Взаимодействие программы с окружением

Курс «Технология программирования»

Кафедра управления и информатики НИУ «МЭИ»

Осенний семестр 2015 г.

Обработка ошибок: код возврата

```
int convert_temperature(
    double temperature,
    char from, char to,
    double& result)
{
    if (from != 'K' && from != 'C')
        return 1;
    //…
}
```

Код	Ошибка
0	Нет ошибки.
1	Неизвестная шкала from.
2	Неизвестная шкала to.
3	temperature < 0 °K

Лишняя переменная — повод ошибиться.

(Аналог cout для сообщений об ошибках.)

```
double kelvins;
switch (convert_temperature(celsius, 'C', 'K', kelvins)) {
    case 0:  cout << kelvins << "K\n"; break;
    case 1:  cerr  << "Неизвестная исходная шкала!\n"; break;
    default: cerr  << "Неизвестная ошибка!\n";
}
```

Обработка ошибок: доступ к последней ошибке

```
int last_error = 0;

int get_last_error() { return last_error; }

double convert_temperature(
    double temperature, char from, char to)
{
    if (from != 'K' && from != 'C') {
        last_error = 1;
        return 0.0;
    }
    // …
}

double kelvins = convert_temperature(celsius, 'C', 'K');

switch (get_last_error()) { … }
```

Глобальная переменная. Объявлена вне функций, доступна в любой из них.

Возвращаемое при ошибке значение не имеет смысла. Использовать его

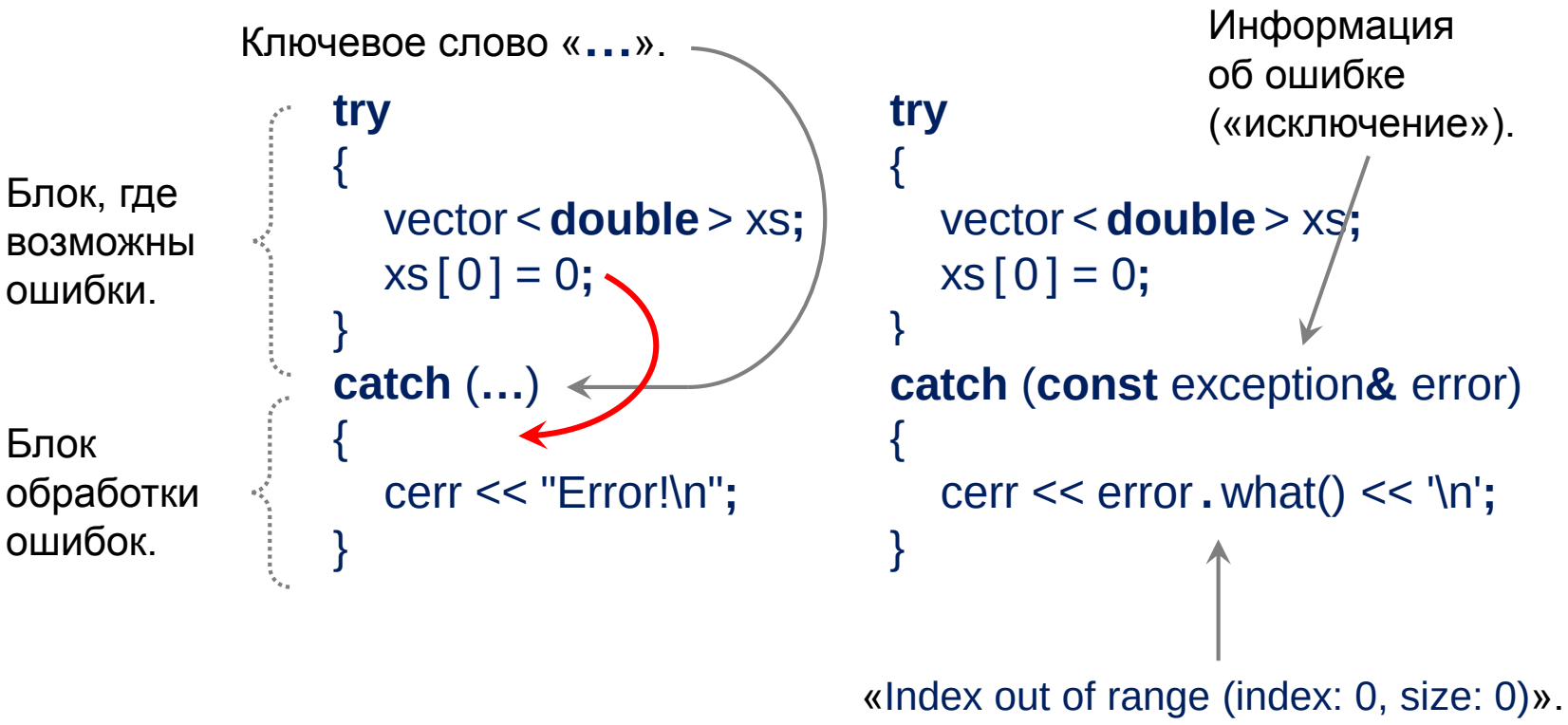
1) возможно, но это некорректно.

2) Проверка кода нужна, но необязательна.

Проблемы обработки ошибок

- Что плохо (challenges)?
  - Проверку ошибок легко забыть.
  - Обработка ошибок загромождает реализацию алгоритма.
  - Лавинообразный рост сложности обработки ошибок при вложенных вызовах.
  - Причину ошибки трудно уточнить.
- Чего хочется (ideals)?
  - Ошибка должна требовать обработки.
    - Или ошибка обработана, или программа завершается.
  - Сосредоточить код проверки в одной точке.
  - Автоматический переход к обработке при ошибках.
  - Доступ к информации о причинах ошибки.

Исключения (exceptions)



- ✓ Обработка ошибок в одном блоке.
- ✓ Автоматический переход к обработке.
- ✓ Доступна информация о причинах ошибки.

Распространение исключений

```
try
{
    cout << volume(1, −2, 3);
}
catch (const exception& e)
{
    cout << e.what();
}
```

```
double area (
    double width, double height )
{
    if (width < 0)     error ( "area: width < 0" );
    if (height < 0)    error ( "area: height < 0" );
    return width * height;
}
```

Где **catch**?

```
double volume(
    double width, double height, double depth )
{
    if (depth < 0) error ( "volume: depth < 0" );
    return  area ( width, height ) * depth;
}
```

- ✓ Обработка ошибок остается простой даже в сложной программе.

Возбуждение, перехват и типы исключений

- error("сообщение об ошибке");
  - Определена в sdt.h в учебных целях.
  - Добавляет к сообщению имя файла и номер строки.
- **throw** исключение(*аргументы, обычно сообщение*);
  - invalid_argument("width < 0");
    - Длина меньше нуля и т. п.
  - logic_error("логическая ошибка");
    - Корреляция между векторами неравной длины и т. п.
- Блоков **catch** может быть несколько.
- Блок **catch** (…) { } перехватывает всё, что было **throw**.
  - Ошибки, подобные разыменованию **nullptr**, перехватить нельзя.

Иерархия исключений

- exception
  - logic_error — Ошибка в логике работы.
    - invalid_argument — Некорректный аргумент функции.
    - out_of_range — Недопустимый индекс.
  - runtime_error — Технический сбой.
    - system_error — Ошибка в системной функции.
- У exception и ниже есть what(). Полный список в [cppref].

```
try { kelvins = convert( celsius[i], 'C', 'K' ); }
catch (const out_of_range& oor) { … }
catch (const logic_error& le)   { … }
catch (const exception& e)      { … }
catch (…)                       { … }
```

Порядок блоков — от частных случаев к общим. Выполняется первый подходящий, даже если ниже есть соответствие точнее.

Исключения и конструкторы

```
Temperature t;   // теперь неверно!
```

```
struct Temperature {
    double value;
    Scale scale;
    Temperature(
        double value, char symbol);
};

Temperature::Temperature(
    double value, char symbol)
{
    if (symbol != 'C' && symbol != 'c')
        error("Неизвестная шкала " + symbol);
    scale = Celsius;
    this->value = value;
}
```

```
try
{
    double value; char symbol;
    cin >> value >> symbol;
    Temperature t{ value, symbol };
}
catch (const exception& e)
{
    cerr << e.what() << '\n';
}
```

Указатель на текущую структуру (которая инициализируется).

Перегрузка операторов

```
Temperature boiling { 100, 'C' };

if (data > boiling) { … }

Temperature mean { 0, 'C' };
mean = mean + data;
```

<, += — отдельные операторы!

```
Temperature operator + (
    const Temperature& lhs,
    const Temperature& rhs)
{
    return {
        lhs.value + rhs.value,
        lhs.scale
    };
}
```

Предполагается одинаковая шкала для краткости!

Особое имя функции.

```
bool operator > (
    const Temperature& lhs,
    const Temperature& rhs)
{
    return lhs.value > rhs.value;
}
```

Типы результата и параметров должны быть точно такими.

«Left-Hand Side» и «Right-Hand Side»

Вывод пользовательских типов

сокращенная форма записи

```
cout << x;          →  operator<<(cout, x);
cout << x << y;        operator<<(operator<<(cout, x), y);
1 + 2 + 3              (1 + 2) + 3
```

тип cout* →

```
ostream& operator<<(
    ostream& output, const Temperature& data)
{
    output << data.value << data.scale;
    return output;
}
```

```
cout << data;  // 237.15K
```

* На самом деле, нет.

Ввод пользовательских типов

тип cin*

Ссылка не **const**, так как data изменяется.

```
istream& operator>>(
    istream& input, Temperature& data)
{
    input >> data.value >> data.scale;
    if (data.scale != 'K' || data.value < 0) {
        input.setstate(ios_base::failbit);
    }
    return input;
}
```

Здесь можно выполнить преобразования и проверку ввода.

Как сообщить об ошибке?..

А чтобы это работало?

- `Temperature data;`
  `while (cin >> data) { … }`
-
```
if (!(cin >> data)) {
    cout << "Incorrect temperature input!";
}
```

Окончание ввода cin отследит.

Файловый ввод и вывод

Или:

```
ifstream input;
input.open("file.txt");
```

- Ничего или ios::out.
- ios_base::ate — дописывать в конец,
  - Append (Output) в Pascal;
- ios_base::trunc — очистить файл перед записью,
  - Truncate (Output) в Pascal.

- Ввод:

```
ifstream input("file.txt");
input >> temperature;
```

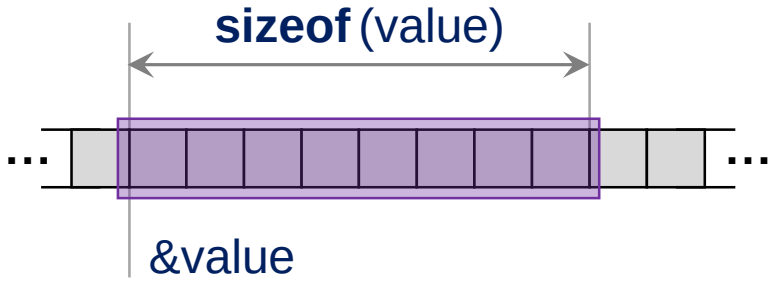
- Вывод:

```
ofstream output("file.txt", режим);
output << "Result: " << result << '\n';
```

- Закрытие файла — автоматически.
- Работа с файлами подобна работе с cin и cout.

Двоичные файлы

- Хранят данные так, как они представлены в памяти, — в виде набора байт.
- Компактная запись рисунков и т. п.
- Удобны для чтения машиной.
- Режим: *режим* | ios_base :: binary
  (для ввода: ios_base :: binary).



- Вывод:

```
double value = 42;
output.write( as_bytes(value), sizeof(value) );
```

- Ввод:

```
input.read( as_bytes(value), sizeof(value) );
```

- Векторы и строки можно (пока) вводить и выводить поэлементно.

Форматный вывод

double value = 12.34567;

- cout << setprecision(2);

Действует на cout все время после установки.

- cout << value // 12
  << fixed << value // 12.35
  << scientific << value // 1.23e+01
  << defaultfloat << value; // 12

…значащих цифр

…цифр после запятой

…цифр после запятой в мантиссе

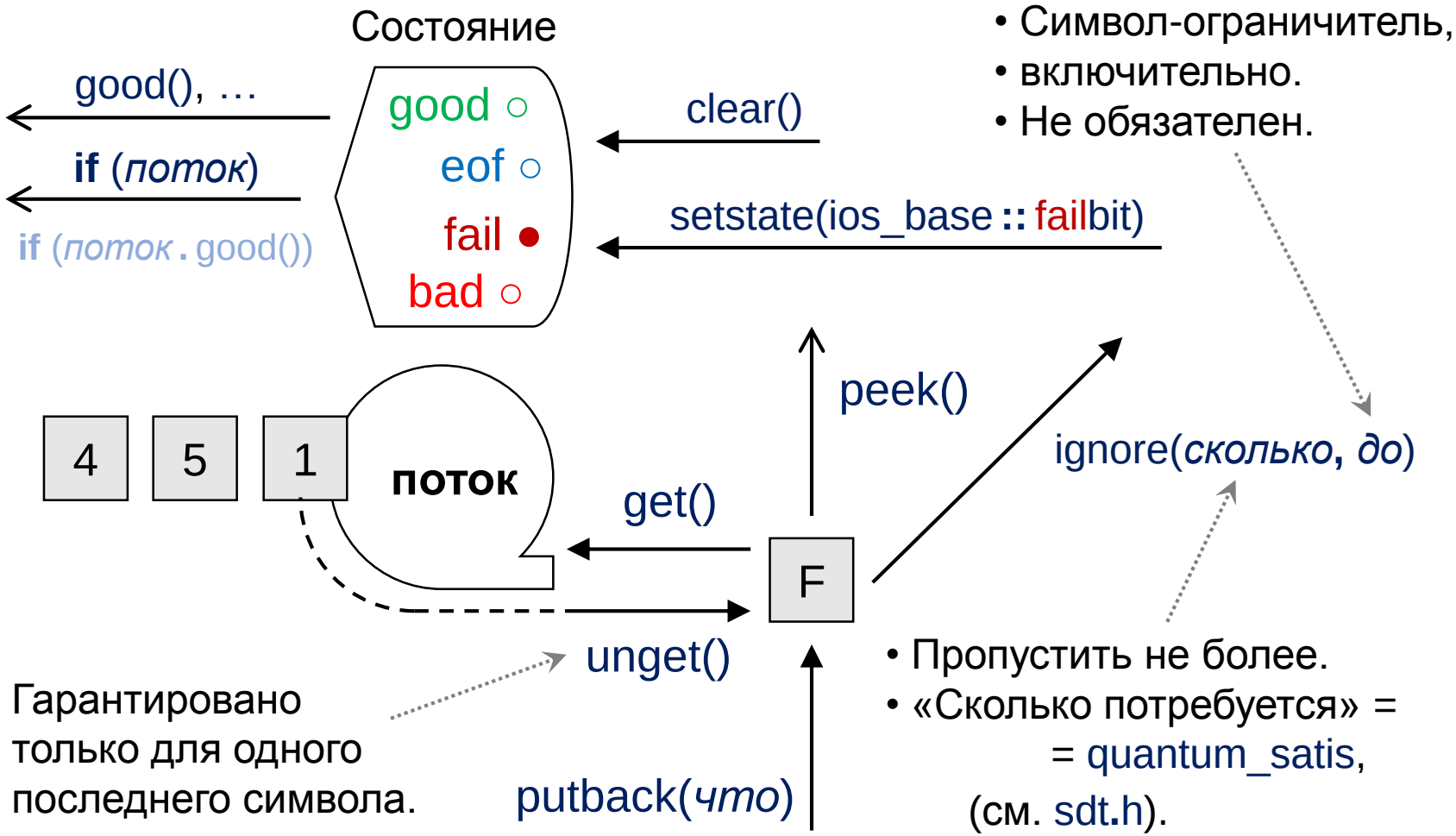
Действуют только на следующее выводимое значение.

- Функция printf("%03.3f \n", value); // 012.346
  - Форматная строка подобна таковой в MATLAB.
  - Не работает с vector<T>.
  - Не рекомендуется в C++.

Форматный вывод (продолжение)

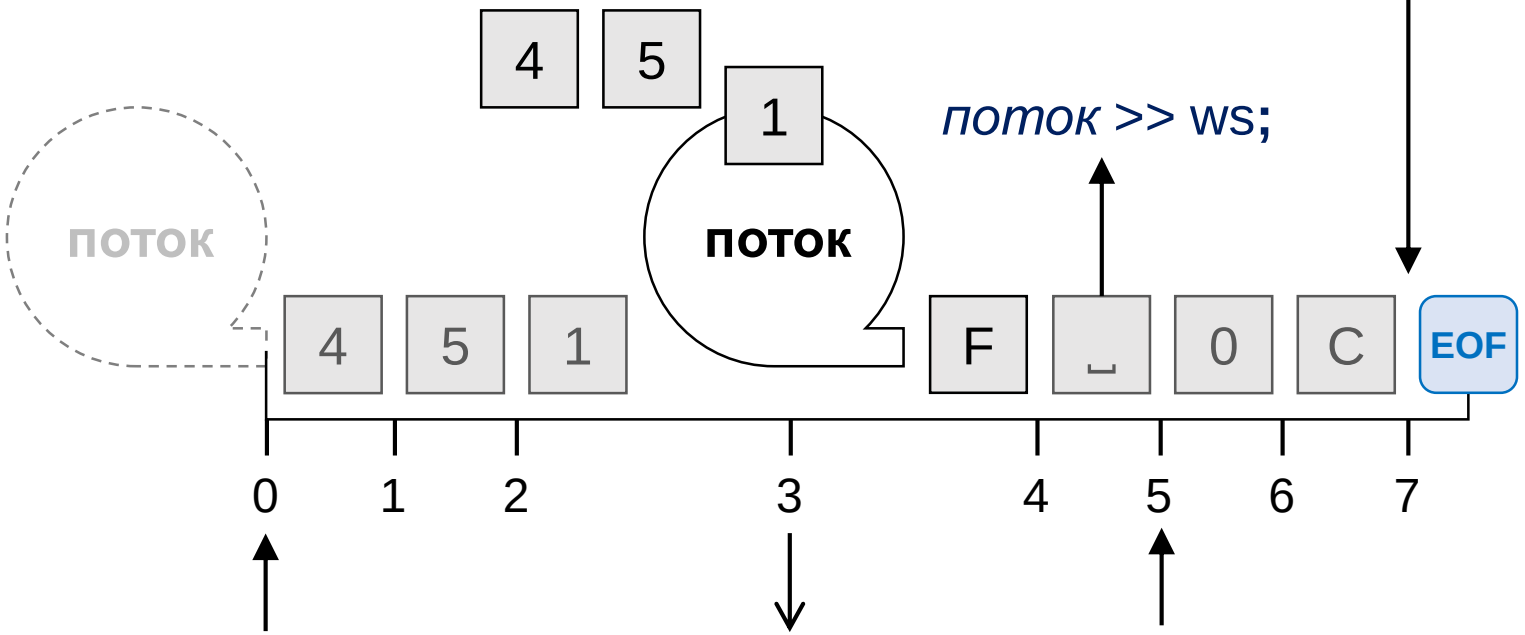
- cout << setw(15) << setfill('.') << left << "Code:"
  << setw(4) << setfill('0') << right << 12;
- `Code:.........0012`
- cout << endl;
  - cout << '\n';
    cout.flush();

Потоковый ввод



Перемещение в потоке

Для вывода: tell**p**(), seek**p**().



seek**g**(ios_base::end, 0)

поток >> ws;

seek**g**(ios_base::beg, 0)

seek**g**(0)

tell**g**()

seek**g**(ios_base::cur, 2)

seek**g**(tell**g**() + 2)

Потоки в памяти

- **string** input**;**
  getline(cin**,** input)**;**

  stringstream source(input)**;**

  size_t count = 0**;**
  for (**string** word**;** source >> word**;** ++count)**;**
  cout << "Word count: " << count << '\n'**;**

Из строки вычитываются слова, разделенные пробелами.

Тело цикла пустое:
++count и есть подсчет слов.

- **int** parse (**string** text) {
    stringstream stream (text)**;**
    int result**;**
    stream >> result**;**
    return result**;**
  }

- stringstream — Чтение и запись.
  - istringstream — Только чтение.
  - ostringstream — Только запись.
- Работает и с двоичными данными.

Функции printf() и scanf()

- Форматная строка аналогична MATLAB.
  - Но только для скалярных значений.
  - Корректность не проверяется.
- Функция printf() не работает со **string** напрямую:

```
string hello = "hello";
printf("%s", hello.c_str());
```

- Подробная справка по форматной строке: [cppref].
- **float** value;
  scanf("%f", &value);
  - Некорректный вызов приведет к порче памяти!

Работа с файлами в C

Тип-указатель (не разыменовывается), обозначающий открытый файл.

```
FILE* file = fopen("name.ext", "r");
fscanf(file, "%d", &number);
string line(120, '\0');
fgets(line.data(), line.size(), file);
fprintf(file, "%4.2f", 3.14);
fputs(line.c_str(), file);
fseek(file, 0, SEEK_END);
long position = ftell(file);
fclose(file);
```

Режим доступа [cppref]:
"r" — чтение текста;
"w" — перезапись файла;
"a" — дозапись в конец.

1. Строка из 120 символов '\0'.
2. Это специальный символ «конец строки».

Перемещение на 0 символов от конца, то есть к концу файла. См. также [cppref].

Параметры командной строки

- ```
  #include "sdt.h"
  int main(int argc, char* argv[])
  {
      for (size_t i = 0; i < argc; ++i)
          cout << "argv[" << i << "] = " << argv[i] << '\n';
  }
  ```
- C:\> *program.exe 1 a "b c"*
  argv[0] = C:\program.exe
  argv[1] = 1
  argv[2] = a
  argv[3] = b c

argv[0] содержит
имя файла программы.

Параметры с пробелами
нужно передавать в кавычках.

- Можно задать параметры запуска проекта C::B.
  ```

Препроцессор

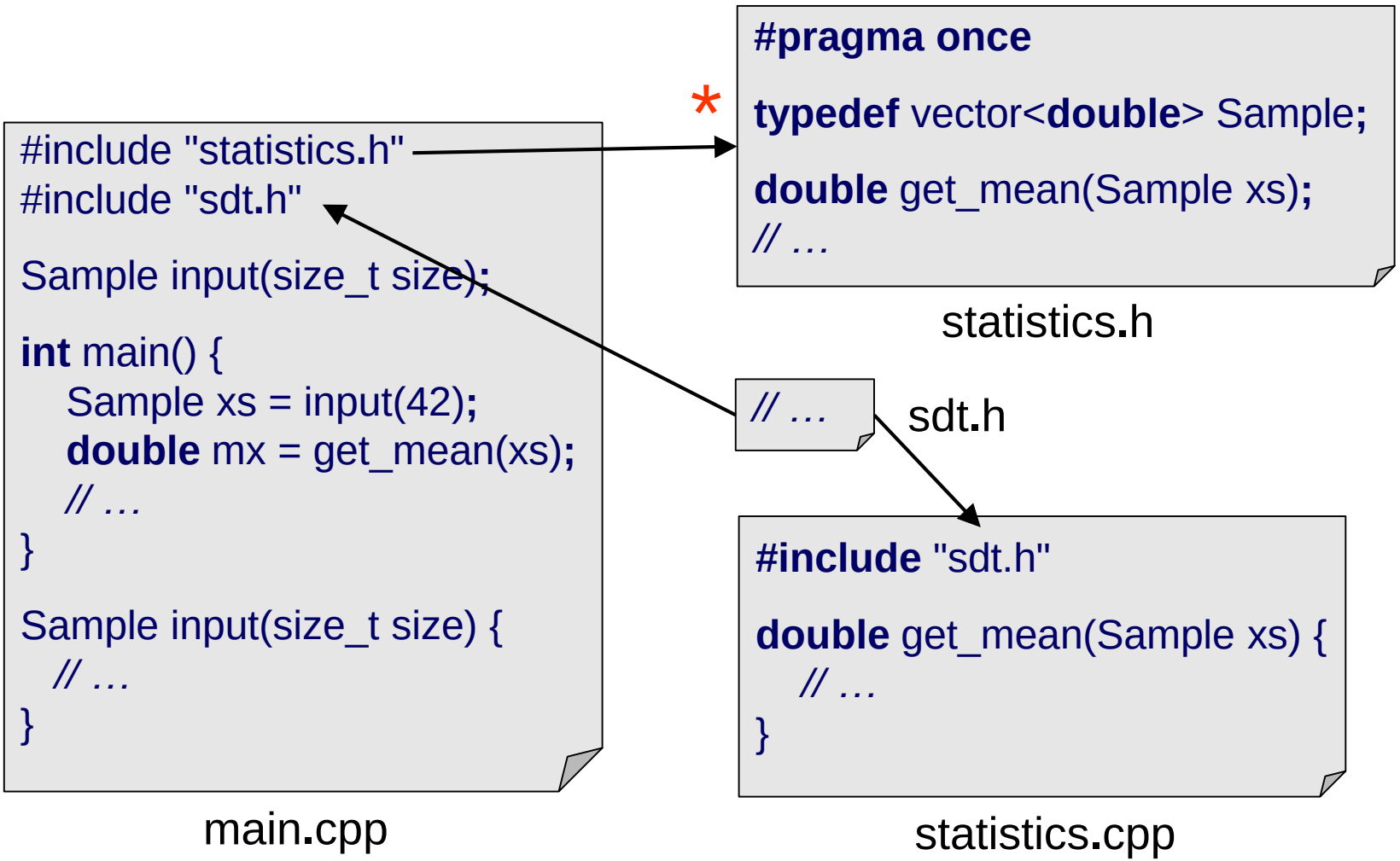
- Работает до компиляции программы над её текстом.
- Включение текстов одних файлов в другие.
  - **#include** "путь/к/файлу.h"
  - **#include** <файл.h>
- Макросы: замена текста
  - Не рекомендуется использовать, т. к. меняют текст программы.
  - простая

```
#define PI 3.14159265358979
cout << PI;      // cout << 3.14159265358979
```

  - с подстановкой

```
#define mean(a, b)     (0.5 * ((a) + (b)))
cout << mean(3, 4);    // cout << (0.5 * ((3) + (4)))
```

Раздельная компиляция



Раздельная компиляция

- Исходный текст распределяется по файлам.
  - Заголовочные файлы (headers, *.h)
    - содержат объявления функций и типов данных;
    - не содержат (не должны содержать) реализаций;
    - включаются в файлы реализации через **#include**,
      - один раз в один файл с помощью **#pragma once**.
  - Файлы реализации (sources, *.cpp)
    - могут содержать и объявления, и определения;
    - компилируются по отдельности;
    - не должны включаться друг в друга,
      - иначе получится, что у функции несколько реализаций (исходная и включенная).

* **typedef** A B вводит новое имя B для типа A.

Ресурсы и литература

- *Online-справочник C++ Reference* [cppref].
- *Programming Principles and Practices Using C++:*
  - глава 5 — обработка ошибок, исключения;
  - глава 9, раздел 9.6 — перегрузка операторов;
  - глава 10 — ввод и вывод, в том числе файловый;
  - глава 11 — форматный ввод и вывод;
  - упражнения к главам 5, 10, 11.
- *C++ Primer:*
  - раздел 5.6 — использование исключений.
  - глава 8 — ввод и вывод;
  - глава 14 — перегрузка операторов, в т. ч. ввода и вывода.