022

POLYTECH
NICE-SOPHIA

Cours Programmation C++

A pragmatic approach to modern C++ programming

Introduction

Course Syllabus 2019 (1)

- Lectures & Labs (24 hours)
- Exam (2 hours)
- Grade:
  - Lab: two evaluated reports
  - Project: one "tiny" project
  - Exam: final examination
  - Final score: weighted average of the 3 grades.
    - 1/3 lab
    - 1/3 project
    - 1/3 exam

Course Syllabus (2)

- 2020

#	Week	Date	Time	Where	Hours (Cumul)
1	06	Friday, 7-Feb-2020	8:30 – 12:00	E+134	3:30 (3:30)
2	07	Friday, 14-Feb-2020	8:30 – 12:00	E+134	3:30 (7:00)
3	08	Friday, 21-Feb-2020	8:30 – 12:00	E+133	3:30 (10:30)
	09	Friday, 28-Feb-2020			
4	10	Friday, 6-Mar-2020	8:30 – 12:00	E+133	3:30 (14:00)
5	11	Friday, 13-Mar-2020	8:30 – 12:00	E+145	3:30 (17:30)
6	12	Friday, 20-Mar-2020	8:30 – 12:00	E+145	3:30 (21:00)
7	13	Friday, 27-Mar-2020	8:30 – 12:00	E+145	3:30 (24:30)
8	14	Friday, 3-Apr-2020	8:30 – 10:30	E+145	2:00 (26:30)

Course Objectives

- *Write* C++ programs of low/medium complexity what will pass modern code review.
- *Use* common tool chains to compile, debug and test your program
- *Understand* how the compiler works and the mechanisms behind memory allocation and de-allocation of your objects.
- *Select* C++ appropriate datatypes (built-in, library based) as key enablers of efficient solutions to your project.
- *Build* your knowledge on common design patterns (GoF)
- *Explore* available documentation to write more complex programs.

Pre-requisite

- I assume that you are all familiar with
  - C programming language
    - basic types, statements, structure, pointers
  - Common data-structure and algorithms
    - array, list, binary tree, hash-table
    - quick-sort, binary search
  - Computer Architecture
    - binary and hexadecimal notation
    - CPU registers, notion of assembly programming
    - memory hierarchy, heap, stack
  - Linux or Windows Environment
    - Linux shell, Visual Studio, Eclipse, GCC, clang, ...

Introduction (1)

- What is C++?
- C++ is a

1. general-purpose (as opposed to special purpose)
2. multi-paradigm (can do functional, can do OO)
3. strongly-typed, (compile time static type checking)
4. value-semantic, (pass by value, reference all possible)
5. systems-level ***language*** (manage all computer resources)
6. with lexical scoping, (easy to bind variable to object)
7. deterministic destruction, (as opposed to garbage collection)
8. and a single-pass compiler (compiled language)

Credit: Charles Bay CppCon 2019 (link)

Introduction (2)

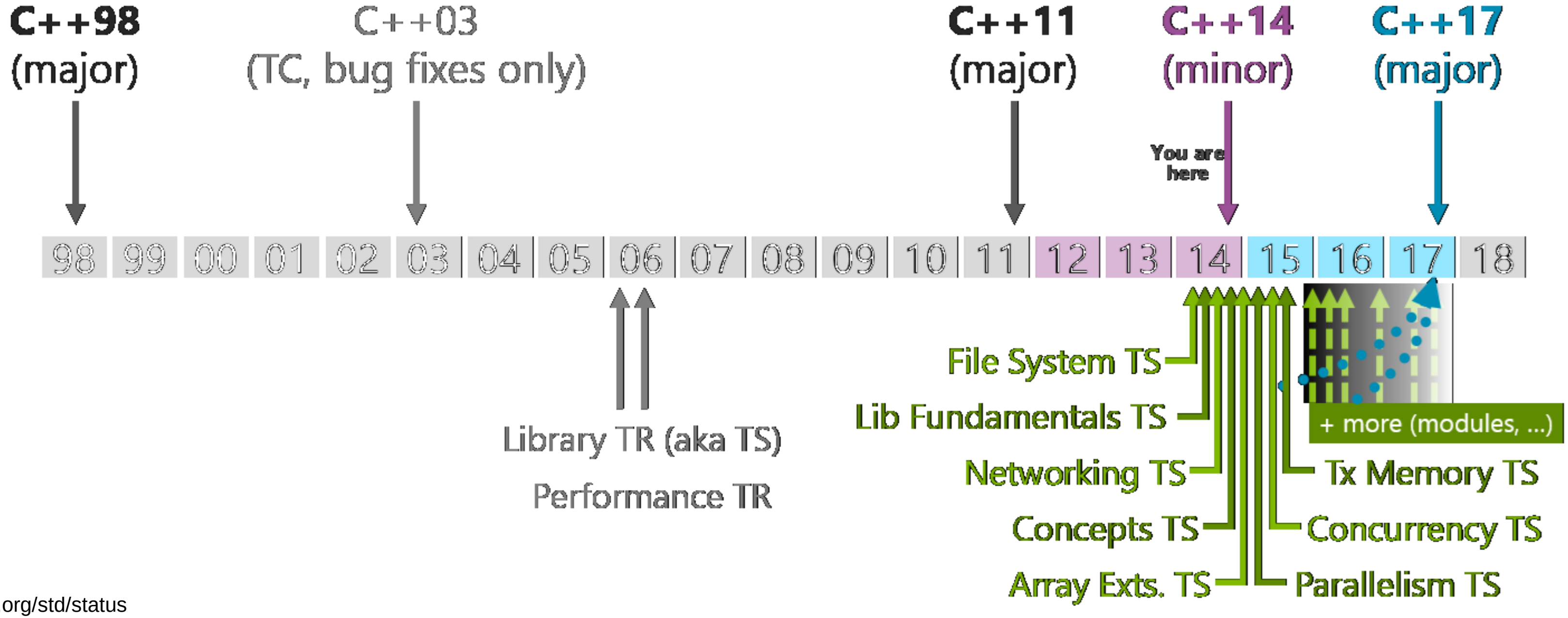
- Why C++
  - Efficient code (may not be optimal code).
  - Give good control on memory layout

***Efficient**: performing in the best possible manner with the least waste or time and effort.*

- Focus on commonly used modern C++
  - Use defaults, basic styles and idioms
  - Write for clarity and correctness first
  - Avoid the “complexity addiction”

Introduction (3)

- We shall cover C++ 11/14 and associated STL (standard template library)



Source: http://isocpp.org/std/status

Introduction (4)

- Is C++ Too Complicated?
- But how does one measure “complexity”?
- Pages in the language specification?
  - C++98 Language: 309 pp.
  - C++11 Language: 424 pp.
  - Java SE 7 Edition: 606 pp.
  - C# ECMA Standard 4th Ed. June ‘06: 531 pp.

Source: An Overview of C++11 and C++14 - Leor Zolman - CppCon 2014b

References [1]

- C++ FAQs, Tutorials and Courses
  - http://en.wikipedia.org/wiki/C%2B%2B
  - http://www.parashift.com/c%2B%2B-faq-lite
  - http://www.labri.fr/perso/nrougier/teaching/c++-crash-course/index.html
  - http://www.tutorialspoint.com/cplusplus/index.htm
  - https://github.com/cppcon/cppcon2015 also 2016, ... 2019
  - https://en.wikibooks.org/wiki/C%2B%2B_Programming
- Books
  - The C++ Programming Language, from Bjarne Stroustrup (2013)
  - The C++ Standard Library: A Tutorial and Reference, from Nicolai M. Josuttis (2012)

References [2]

- Language Reference
  - http://en.cppreference.com
  - https://isocpp.org/faq
- YouTube
  - https://www.youtube.com/user/CppCon

First Concepts

Assignment #1 (1)

- Write a short program in C or C++ which reads a sequence of numbers (double precision) from a file and compute the *mean* and *median* value. File size is unknown, you can not read the file twice.

data.txt

```
2615.93
863.93
1990.52
2815.77
1181.31
1321.13
455.36
812.47
2638.90
17301.72
```

```
shell> mean_and_median data.txt
number of elements = 10, median = 1655.83, mean
= 3199.70
```

Assignment #1 (2)

- Phase #1: Start with high level, plain English statements:

```
Open file “data.txt”
For each line of the file do {
  ...


}
...
```

Assignment #1 (3)

- Phase #2: Refine your statements, introducing variables:

```
fp = Open file “data.txt” in read mode
line = A buffer of character
while !(at the end of fp) {
  store char of fp  in line, stop at \n
  ...

}
...
```

Assignment #1 (4-1)

Phase #3: Refine your statements: map to existing C library functions, you will need all these functions:

```
FILE *fopen(const char *path, const char *mode);
char *fgets(char *buffer, int size, FILE *fin);
double strtod(const char *nptr, char **endptr);
void qsort(void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
void *malloc( size_t size );
void *realloc( void *ptr, size_t new_size );
```

Formulas for cumulative mean

$$CMA_n = \frac{x_1 + \cdots + x_n}{n}\,. \qquad CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1} = CMA_n + \frac{x_{n+1} - CMA_n}{n+1}$$

Assignment #1 (4-2)

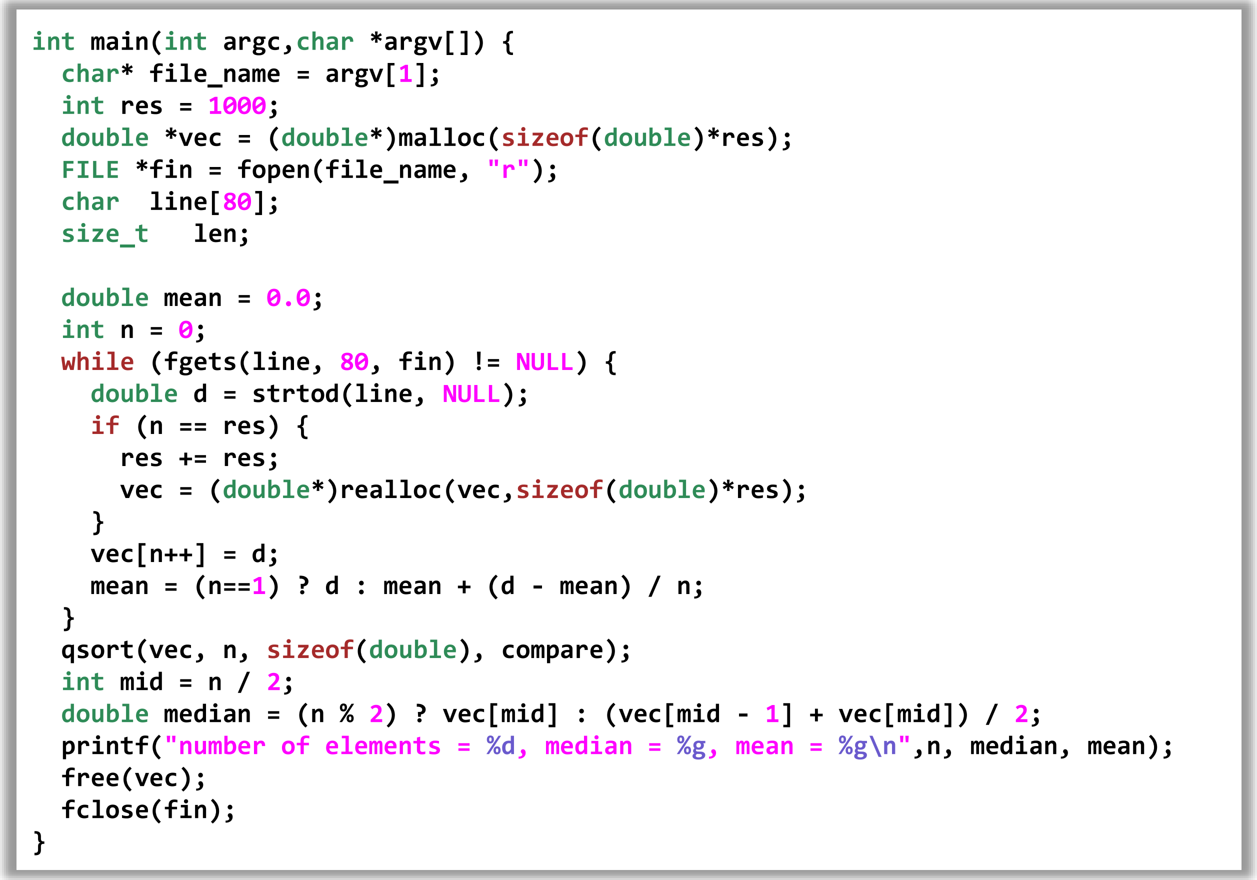
Phase #3: Refine your statements: map to existing C++ library functions, you will need all these methods:

```
std::ifstream(const char *path, ios_base::openmode mode);
std::getline(std::ifstream &fp, std::string &str);
double std::stod(const std::string &str);
void std::sort(iterator &first, iterator &last);
void std::vector::push_back(const T& value);
```

Formulas for cumulative mean

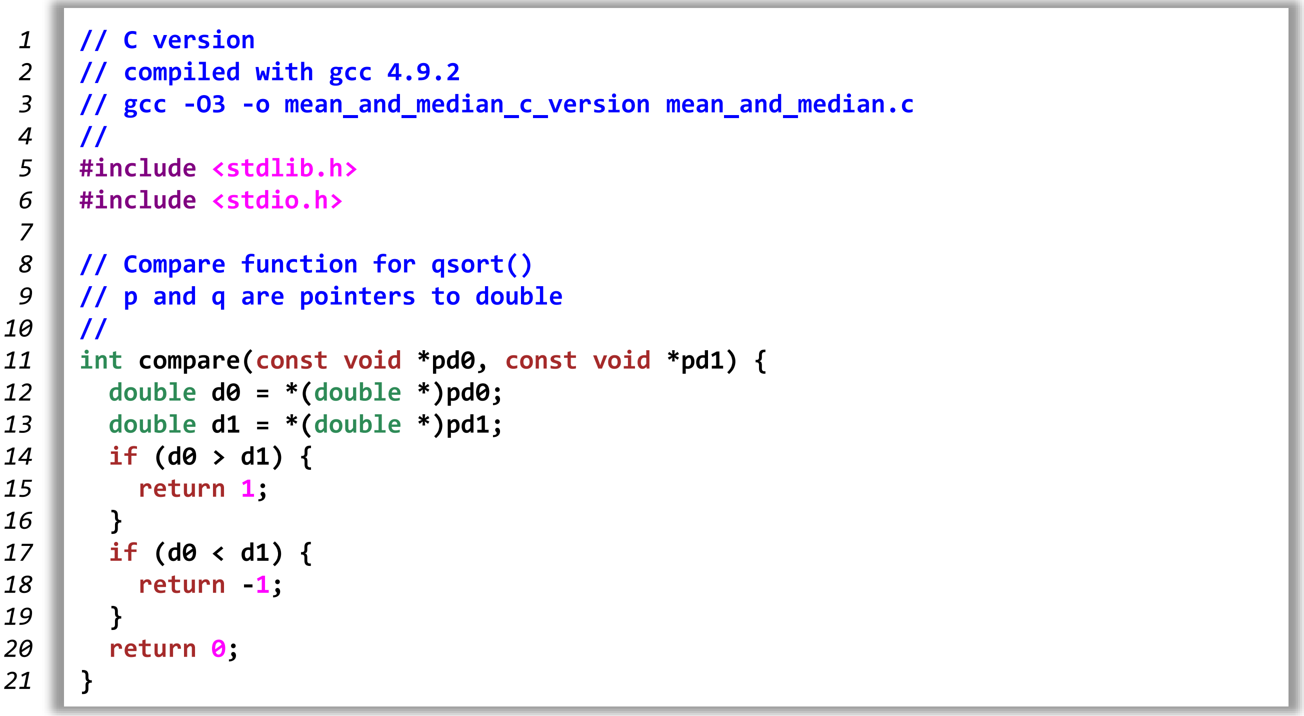
$$CMA_n = \frac{x_1 + \cdots + x_n}{n} \, . \qquad CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1} = CMA_n + \frac{x_{n+1} - CMA_n}{n+1}$$

Implemented in C (1)



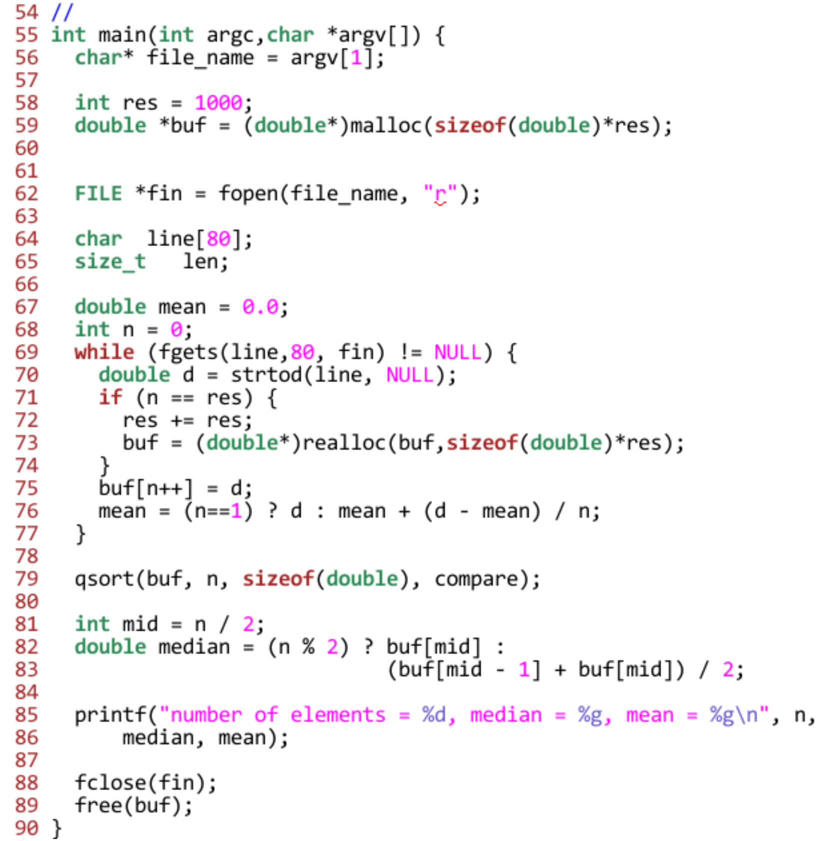
```c
int main(int argc,char *argv[]) {
  char* file_name = argv[1];
  int res = 1000;
  double *vec = (double*)malloc(sizeof(double)*res);
  FILE *fin = fopen(file_name, "r");
  char  line[80];
  size_t   len;

  double mean = 0.0;
  int n = 0;
  while (fgets(line, 80, fin) != NULL) {
    double d = strtod(line, NULL);
    if (n == res) {
      res += res;
      vec = (double*)realloc(vec,sizeof(double)*res);
    }
    vec[n++] = d;
    mean = (n==1) ? d : mean + (d - mean) / n;
  }
  qsort(vec, n, sizeof(double), compare);
  int mid = n / 2;
  double median = (n % 2) ? vec[mid] : (vec[mid - 1] + vec[mid]) / 2;
  printf("number of elements = %d, median = %g, mean = %g\n",n, median, mean);
  free(vec);
  fclose(fin);
}
```

Implemented in C (2)

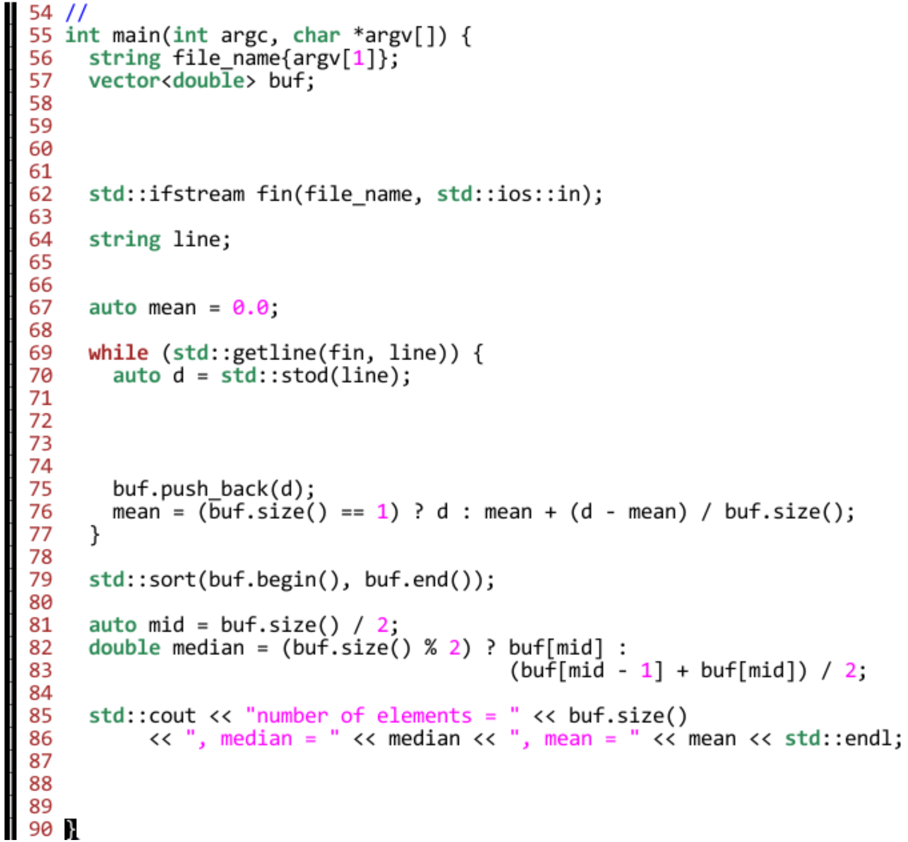


```c
// C version
// compiled with gcc 4.9.2
// gcc -O3 -o mean_and_median_c_version mean_and_median.c
//
#include <stdlib.h>
#include <stdio.h>

// Compare function for qsort()
// p and q are pointers to double
//
int compare(const void *pd0, const void *pd1) {
  double d0 = *(double *)pd0;
  double d1 = *(double *)pd1;
  if (d0 > d1) {
    return 1;
  }
  if (d0 < d1) {
    return -1;
  }
  return 0;
}
```

C vs. C++

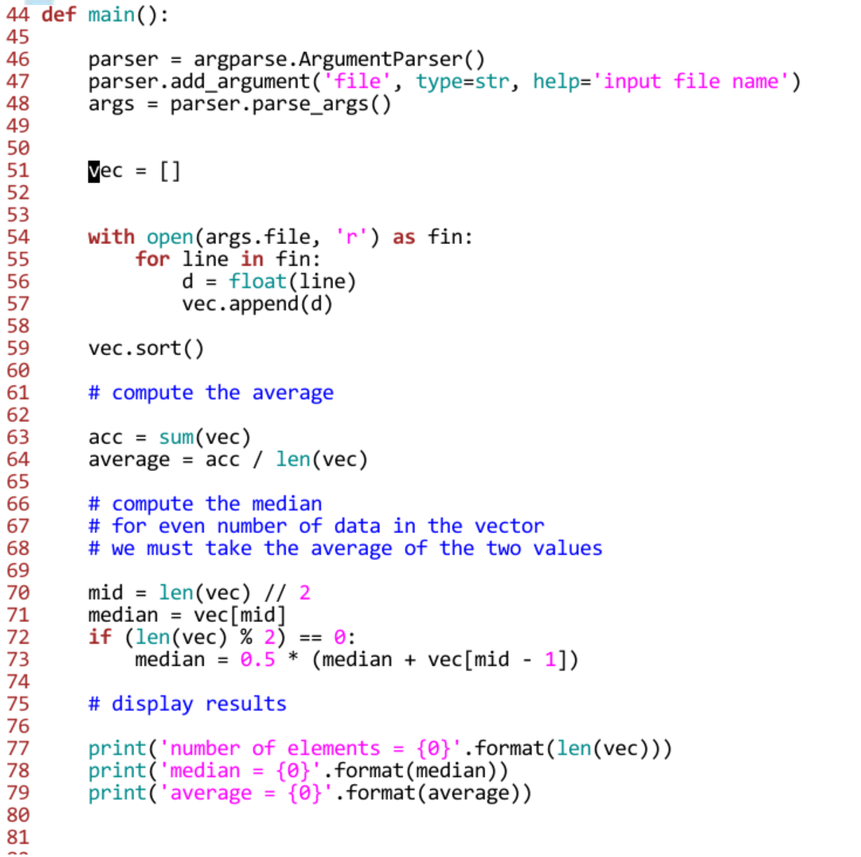


```
//
int main(int argc,char *argv[]) {
  char* file_name = argv[1];

  int res = 1000;
  double *buf = (double*)malloc(sizeof(double)*res);


  FILE *fin = fopen(file_name, "r");

  char  line[80];
  size_t   len;

  double mean = 0.0;
  int n = 0;
  while (fgets(line,80, fin) != NULL) {
    double d = strtod(line, NULL);
    if (n == res) {
      res += res;
      buf = (double*)realloc(buf,sizeof(double)*res);
    }
    buf[n++] = d;
    mean = (n==1) ? d : mean + (d - mean) / n;
  }

  qsort(buf, n, sizeof(double), compare);

  int mid = n / 2;
  double median = (n % 2) ? buf[mid] :
                            (buf[mid - 1] + buf[mid]) / 2;

  printf("number of elements = %d, median = %g, mean = %g\n", n,
      median, mean);

  fclose(fin);
  free(buf);
}
```

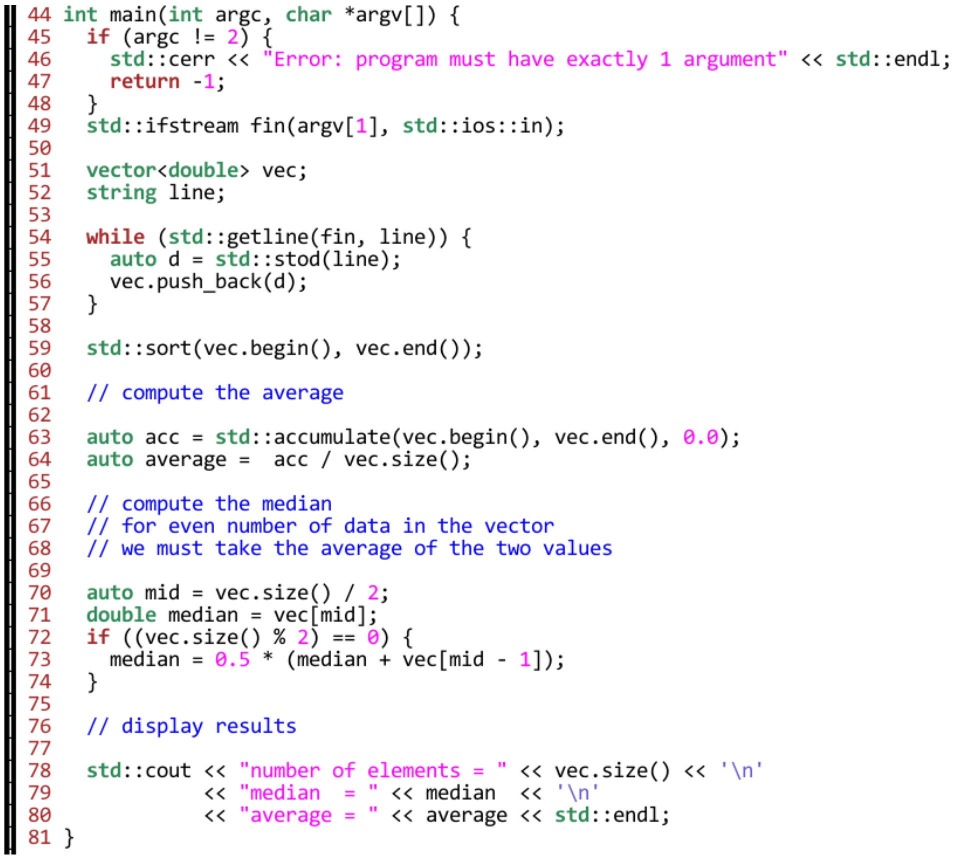


```
//
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> buf;




  std::ifstream fin(file_name, std::ios::in);

  string line;


  auto mean = 0.0;

  while (std::getline(fin, line)) {
    auto d = std::stod(line);




    buf.push_back(d);
    mean = (buf.size() == 1) ? d : mean + (d - mean) / buf.size();
  }

  std::sort(buf.begin(), buf.end());

  auto mid = buf.size() / 2;
  double median = (buf.size() % 2) ? buf[mid] :
                                     (buf[mid - 1] + buf[mid]) / 2;

  std::cout << "number of elements = " << buf.size()
       << ", median = " << median << ", mean = " << mean << std::endl;



}
```

Python vs. C++



```python
def main():

    parser = argparse.ArgumentParser()
    parser.add_argument('file', type=str, help='input file name')
    args = parser.parse_args()


    vec = []


    with open(args.file, 'r') as fin:
        for line in fin:
            d = float(line)
            vec.append(d)

    vec.sort()

    # compute the average

    acc = sum(vec)
    average = acc / len(vec)

    # compute the median
    # for even number of data in the vector
    # we must take the average of the two values

    mid = len(vec) // 2
    median = vec[mid]
    if (len(vec) % 2) == 0:
        median = 0.5 * (median + vec[mid - 1])

    # display results

    print('number of elements = {0}'.format(len(vec)))
    print('median = {0}'.format(median))
    print('average = {0}'.format(average))

```



```cpp
int main(int argc, char *argv[]) {
  if (argc != 2) {
    std::cerr << "Error: program must have exactly 1 argument" << std::endl;
    return -1;
  }
  std::ifstream fin(argv[1], std::ios::in);

  vector<double> vec;
  string line;

  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
  }

  std::sort(vec.begin(), vec.end());

  // compute the average

  auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
  auto average =  acc / vec.size();

  // compute the median
  // for even number of data in the vector
  // we must take the average of the two values

  auto mid = vec.size() / 2;
  double median = vec[mid];
  if ((vec.size() % 2) == 0) {
    median = 0.5 * (median + vec[mid - 1]);
  }

  // display results

  std::cout << "number of elements = " << vec.size() << '\n'
            << "median  = " << median  << '\n'
            << "average = " << average << std::endl;
}
```

Implemented in C++ (1)

```
//
// C++ version
// compiled with g++
// g++ -std=c++14 -O3 -o mean_and_median_cpp_version mean_and_median.cpp
//
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>

using std::string;
using std::vector;
```

Implemented in C++ (2)

```
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

C++ vs C: Speed?

```
shell> gcc -O3 -o mean_and_median_c_version mean_and_median.c

shell> time mean_and_median_c_version data/data_10000000.txt
number of elements = 9995416, median = 1721.52, mean = 2126.19
0:06.88 0.0%      0+0k 0+0io 1196pf+0w
```

```
shell> g++ -std=c++14 -O3 -o mean_and_median mean_and_median.cpp

shell> time mean_and_median data/data_10000000.txt
number of elements = 9995416, median = 1721.52, mean = 2126.19
0:06.62 0.1%      0+0k 0+0io 1197pf+0w
```

C++ vs C: Size?

```
shell> ls -l mean_and_median*.exe
-rwxr-xr-x 1 bernard None 70799 Dec 23 16:59 mean_and_median_c.exe
-rwxr-xr-x 1 bernard None 80709 Dec 23 17:08 mean_and_median_cpp.exe
```

Key Features of C++ (1)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`std::string` is a *sequence container* that encapsulates `char` value. Unlike C based strings, C++ strings do not end with \0.

`std::sort` is one of many algorithms available in the STL library.

Key Features of C++ (2)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`std::vector<>` is a *template* based sequence container that encapsulates dynamic size arrays. The elements are stored contiguously and can be accessed using offsets on regular pointers to elements `*(vec+n)` or `vec[n]`

Key Features of C++ (3)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`std::ifstream` implements high-level input operations on file.

Key Features of C++ (4)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`::` hierarchical names

`auto` automatic type inference

Key Features of C++ (5)

```
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                      (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`std::cout <<`
Forget your old `printf`, we'll use *stream* based output.
We'll use input stream `std::cin >>` to capture input.

Key Features of C++ (6)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

`vec.push_back(d)`
Appends a *copy* of the given element value to the end of the array.

Key Features of C++ (7)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<double> vec;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  auto mean = 0.0;
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
    mean = (vec.size() == 1) ? d : mean + (d - mean) / vec.size();
  }
  std::sort(vec.begin(), vec.end());
  auto mid = vec.size() / 2;
  double median = (vec.size() % 2) ? vec[mid] :
                                     (vec[mid - 1] + vec[mid]) / 2;
  std::cout << "number of elements = " << vec.size()
            << " median = " << median << " mean = " << mean << std::endl;
}
```

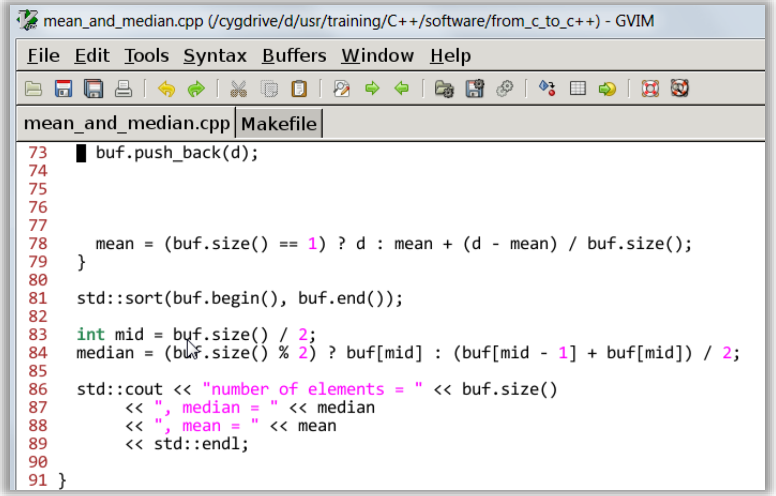
}
Exit from current scope: free all stack allocated resources.

More Features

- User defined types (classes) with inheritance and polymorphism
- Separate name space with namespaces
- Auto and reference variables, const attributes
- Function overloading, lambda functions
- Generic programming with template
- Advanced error handling with exceptions
- Stream based input/output
  - {i,o}fstream, {i,o}stringstream, cout, cin, cerr,
- Extensive library of containers
  - array, vector, list, queue, binary tree, hash table
- Miscellaneous library functions
  - multi-thread programming, time measurement
  - regular expressions, complex numbers, random generators

Compilation & Link (1)

- Old school flow
  - Shell based (zsh or bash)
  - Must use a text editor to enter your program (notepad++, sublime, ...)
  - Using g++ (>= 7.2.0) tool chain and Makefile script to facilitate compile and link.

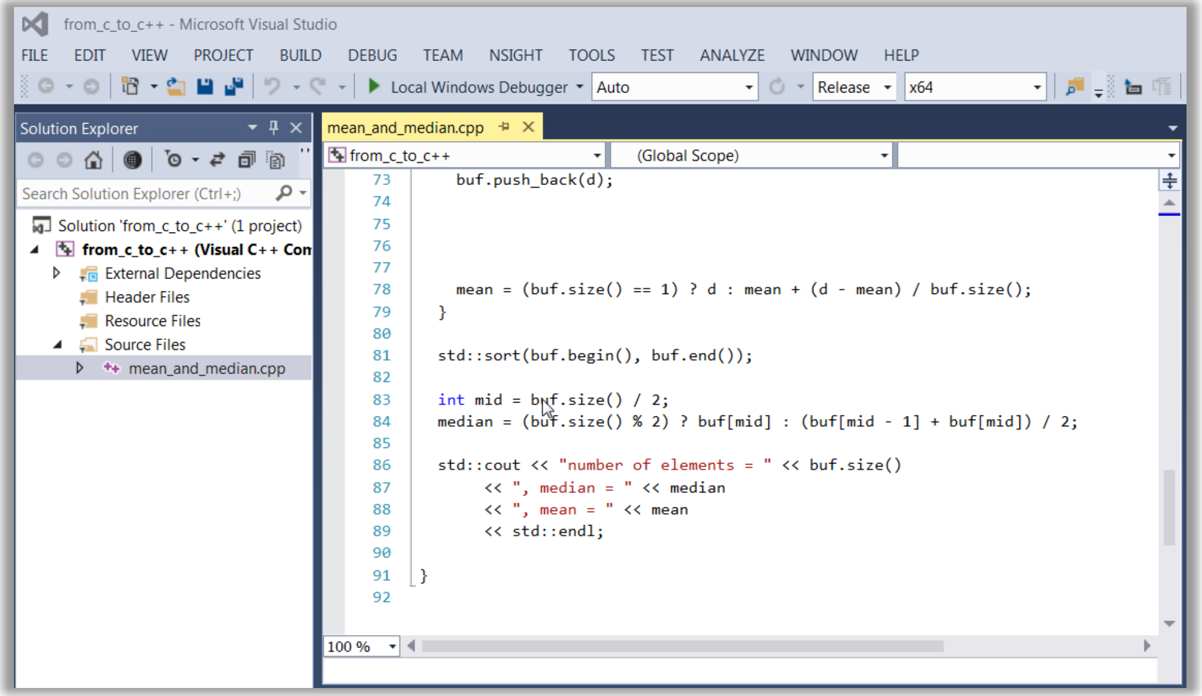

mean_and_median.cpp (/cygdrive/d/usr/training/C++/software/from_c_to_c++) - GVIM
File Edit Tools Syntax Buffers Window Help
mean_and_median.cpp
Makefile

```
    buf.push_back(d);



    mean = (buf.size() == 1) ? d : mean + (d - mean) / buf.size();
  }

  std::sort(buf.begin(), buf.end());

  int mid = buf.size() / 2;
  median = (buf.size() % 2) ? buf[mid] : (buf[mid - 1] + buf[mid]) / 2;

  std::cout << "number of elements = " << buf.size()
       << ", median = " << median
       << ", mean = " << mean
       << std::endl;

}
```

```
shell> make
Compiling release version of mean_and_median.cpp
g++ -c -std=c++14 -O3 -o ./build/mean_and_median.r.o mean_and_median.cpp
g++ -o mean_and_median_cpp ./build/mean_and_median.r.o
shell> mean_and_median_cpp data_10.txt
number of elements = 10, median = 1655.83, mean = 3199.7
```

Compilation & Link (2)

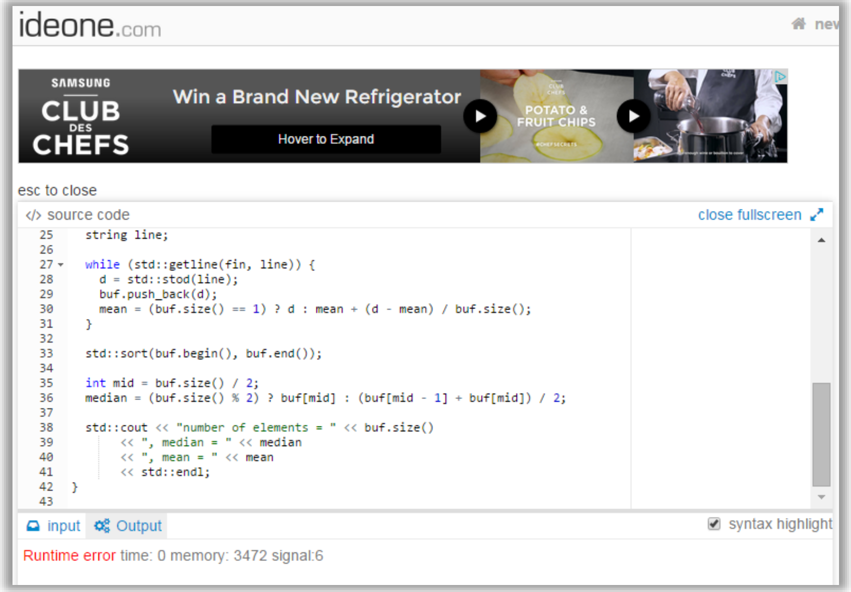
- IDE based flow: Visual Studio, Eclipse, QT Creator, CodeBlock


from_c_to_c++ - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM NSIGHT TOOLS TEST ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Release
x64
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'from_c_to_c++' (1 project)
from_c_to_c++ (Visual C++ Con
External Dependencies
Header Files
Resource Files
Source Files
mean_and_median.cpp
mean_and_median.cpp
from_c_to_c++
(Global Scope)
100 %

```
    buf.push_back(d);



    mean = (buf.size() == 1) ? d : mean + (d - mean) / buf.size();
  }

  std::sort(buf.begin(), buf.end());

  int mid = buf.size() / 2;
  median = (buf.size() % 2) ? buf[mid] : (buf[mid - 1] + buf[mid]) / 2;

  std::cout << "number of elements = " << buf.size()
       << ", median = " << median
       << ", mean = " << mean
       << std::endl;

}

```

Compilation & Link (3)

- Web based flow
  - gcc.godbolt.org (Clang, GCC, Intel ICC)
  - Rise4Fun (Microsoft VC++)
  - Rextester (Clang, GCC, VC++)
  - ideone.com (GCC)
- Limited to “simple code”
  - No file read or write



```
  string line;

  while (std::getline(fin, line)) {
    d = std::stod(line);
    buf.push_back(d);
    mean = (buf.size() == 1) ? d : mean + (d - mean) / buf.size();
  }

  std::sort(buf.begin(), buf.end());

  int mid = buf.size() / 2;
  median = (buf.size() % 2) ? buf[mid] : (buf[mid - 1] + buf[mid]) / 2;

  std::cout << "number of elements = " << buf.size()
       << ", median = " << median
       << ", mean = " << mean
       << std::endl;
}

```

```
Runtime error time: 0 memory: 3472 signal:6
```

Coding Style

- Your code is written once, but it will be read and updated many times (debugging, refactoring, code review).
  - Most time in software development is spent debugging and maintaining existing code!
- Coding style guarantees a common layout and a consistent structure, but no international standard...
- We shall use Google coding style (details)

```
shell> cpplint mean_and_median.cpp
mean_and_median.cpp:90:  Redundant blank line at the end of a code block
should be deleted.  [whitespace/blank_line] [3]
Done processing mean_and_median.cpp
Total errors found: 1
```

Static Analysis

- Maximize the opportunity to find bug before run time
  - Cost of latent bugs increases with time (design, compile, debug, release, ...)
- Most compilers offer options to detect common problems
  - use of uninitialized variables, out of bound array indexes, ...
- With g++ we shall use additional compiler options which turn on warnings

```
shell> make
Compiling release version of mean_and_median.cpp
g++ -Wall -Wshadow -Wextra -Werror -c -std=c++14 -O3 -o
./build/mean_and_median.r.o mean_and_median.cpp
g++ -o mean_and_median_cpp ./build/mean_and_median.r.o
```