ООП 2021
Лекция 9

Потоки данных.
Удаленные функции.
Локализация.

oopCpp@yandex.ru

Потоковые классы

Поток — это общее название потока данных. В C++ поток представляет собой объект некоторого класса. Разные потоки предназначены для представления разных видов данных. Например, класс **ifstream** олицетворяет собой поток данных от входного дискового файла.

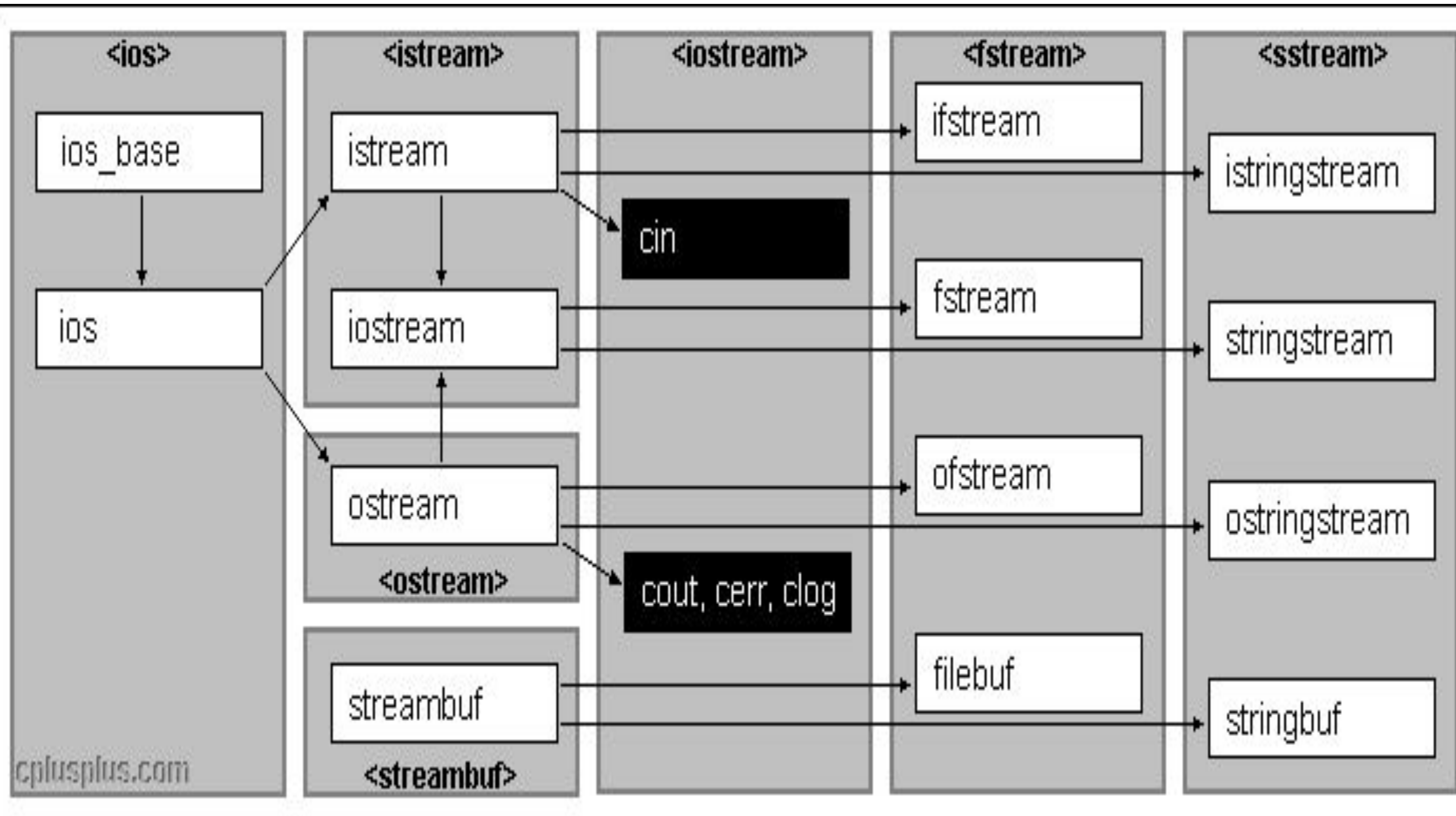
Одним из аргументов в пользу потоков является простота использования.

Если вам приходилось когда-нибудь использовать символ управления форматом %d при форматировании вывода с помощью %d в printf(), вы оцените это. Ничего подобного в потоках вы не встретите, ибо каждый объект **сам знает**, как он должен выглядеть на экране или в файле. Это избавляет программиста от одного из основных источников ошибок.

Другим арументом является то, что можно перегружать стандартные операторы и функции вставки (<<) и извлечения (>>) для работы с создаваемыми классами. Это позволяет работать с собственными классами как со стандартными типами, что, опять же, делает программирование проще и избавляет от множества ошибок, не говоря уж об эстетическом удовлетворении.

Поток данных - лучший способ записывать данные в файл, лучший способ организации данных в памяти для последующего использования при вводе/выводе текста в окошках и других элементах графического интерфейса пользователя (GUI)

Общая структура потоковых классов



ios_base

- event
- event_callback
- failure
- flags
- fmtflags
- getloc
- imbue
- Init
- ios_base
- iostate
- iword
- openmode
- operator=
- precision
- pword
- register_callback
- seekdir
- setf
- sync_with_stdio
- unsetf
- width
- xalloc

Чтобы сделать классы потоков **некопируемыми**, basic_ios в **C++98** объявлен следующим образом (Мейерс, Эффективный и современный C++):

```
template <class charT, class traits = char_traits<charT> >
class basic_ios : public ios_base {
public:
private:
basic_ios( const basic_ios& );                    //Не определен
basic_ios& operator=( const basic_ios&);   //Не определен
};
```

В **C++ 11** имеется лучший способ достичь по сути того же самого: воспользоваться конструкцией «delete» : чтобы пометить копирующий конструктор и копирующее присваивание как **удаленные функции**:

```
template <class charT, class traits = char_traits<charT> >
class basic_ios : public  ios_base {
public:
basic_ios(const basic_ios& ) = delete;
basic_ios& operator=(const basic_ios&) = delete;
};
```

<u>Удаленные функции</u> не могут использоваться никоим образом, так что даже код функции-члена или функций, объявленных как **friend**, **<u>не будет компилироваться</u>**, если попытается копировать объекты basic_ios.

Это существенное улучшение по сравнению с поведением C++98, где такое некорректное применение функций не диагностируется до компоновки.

По соглашению удаленные функции объявляются как **public**, а не **private**. Тому есть причина. Когда код клиента пытается использовать функцию-член, C++ проверяет доступность до проверки состояния удаленности. Когда клиентский код пытается использовать функцию, объявленную как private, некоторые компиляторы жалуются на то, что это закрытая функция, несмотря на то что доступность функции никак не влияет на возможность ее использования. Стоит принять это во внимание при пересмотре старого кода и замене не определенных функций-членов, объявленных как private, удаленными, поскольку объявление удаленных функций как pubic в общем случае приводит к более корректным сообщениям об ошибках.

Важным преимуществом удаленных функций является то, что удаленной может быть **<u>любая функция,</u>** в то время как быть private могут только функции-члены.

```
bool isLucky( int number) ;
if ( isLucky ( 'a') )  // Является ли 'a' счастливым числом?
if (isLucky (true) )   // Является ли true счастливым числом?
if (isLucky(3.5) )  // Следует ли выполнить усечение до 3  перед проверкой ?
bool isLucky (char) = delete;    // Отвергаем символ
bool isLucky(bool) = delete;     // Отвергаем булевы значения
bool isLucky (double) = delete;  // Отвергаем double и float
```

Еще один трюк, который могут выполнять **удаленные функции** (а функции-члены, объявленные как private нет), заключается в предотвращении использования инстанцирований шаблонов, которые **должны быть запрещены**. Предположим, например, что нам нужен шаблон, который работает со встроенными указателями:

```
template<typename T>
void processPointer(T* ptr) ;
```

В мире указателей есть два особых случая. Один из них указатели **void***, поскольку их нельзя разыменовывать, увеличивать или уменьшать и т.д. Второй указатели **char***, поскольку они обычно представляют указатели на C-строки, а не на отдельные символы.

Эти особые случаи часто требуют особой обработки; будем считать, что в случае шаблона processPointer эта особая обработка заключается в том, чтобы отвергнуть вызовы с такими типами (т.е. **должно быть невозможно вызвать** processPointer с указателями типа void* или char*).

Это легко сделать. Достаточно удалить эти инстанцирования:

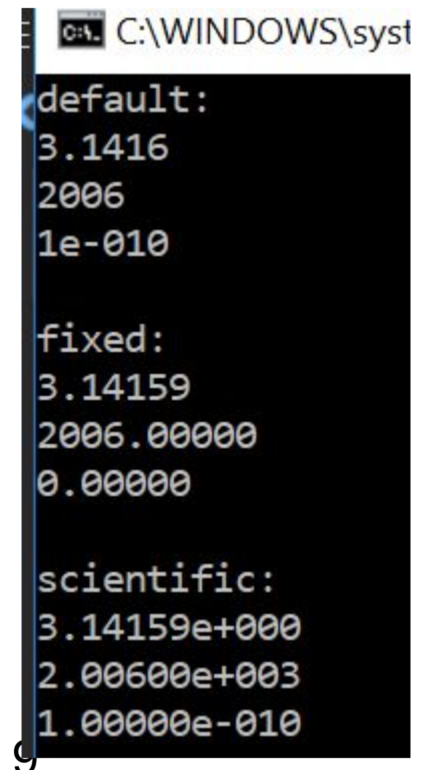
```
template<> void processPointer<void>(void*) = delete;
template<> void processPointer<char>(char*) = delete;
```

Флаги форматирования класса ios

skipws Пропуск пробелов при вводе
Left Выравнивание по левому краю
right Выравнивание по правому краю
internal Заполнение между знаком или основанием числа и самим числом
dec Перевод в десятичную форму
oct Перевод в восьмеричную форму
hex Перевод в шестнадцатеричную форму
boolalpha Перевод логического 0 и 1 соответственно в «false» и «true»
showbase Выводить индикатор основания системы счисления (0 для восьмеричной, 0x для шестнадцатеричной)
showpoint Показывать десятичную точку при выводе
uppercase Переводить в верхний регистр буквы X, E и буквы шестнадцатеричной системы счисления (ABCDEF) (по умолчанию — в нижнем регистре)
showpos Показывать «+» перед положительными целыми числами
scientific Экспоненциальный вывод чисел с плавающей запятой
fixed Фиксированный вывод чисел с плавающей запятой
unitbuf Сброс потоков после вставки
stdio сброс stdout, sterror после вставки

Пример

```
#include <iostream>
int main () {
  double a = 3.1415926534;
  double b = 2006.0;
  double c = 1.0e-10;
std::cout.precision (5);
std::cout << "default:\n";
  std::cout << a << '\n' << b << '\n' << c << '\n';
std::cout << '\n';
std::cout << "fixed:\n" << std::fixed;
  std::cout << a << '\n' << b << '\n' << c << '\n';

  std::cout << '\n';

  std::cout << "scientific:\n" << std::scientific;
  std::cout << a << '\n' << b << '\n' << c << '\n';
  return 0;
}
```

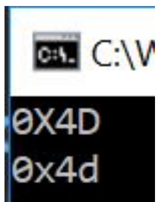


Пример

```
#include <iostream>
#include <sstream>

int main () {
  char a, b, c;

  std::istringstream iss ("  123");
  iss >> std::skipws >> a >> b >> c;
  std::cout << a << b << c << '\n';

  iss.seekg(0);
  iss >> std::noskipws >> a >> b >> c;
  std::cout << a << b << c << '\n';
  return 0;
}
```

Пример

```
#include <iostream>

int main () {
  std::cout << std::showbase << std::hex;
  std::cout << std::uppercase << 77 << '\n';
  std::cout << std::nouppercase << 77 << '\n';
  return 0;
}
```



Пример

```
#include <iostream>
#include <fstream>
int main () {
std::ofstream ofs;
  ofs.open ("test.txt", std::ofstream::out | std::ofstream::app);
  ofs << " more lorem ipsum";
  ofs.close();
std::ifstream ifs;
  ifs.open ("test.txt", std::ifstream::in);
  char c = ifs.get();
  while (ifs.good()) {
    std::cout << c;
    c = ifs.get();
  }
  ifs.close();
  return 0;
}
```

Пример

```
#include <fstream>
#include <cstdio>       // EOF – это знак конца файла: end of file
int main () {
  std::ifstream ifs ("test.txt");
  std::ofstream ofs ("copy.txt");
  std::filebuf* inbuf  = ifs.rdbuf();
  std::filebuf* outbuf = ofs.rdbuf();

  char c = inbuf->sbumpc();
  while (c != EOF)
  {
    outbuf->sputc (c);
    c = inbuf->sbumpc();
  }
  ofs.close();
  ifs.close();
 return 0;
}
```

Есть несколько способов установки флагов форматирования, для каждого свои. Так как они являются компонентами класса ios, обычно к ним обращаются посредством написания имени класса и оператора явного задания

(например, ios::skipws).

Все без исключения флаги могут быть выставлены с помощью методов setf() и unsetf().

cout.setf(ios::left); // выравнивание текста по левому краю

cout << "Этот текст выровнен по левому краю"

cout.unsetf (ios::left); // вернуться к прежнему форматированию

Многие флаги могут быть установлены с помощью манипуляторов:

ws Включает пропуск пробелов при вводе

dec Перевод в десятичную форму

oct Перевод в восьмеричную форму

hex Перевод в шестнадцатеричную форму

endl Вставка разделителя строк и очистка выходного потока

ends Вставка символа отсутствия информации для окончания выходной строки

flush Очистка выходного потока

lock Закрыть дескриптор файла

unlock Открыть дескриптор файла .

Манипуляторы с аргументами

setw ( ) ширина поля (int) Устанавливает ширину поля для вывода данных

setfill () символ заполнения (int) Устанавливает символ заполнения (по умолчанию, пробел)

setprecision() точность (int) Устанавливает точность (число выводимых знаков)

setiosflags ( )Флаги форматирования (long)Устанавливает указанные флаги форматирования

resetiosflags ( ) Флаги форматирования (long)Сбрасывает указанные флаги форматирования

Функции класса ios

ch = fill (); Возвращает символ заполнения (символ, которым заполняется неиспользуемая часть текстового поля; по умолчанию — пробел)

fill(ch); Устанавливает символ заполнения

p = precision(); Возвращает значение точности (число выводимых знаков для формата с плавающей запятой)

precision (p); Устанавливает точность p

w = width(); Возвращает текущее значение ширины поля (в символах)

width(w); Устанавливает ширину текущего поля

setf(flags); Устанавливает флаг форматирования (например, ios::left)

unsetf(flags); Сбрасывает указанный флаг форматирования

setf(flags, field); Очищает поле, затем устанавливает флаги форматирования

Класс istream

`>>`	Форматированное извлечение данных всех основных (и перегружаемых) типов из потока
`get(ch)`	Извлекает один символ в ch
`get(str)`	Извлекает символы в массив str до ограничителя '\n'
`get(str, MAX)`	Извлекает до MAX числа символов в массив
`get(str, DELIM)`	Извлекает символы в массив str до указанного ограничителя (обычно '\n'). Оставляет ограничитель в потоке
`get(str, MAX, DELIM)`	Извлекает в массив str до MAX символов или до символа DELIM. Оставляет ограничитель в потоке
`getline(str,MAX, DELIM)`	Извлекает в массив str до MAX символов или символа DELIM. Извлекает ограничитель из потока
`putback(ch)`	Вставляет последний прочитанный символ обратно во входной поток
`ignore(MAX,DELIM)`	Извлекает и удаляет до MAX числа символов до ограничителя включительно (обычно '\n'). С извлеченными данными ничего не делает
`peek(ch)`	Читает один символ, оставляя его в потоке
`count = gcount()`	Возвращает число символов, прочитанных только что встретившимися вызовами get(), getline() или read()
`read(str, MAX)`	(Для файлов.) Извлекает вплоть до MAX числа символов в массив str
`seekg()`	Устанавливает расстояние (в байтах) от начала файла до файлового указателя
`seekg(pos,seek_dir)`	Устанавливает расстояние (в байтах) от указанной позиции в файле до указателя файла. seek_dir может принимать значения ios::beg, ios::cur, ios::end
`pos = tellg(pos)`	Возвращает позицию (в байтах) указателя файла от начала файла

Класс ostream

<<	Форматированная вставка данных любых стандартных (и перегруженных) типов
put(ch)	Вставка символа ch в поток
flush()	Очистка буфера и вставка разделителя строк
write(str, SIZE)	Вставка SIZE символов из массива str в файл
seekp(position)	Устанавливает позицию в байтах файлового указателя относительно начала файла
seekp(position, seek_dir)	Устанавливает позицию в байтах файлового указателя относительно указанного места в файле, seek_dir может принимать значения ios::beg, ios::cur, ios::end
pos = tellp()	Возвращает позицию указателя файла в байтах

Состояние потока

```
static const _Iostate goodbit =   (_Iostate)0x0;
static const _Iostate eofbit =    (_Iostate)0x1;
static const _Iostate failbit =   (_Iostate)0x2;
static const _Iostate badbit =    (_Iostate)0x4;
static const _Iostate _Hardfail = (_Iostate)0x10;
```

```
#include <iostream>     // std::cout, std::ios
#include <sstream>      // std::stringstream
void print_state (const std::ios& stream) {
  std::cout << " good()=" << stream.good();
  std::cout << " eof()=" << stream.eof();
  std::cout << " fail()=" << stream.fail();
  std::cout << " bad()=" << stream.bad();
}

int main () {
  std::stringstream stream;
  stream.clear (stream.goodbit);
  std::cout << "goodbit:"; print_state(stream); std::cout << '\n';
  stream.clear (stream.eofbit);
  std::cout << " eofbit:"; print_state(stream); std::cout << '\n';
  stream.clear (stream.failbit);
  std::cout << "failbit:"; print_state(stream); std::cout << '\n';
  stream.clear (stream.badbit);
  std::cout << " badbit:"; print_state(stream); std::cout << '\n';
   return 0;   }
```

C:\WINDOWS\system32\cmd.exe

```
goodbit: good()=1 eof()=0 fail()=0 bad()=0
 eofbit: good()=0 eof()=1 fail()=0 bad()=0
failbit: good()=0 eof()=0 fail()=1 bad()=0
 badbit: good()=0 eof()=0 fail()=1 bad()=1
```

Файлы

Обычно мы имеем намного больше данных, чем способна вместить основная память нашего компьютера, поэтому большая часть информации хранится на дисках или других средствах хранения данных высокой емкости. Такие устройства также предотвращают исчезновение данных при выключении компьютера — такие данные являются персистентными.

На самом нижнем уровне файл просто представляет собой последовательность байтов, пронумерованных начиная с нуля.

Файл имеет формат; иначе говоря, набор правил, определяющих смысл байтов. Например, если файл является текстовым, то первые четыре байта представляют собой первые четыре символа. С другой стороны, если файл хранит бинарное представление целых чисел, то первые четыре байта используются для бинарного представления первого целого числа. Формат по отношению к файлам на диске играет ту же роль, что и типы по отношению к объектам в основной памяти. Мы можем приписать битам, записанным в файле, определенный смысл тогда и только тогда, когда известен его формат. При работе с файлами поток ostream преобразует объекты, хранящиеся в основной памяти, в потоки байтов и записывает их на диск. Поток istream действует наоборот: он считывает поток байтов с диска и составляет из них объект.

Работа с файлами

Для того чтобы **прочитать** файл, мы должны

- знать его имя;
- открыть его (для чтения);
- считать символы;
- закрыть файл (хотя это обычно выполняется неявно).

Для того чтобы **записать** файл, мы должны

- назвать его;
- открыть файл (для записи) или создать новый файл с таким именем;
- записать наши объекты;
- закрыть файл (хотя это обычно выполняется неявно).

Пример

```
#include <iostream>
#include <fstream>

int main () {
  std::ifstream is("example.txt");   // open file
    char c;
  while (is.get(c))                  // loop getting single characters
    std::cout << c;

  if ( is.eof() )                    // check for EOF
    std::cout << "[EoF reached]\n";
  else
    std::cout << "[error reading]\n";
  is.close();                        // close file
  return 0;
}
```

Буфер потока

Буфер потока - это объект, ответственный за выполнение операций чтения и записи объекта потока, с которым он связан: поток делегирует все такие операции связанному с ним объекту буфера потока, который является посредником между потоком и его управляемым входом и выходом последовательности.

Все объекты потока, независимо от того, буферизированы или не буферизованы, имеют связанный поток буфера: некоторые типы буфера потока могут затем быть настроены либо на использование промежуточного буфера, либо нет.

Объекты буфера потока сохраняются внутри, по крайней мере:

Локальный объект, используемый для операций, зависящих от языка.

Набор внутренних указателей для хранения входного буфера: eback, gptr, egptr.

Набор внутренних указателей для хранения выходного буфера: pbase, pptr, epptr.

Пример

```
#include <iostream>
#include <fstream>

int main () {
  char ch;
  std::ofstream ostr ("test.txt");
  if (ostr) {
    std::cout << "Writing to file. Type a dot (.) to end.\n";
    std::streambuf * pbuf = ostr.rdbuf();
    do {
      ch = std::cin.get();
      pbuf->sputc(ch);
    } while (ch!='.');
    ostr.close();
  }

  return 0;
}
```

Пример

```
#include <iostream>
#include <fstream>

int main () {
std::ofstream ofs;
  ofs.open ("test.txt", std::ofstream::out | std::ofstream::app);
  ofs << " more lorem ipsum";
  ofs.close();
  std::ifstream is;
  std::filebuf * fb = is.rdbuf();

  fb->open ("test.txt",std::ios::in);

  std::cout<<fb;
  fb->close();

  return 0;
}
```

stringbuf

default (1)
explicit basic_stringbuf (ios_base::openmode which = ios_base::in | ios_base::out);

initialization (2)
explicit basic_stringbuf (const basic_string<char_type,traits_type,allocator_type>& str,
ios_base::openmode which = ios_base::in | ios_base::out);

copy (3)
basic_stringbuf (const basic_stringbuf&) = **delete**;

move (4)
basic_stringbuf (basic_stringbuf&& x);

Пример

```
#include <iostream>     // std::cout, std::ostream, std::hex
#include <sstream>      // std::stringbuf
#include <string>       // std::string

int main () {
  std::stringbuf   buffer;      // empty stringbuf

  std::ostream os (&buffer);  // associate stream buffer to stream
      // mixing output to buffer with inserting to associated stream:
  buffer.sputn ("255 in hexadecimal: ",20);
  os << std::hex << 255;

  std::cout << buffer.str();
  return 0;
}
```

Output:
255 in hexadecimal: ff

istream::operator>>

Для чтения данных из потоков. Перегрузок много:

```
istream& operator>> (bool& val);
istream& operator>> (short& val);
istream& operator>> (unsigned short& val);
istream& operator>> (int& val);
istream& operator>> (unsigned int& val);
istream& operator>> (long& val);
istream& operator>> (unsigned long& val);
istream& operator>> (long long& val);
istream& operator>> (unsigned long long& val);
istream& operator>> (float& val);
istream& operator>> (double& val);
istream& operator>> (long double& val);
istream& operator>> (void*& val);
istream& operator>> (streambuf* sb );
istream& operator>> (istream& (*pf)(istream&));
istream& operator>> (ios& (*pf)(ios&));
istream& operator>> (ios_base& (*pf)(ios_base&));
```

Пример

```
int main () {
  int n;

  std::cout << "Enter a number: ";
  std::cin >> n;
  std::cout << "You have entered: " << n << '\n';

  std::cout << "Enter a hexadecimal number: ";
  std::cin >> std::hex >> n;          // manipulator
  std::cout << "Its decimal equivalent is: " << n << '\n';

  return 0;
}
```

basic_istream:: get

Извлекает символы из потока в качестве неформатированного ввода:

```
int_type get();
basic_istream& get (char_type& c);
basic_istream& get (char_type* s, streamsize n);
basic_istream& get (char_type* s, streamsize n, char_type delim);
basic_istream& get (basic_streambuf<char_type, traits_type>& sb);
basic_istream& get (basic_streambuf<char_type, traits_type>& sb, char_type delim);
```

Пример

```
#include <iostream>     // std::cin, std::cout
#include <fstream>      // std::ifstream
int main () {
  char str[256];
  std::cout << "Enter the name of an existing text file: ";
  std::cin.get (str,256);    // get c-string
  std::ifstream is(str);     // open file
  char c;
  while (is.get(c))          // loop getting single characters
    std::cout << c;
  is.close();                // close file
  return 0;
}
```

В этом примере запрашивается имя существующего текстового файла и выводится его содержимое на экран, используя cin.get как отдельные символы, так и c-строки.

basic_istream:: getline

Извлекает символы из потока в виде неформатированных входных данных и сохраняет их в виде c-строки до тех пор, пока либо извлеченный символ не станет тем delimiting character, либо n символов не будут записаны (включая завершающий нулевой символ).

```
int_type get();
basic_istream& get (char_type& c);
basic_istream& get (char_type* s, streamsize n);
basic_istream& get (char_type* s, streamsize n, char_type delim);
basic_istream& get (basic_streambuf<char_type, traits_type>& sb);
basic_istream& get (basic_streambuf<char_type, traits_type>& sb, char_type delim);
```

Пример

```
#include <iostream>
int main () {
  char name[256], title[256];
  std::cout << "Please, enter your name: ";
  std::cin.getline (name,256);
  std::cout << "Please, enter your favourite movie: ";
  std::cin.getline (title,256);
  std::cout << name << "'s favourite movie is " << title;
  return 0;
}
```

Этот пример иллюстрирует, как получить строки из стандартного входного потока (cin).

basic_ostream::operator<<

Этот оператор (<<), применяемый к выходному потоку, известен как оператор вставки: basic_ostream& operator<< (bool val);

basic_ostream& operator<< (short val);

basic_ostream& operator<< (unsigned short val);

basic_ostream& operator<< (int val);

basic_ostream& operator<< (unsigned int val);

basic_ostream& operator<< (long val);

basic_ostream& operator<< (unsigned long val);

basic_ostream& operator<< (long long val);

basic_ostream& operator<< (unsigned long long val);

basic_ostream& operator<< (float val);

basic_ostream& operator<< (double val);

basic_ostream& operator<< (long double val);

basic_ostream& operator<< (void* val);

basic_ostream& operator<< (basic_streambuf<char_type,traits_type>* sb);

basic_ostream& operator<< (basic_ostream& (*pf)(basic_ostream&));

basic_ostream& operator<< (basic_ios<char_type,traits_type>& (*pf)(basic_ios<char_type,traits_type>&));

basic_ostream& operator<< (ios_base& (*pf)(ios_base&));

Пример

```
#include <iostream>     // std::cout, std::right, std::endl
#include <iomanip>      // std::setw

int main () {
  int val = 65;

  std::cout << std::right;          // right-adjusted (manipulator)
  std::cout << std::setw(10);    // set width (extended manipulator)

  std::cout << val << std::endl; // multiple insertions

  return 0;
}
```

Output: 65

Пример с файлом

```
#include <fstream>

int main () {

  std::fstream fs;
  fs.open ("test.txt", std::fstream::in | std::fstream::out | std::fstream::app);

  fs << " test ";

  fs.close();

  return 0;
}
```

Локализация

По книге Д. Р. Стефенс, К. Диггинс, Д. Турканис, Д. Когсуэлл «C++. Сборник рецептов» (2007)

Обеспечение возможности работы программы в различных регионах (локализация), как правило, требует решения двух задач: форматирования строк, воспринимаемых пользователем, в соответствии с местными соглашениями (например, даты, времени, денежных сумм и чисел) и обеспечения работы с различными символьными наборами.

Большая часть программного обеспечения будет работать в странах, отличных от той, где они были написаны. Для поддержки этой практики стандартная библиотека STL имеет несколько средств, способствующих написанию программного кода, предназначенного для работы в различных странах. Однако они спроектированы не так, как многие другие средства стандартной библиотеки, например строки, файловый ввод-вывод, контейнеры, алгоритмы и т. п. Например, класс, представляющий локализацию, имеет имя locale и содержится в заголовочном файле <locale>. Класс locale предоставляет средства для записи и чтения потоков с применением специфичного для данной местности форматирования и получения таких сведений о локализации, как, например, ее символ валюты или формат даты. Однако стандартом предусматривается обеспечение только одной локализации, и этой локализацией является C-локализация, или классическая локализация.

Классическая локализация использует стандарт ANSI C: принятые в американском варианте английского языка соглашения по форматированию и 7-битовой код ASCII. И от реализации зависит, будут ли обеспечены экземпляры locale для других языков и регионов.

Заголовочный файл <locale> имеет три основные части.

Во-первых, это класс locale (локализация). Он инкапсулирует все поддерживаемые в C++ особенности локализованного поведения и обеспечивает точки входа для получения различной информации о локализации, необходимой для выполнения локализованного форматирования.

Во-вторых, самыми маленькими элементами локализации и конкретными классами, с которыми вы будете работать, являются классы, называемые фасетами (facets). Примером фасета является, например, класс time_put, предназначенный для записи даты в поток.

В-третьих, каждый фасет принадлежит к некоторой категории, которая объединяет связанные фасеты в одну группу. Например, имеются числовая, временная и денежная категории.

Каждая программа на C++ имеет, по крайней мере, одну локализацию, называемую глобальной локализацией. По умолчанию это будет классическая локализация C, пока вы не измените ее на что-нибудь другое. Один из конструкторов locale позволяет инстанцировать локализацию, предпочитаемую пользователем, хотя точное определение предпочитаемой пользователем локализации полностью зависит от реализации.

Unicode

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void main( ) {
// Создать несколько строк с символами кода Unicode
wstring ws1 = L"Infinity: \u221E";
wstring ws2 = L"Euro. _";
wchar_t  w[ ] = L"Infinty. \u22lE";
        wofstream out("unicode.txt");
out << ws2 << endl;
wcout << ws2 << endl;
}
```

Основной вопрос, возникающий при кодировании строк в Unicode- коде, связан с выбором способа ввода строки в редакторе исходных текстов. В С++ предусмотрен тип расширенного набора символов wchar_t, который может хранить строки в коде Unicode. Точное представление wchar_t зависит от реализации, однако часто используется формат UTF-32 (или UTF-16). Класс wstring определяется в <string> как последовательность символов типа wchar_t, подобно тому как класс string представляет собой последовательность символов типа char.

При работе с различными кодировками наибольшую ловкость приходится проявлять не для ввода правильных символов в исходные файлы, а при определении типа символьных данных, получаемых из базы данных, по запросу HTTP, из пользовательского ввода и т. д., что выходит за рамки стандарта C++.

Стандарт C++ не устанавливает никаких специальных требований, кроме того, что операционная система может использовать для исходных файлов любую кодировку, если она поддерживает, по крайней мере, 96 символов, используемых в языке C++. Для символов, не попадающих в этот набор, называемый *основным исходным набором символов*, стандартом предусматривается возможность их получения с помощью escape-последовательностей: \uXXXX или \UXXXXXXXX. где X – шестнадцатеричная цифра.

Запись и чтение чисел

Пусть требуется записать число в поток в форматированном виде в соответствии с местными соглашениями.

Закрепите (imbue) текущую локализацию за потоком, в который вы собираетесь писать данные, и запишите в него числа. Или можете установить глобальную локализацию и затем создать поток.

```
#include <iostream>
#include <locale>
#include <string>
using namespace std;
// На заднем плане существует глобальная локализация,
// установленная средой. По умолчанию это локализация "C".
// можно ее заменить локализацией locale ::global( const locale &).
int main( ) {
        locale loc(""); // Создать копию пользовательской локализации
cout  << "Locale name =  " <<  loc.name( ) <<  endl;
cout.imbue(loc); // Уведомить cout о необходимости применения
                 // пользовательской локализации при форматировании
cout << "pi in locale " << cout.getloc( ).name( ) <<   " is  3.14 " << endl;}
```

Вывод:

Locale name = Russian_Russia.1251

pi in locale Russian_Russia.1251 is 3.14

Стандартные фасеты

```
regex xpr  ("мир", regex_constants::icase); // icase - Регулярные выражения
                                            // совпадают без учета регистра.
 smatch match;   //  сравнение символов
 string str("ПРИВЕТ МИР!");
 if(regex_search(str, match, xpr))
   cout << "icase ok" << endl;  // выпадет - не найдено - Почему?
 else
   cout << "icase fail" << endl;
```

По тексту: https://habr.com/ru/post/104417/

```
std::locale cp1251_locale("ru"); // добавляем глобально русификацию
   std::locale::global(cp1251_locale);

 regex xpr  ("мир", regex_constants::icase); // icase - Регулярные выражения
                                               // должны совпадать  без учета регистра.
  smatch match;
  string str("ПРИВЕТ МИР!");
  if(regex_search(str, match, xpr))
    cout << "icase ok" << endl;
  else
    cout << "icase fail" << endl; // выпадет – строки совпадают, что и ожидали
```

Домашнее задание на неделю

Проект 32.

- Создать абстрактный базовый класс **именем своей фамилии**, записанной латиницей. И поместить туда функцию print()=0;
- Создать 2 производных класса с типов измененного имени базового класса с суффиксами типа «_1» или «child_1» и «_2» . Первый наследуется от базового, а второй – от первого.
- В каждый производный класс поместить по одному члену данных типа **string*** . В конструкторах (пользовательских) инициировать, если возможно, их значениями, передаваемыми извне, либо (в конструкторах по умолчанию) – **именами типов** объектов.
- Создать отдельный класс DB, путем наследования его от класса list<Base*>.
- Организовать два глобальных хранилища типа DB.
- В главной функции в первое хранилище добавить несколько объектов созданных типов, а затем скопировать данные типа child_2 из первого хранилища во второе и распечатать данные второго хранилища при помощи print.