Файлы и строки
  - Буферизация
- Локализация
- Преобразования строк

Иерархия потоков



Иерархия потоков



Буферизация

- Буферизованные
  - Построчная буферизация
    - cout
    - cin
  - Полная буферизация
    - clog
- Не буферизованные
  - cerr

- Поток либо ассоциирован с буфером либо владеет им
- rdbuf() возвращает буфер потока
- rdbuf(basic_streambuf<>*) устанавливает буфер потока

1. while (file.get(c)) cout.put(c);
2. cout << file.rdbuf();

Ещё пример:

cout.rdbuf(nullptr);

Странности буферизации

- Казалось бы этот код ничего не должен печатать прежде чем выдать запрос

```
std::cout << "Please enter x: ";
std::cin >> x;
```

- Но он печатает

Странности буферизации

- Казалось бы этот код ничего не должен печатать прежде чем выдать запрос

```
std::cout << "Please enter x: ";
std::cin >> x;
```

- Но он печатает
- Причины к этому две:

1. Сцепленность с потоком cin
2. Связанность с stdio

Сцепленность потоков

```
std::cin.tie (nullptr); // расцепить со std::cin

std::cout.sync_with_stdio(false); // развязать со stdio

std::cout << "Please enter x: ";

std::cin >> x;
```

- Теперь поведение ожидаемое
- Сцепленность tie это сцепленность по побочным эффектам, но потоки можно связать и грубее

Связанность потоков

ostream hexout(std::cout.rdbuf()); // теперь у них один буфер

hexout.setf (std::ios::hex, std::ios::basefield);

hexout.setf (std::ios::showbase); // но разные форматы

std::cout << 42 << " ";

hexout << 42 << std::endl;

На экране: 42 2A

Перенаправление потоков

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

auto backup = std::cout.rdbuf();
auto psbuf = filestr.rdbuf();
cout.rdbuf(psbuf); // в этой точке поток cout перенаправлен в файл

cout << "This is written to the file" << endl;

cout.rdbuf(backup);
```

- Ввод и вывод
- Файлы и строки
- Буферизация
- Локализация
- Преобразования строк

Локали и фасеты

Стандартные фасеты и расширение

Пример: разделитель для cin

```
class my_ctype : public std::ctype<char> {
  mask my_table[table_size];
public:
  my_ctype(size_t refs = 0) : std::ctype<char>(&my_table[0], false, refs) {
    std::copy_n(classic_table(), table_size, my_table);
    my_table['-'] = (mask)space;
    my_table[':'] = (mask)space;
  }
};

std::string s1, s2, s3, s4;
std::istringstream input("Ann-Bob Carl:Debora");
std::locale x(std::locale::classic(), new my_ctype);
input.imbue(x);
input >> s1 >> s2 >> s3 >> s4;
```

Механизм iword и xalloc

Пример: модификатор add_one

```
inline int geti() { static int i = ios_base::xalloc(); return i; }

ostream& add_one(ostream& os) { os.iword(geti()) = 1; return os; }

ostream& add_none(ostream& os) { os.iword(geti()) = 0; return os; }

struct my_num_put : num_put<char> {
  iter_type do_put(iter_type s, ios_base& f, char_type fill, long v) const
    { return num_put<char>::do_put(s, f, fill, v + f.iword(geti())); }
  iter_type do_put(iter_type s, ios_base& f, char_type fill, unsigned long v) const
    { return num_put<char>::do_put(s, f, fill, v + f.iword(geti())); }
};

int main() {
  cout.imbue(locale(locale(),new my_num_put));
  cout << add_one << 10 << " " << 11 << "\n" << add_none << 10 << " " << 11 << endl;
}
```

Обсуждение

- Настоящее назначение локалей: интернационализация

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

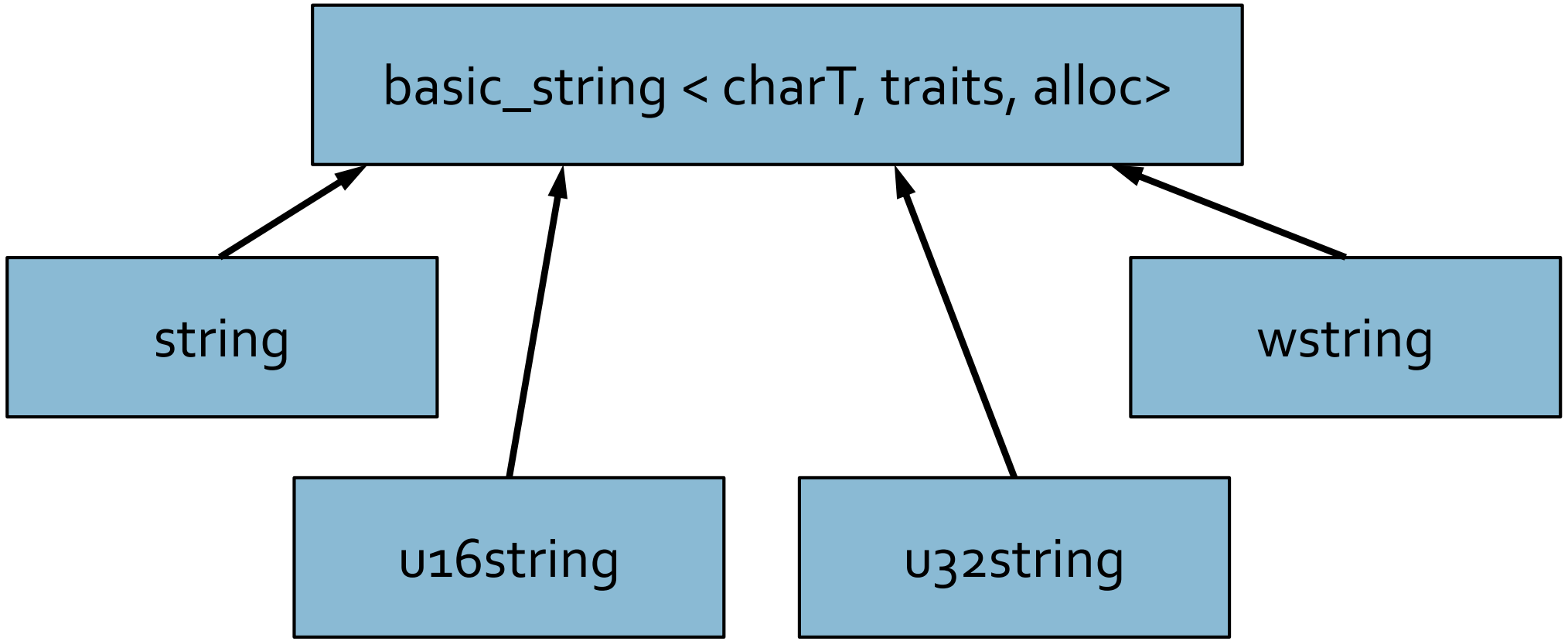
Локали для символов

- Ввод и вывод
- Файлы и строки
- Буферизация
- 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