ROBO4
Algorithms & OOP
Using Modern C++

Welcome

C++ in the Real World

A pragmatic approach to modern C++ programming

POLYTECH
NICE-SOPHIA

Introduction

Course Syllabus 2024 (1)

- Lectures & Labs (30 hours)
- Exam (2 hours)
- Grade:
  - Two evaluated projects (in group)
  - Exam: final examination
  - Final score: weighted average of the 3 grades.
    - 1/3 project#1
    - 1/3 project#2
    - 1/3 exam

Course Syllabus 2024 (1)

- Schedule

#	Week	Date	Time	Where	Hours (Total)	
1	38	Thursday, 19-Sep-2024	8:00 – 12:15	A241	4:00 (4:00)	
2	39	Friday, 27-Sep-2024	8:00 – 12:15	A241	4:00 (8:00)	
	40	Off				
3	41	Friday, 11-Oct-2024	8:00 – 12:15	A241	4:00 (12:00)	
4	42	Friday, 18-Oct-2024	8:00 – 12:15	A241	4:00 (16:00)	
5	43	Tuesday, 22-Oct-2024	8:00 – 12:15	A240	4:00 (20:00)	
	44	Off				
6	45	Friday, 8-Nov-2024	8:00 – 12:15	A241	4:00 (24:00)	
7	46	Friday, 15-Nov-2024	8:00 – 12:15	A244	4:00 (28:00)	
8	47	Friday, 22-Nov-2024	8:00 – 10:00	A242	2:00 (30:00)	Exam
	47	Friday, 22-Nov-2024	10:15 – 12:15	A242	2:00 (32:00)	Project2

project #1

project #2

beware of the date

Course Objectives (1)

- *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 algorithm and design patterns
- *Explore* available documentation to write more complex programs.

Course Objectives (2)

- Focus on useable C++ subset
  - No meta programming, complex template or fancy class derivation
- Use proven industry practices, e.g.
  - const correctness, unique pointer, brace initialization, ...
- Improve common algorithm knowledge
  - sort, search, ...
  - algorithm complexity, big O notation
- Increase knowledge of STL libraries
  - basic containers: string, vector, map, ...
  - useful libraries: chrono, thread, regex, ...

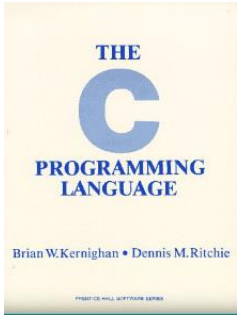
Pre-requisite

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

C++ Overview

C to C++

- Ken Thompson and Dennis Ritchie created the C language in 1972.
  - Fast, Simple and Cross-Platform



- In 1983, Bjarne Stroustrup wanted a language that was:
  - Fast, Simple, Cross-Platform
  - With High-level features (classes and objects)
  - Supporting generic code

=> that was the beginnings of C++

Importance of C++

TIOBE
the software quality company

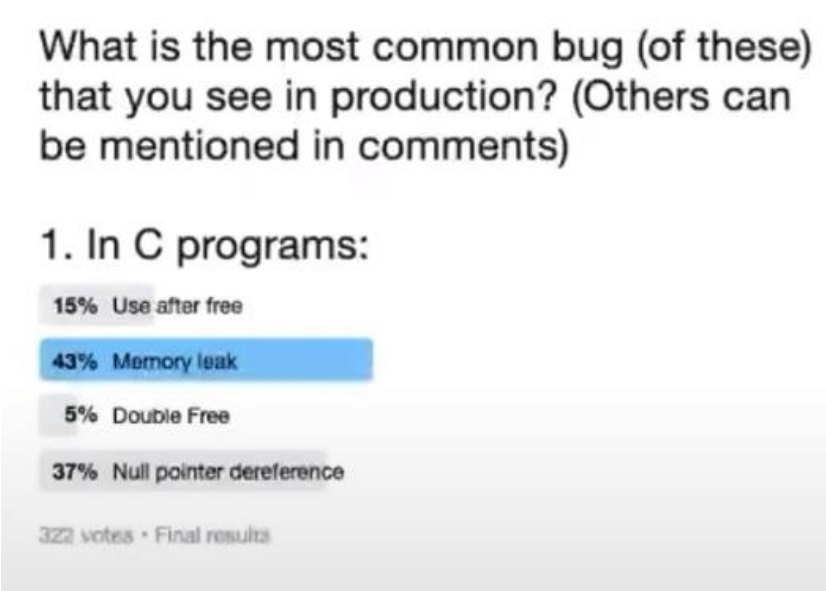
About us | Knowledge | News | Coding Standards | TIOBE Index
Products | Quality Models | Markets | Sche

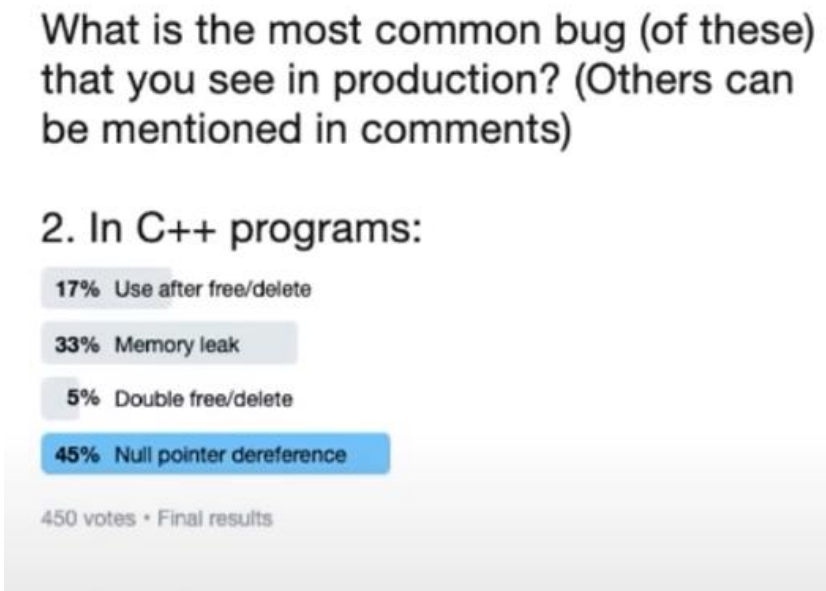
Aug 2023	Aug 2022	Change	Programming Language	Ratings	Change
1	1		Python	13.33%	-2.30%
2	2		C	11.41%	-3.35%
3	4	↑	C++	10.63%	+0.49%
4	3	↓	Java	10.33%	-2.14%
5	5		C#	7.04%	+1.64%
6	8	↑	JavaScript	3.29%	+0.89%
7	6	↓	Visual Basic	2.63%	-2.26%
8	9	↑	SQL	1.53%	-0.14%

Benefits of C++ (1)

- C++ provides direct map to hardware
  - No virtual machine, no interpreted code
- No abstraction layer in production code
  - No language between C++ and assembly code
- Extensive support to modern hardware
  - C++ compilers are available for most CPUs [ARM, Intel, Mips,…]
  - Vast number of low level libraries to access GPU, AI Accelerator

Benefits of C++ (2)





What is C++?

- Fast, Simple, Cross-Platform
- With High-level features (classes and objects)
- Supporting generic code

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

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

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

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

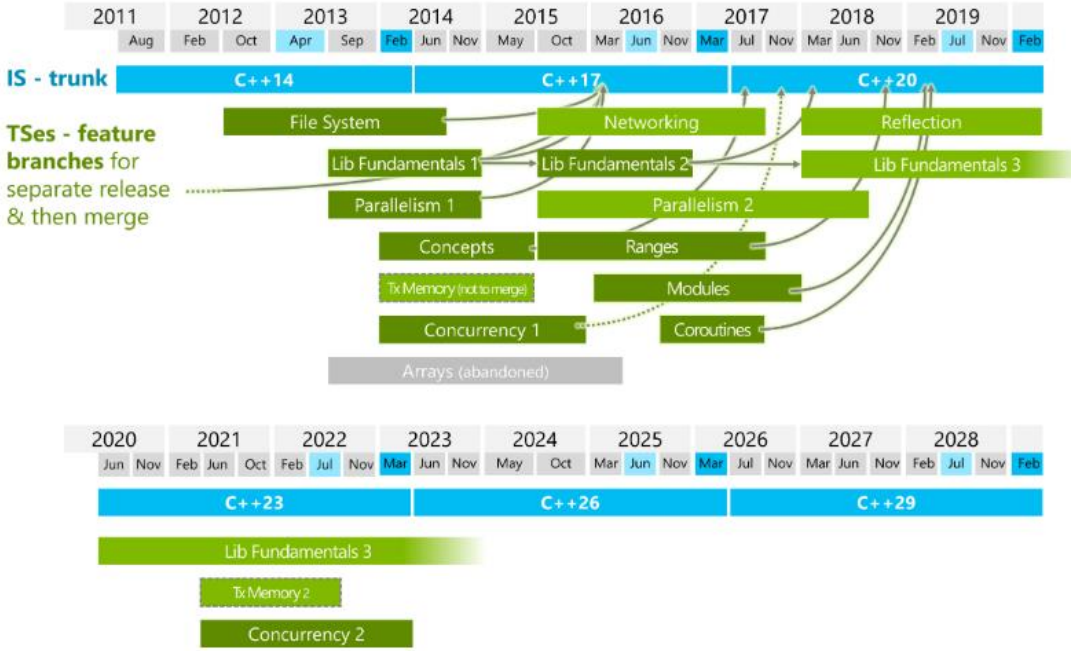
Introduction (3)

- 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

Introduction (2)

- We shall cover *Modern C++* (from C++11) and associated STL (standard template library)



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

Introduction (3)

- C++ has been changing gradually over the past 13 years
  - Latest version (23) brings a list of interesting features

```
#include <iostream>

int main()
{
    std::cout << "Hello World!"
              << std::endl;
    return 0;
}
```

```
import std;

int main()
{
    std::println("Hello World!");

}
```

Which C++? (1)

- Write a simple program accepting 2 positive integers as inputs named *a* and *b*, with $a \leq b$, which returns the sum of the even numbers between these two integers.

```
def calculate(a, b):

    if a > b or a < 0:
        return 0

    even_sum = 0
    for i in range(a, b + 1):
        if i % 2 == 0:
            even_sum += i
    return even_sum
```

Which C++? (3)

Imperative vs. Declarative

```
int calculate(int a, int b) {
    if (a > b or a < 0) {
        return 0;
    }
    int sum = 0;
    for (int n = a; n <= b; ++n) {
        if (n % 2 == 0) {
            sum += n;
        }
    }
    return sum;
}
```

```
int calculate(int a, int b) {
    return (a > b or a < 0) ? 0 :
        ranges::accumulate(
            rv::iota(a, b + 1) |
            rv::filter([](auto e) { return e % 2 == 0; }), 0);
}
```

```
int calculate(int a, int b) {
    if (a > b or a < 0) {
        return 0;
    }
    int max1 = (a-1)/2;
    int max2 = (b)/2;
    int sum1 = (max1 * (max1 + 1));
    int sum2 = (max2 * (max2 + 1));
    return sum2 - sum1;
}
```

References [1]

- C++ FAQs, Tutorials and Courses
  - https://hackingcpp.com/index.html
  - http://www.tutorialspoint.com/cplusplus/index.htm
  - https://isocpp.org/faq
  - https://en.wikipedia.org/wiki/C%2B%2B
  - https://github.com/rougier/cpp-crash-course
  - https://github.com/cppcon/cppcon2015 ... 2022
- Books
  - The C++ Programming Language, from Bjarne Stroustrup (2013)
  - The C++ Standard Library: A Tutorial and Reference, from Nicolai M. Josuttis (2012)
  - https://en.wikibooks.org/wiki/C%2B%2B_Programming

References [2]

- Language Reference
  - http://en.cppreference.com
  - https://isocpp.org/faq

- YouTube
  - https://www.youtube.com/user/CppCon
    All the video of the most prestigious conference on C++ are found here
  - https://www.youtube.com/c/CopperSpice/featured
    Excellent Channel, with lots of short videos < 15min, each one covering a specific C++ topic.

First Concepts
std::string and std::vector<T>

Assignment #1 (1)

- Write a short program in C++ which reads a sequence of numbers (double precision) from a file and compute the *mean* and *median* value. File size is unknown, it could have 10 lines or 1 billion lines, you cannot 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

$$CMA_n = \frac{x_1 + \cdots + x_n}{n}.$$

$$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1} = CMA_n + \frac{x_{n+1} - CMA_n}{n+1}$$

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

$$CMA_n = \frac{x_1 + \cdots + x_n}{n}.$$

$$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1} = CMA_n + \frac{x_{n+1} - CMA_n}{n+1}$$

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])


    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;
  }

  vector<double> vec;
  string line;

  std::ifstream fin(argv[1], std::ios::in);
  while (std::getline(fin, line)) {
    auto d = std::stod(line);
    vec.push_back(d);
  }

  std::ranges::sort(vec);

  // compute the average
  auto sum_of_values = std::ranges::fold_left(vec, 0.0, std::plus<double>());
  auto average =  sum_of_values / 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 = std::midpoint(median, vec[mid - 1]);
  }

  std::println("number of elements = {}", vec.size());
  std::println("median = {}", median);
  std::println("average = {}", average);
}
```

warning
need g++≥
14.0 compiler

Improved C++ Code

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

Using g++ v13 compiler and fmt library

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++20 -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)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argumen
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

`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)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 arg
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

`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)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

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

Key Features of C++ (4)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

`auto` automatic type inference

`::` hierarchical names

Key Features of C++ (5)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

`std::cout <<`
Forget your old `printf`, we'll use *stream* based output.
`fmt::format` enables Python style formatting.

We'll use input stream `std::cin >>` to capture input.

Key Features of C++ (6)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

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

Key Features of C++ (7)

```
1  int main(int argc, char *argv[]) {
2    if (argc != 2) {
3      std::cerr << "Error: program must have exactly 1 argument" << std::endl;
4      return -1;
5    }
6    vector<double> vec;
7    string line;
8    std::ifstream fin(argv[1], std::ios::in);
9    while (std::getline(fin, line)) {
10     auto d = std::stod(line);
11     vec.push_back(d);
12   }
13   std::ranges::sort(vec);
14   auto acc = std::accumulate(vec.begin(), vec.end(), 0.0);
15   auto average =  acc / vec.size();
16   auto mid = vec.size() / 2;
17   double median = vec[mid];
18   if ((vec.size() % 2) == 0) {
19     median = std::midpoint(median, vec[mid - 1]);
20   }
21   std::cout << fmt::format("number of elements = {}\n", vec.size());
22   std::cout << fmt::format("median = {}\n", median);
23   std::cout << fmt::format("average = {}\n", average);
24 }
```

}
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

POLYTECH
NICE-SOPHIA

Tool Chains

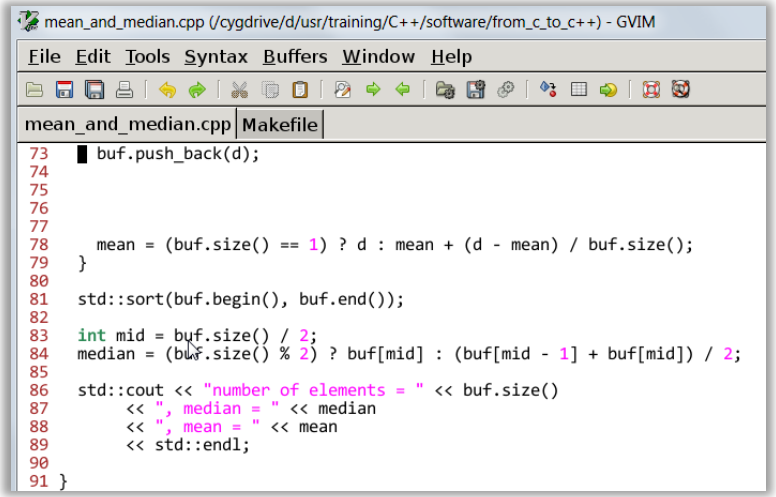
C++ Tool Chain (1)

- Note:
  - This section is not directly related to C++ course, but
  - It is essential for all students to compile/execute/debug programs on his/her laptop.

- Tool Chain: set of consistent tools to
  - Edit your source files
  - Compile & Link the source files to produce an executable
  - Execute the program in a terminal
  - Debug the program
  - Lint the program
  - Manage the source code version

C++ Tool Chain (2)

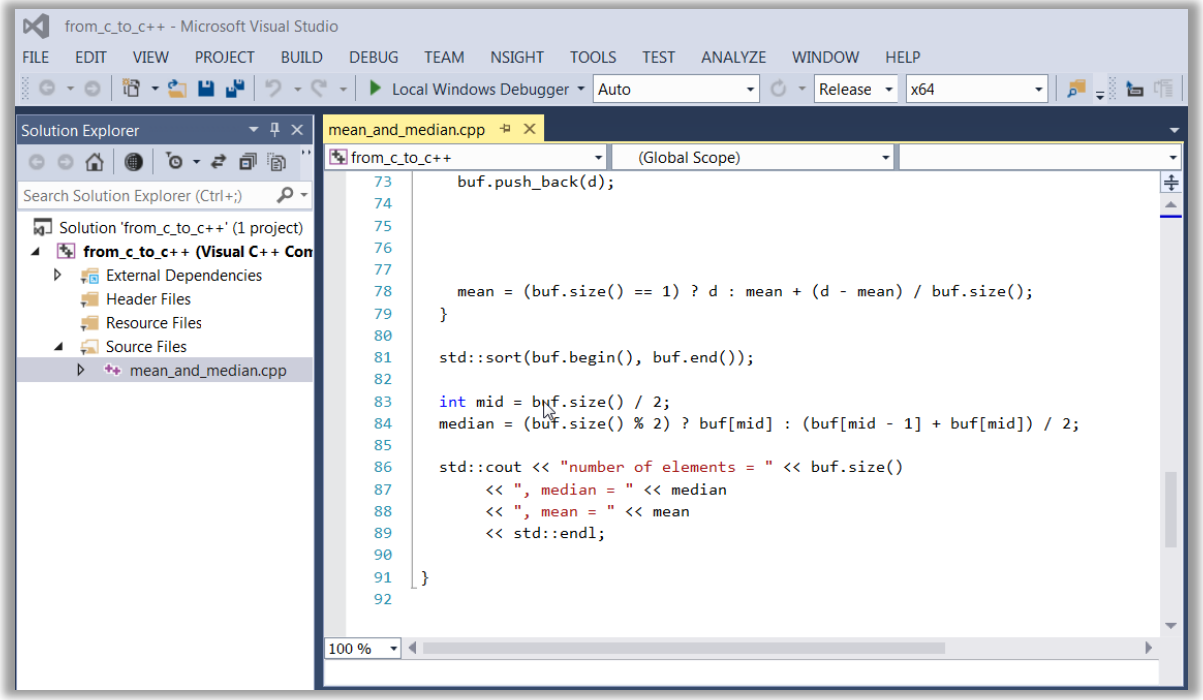
- Old school flow
  - Shell based (zsh or bash)
  - Must use a text editor to enter your program (notepad++, sublime, ...)
  - Using g++ (>= 10.0) tool chain and Makefile script to facilitate compile and link.




```
shell> make
Compiling release version of mean_and_median.cpp
g++ -c -std=c++20 -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
```

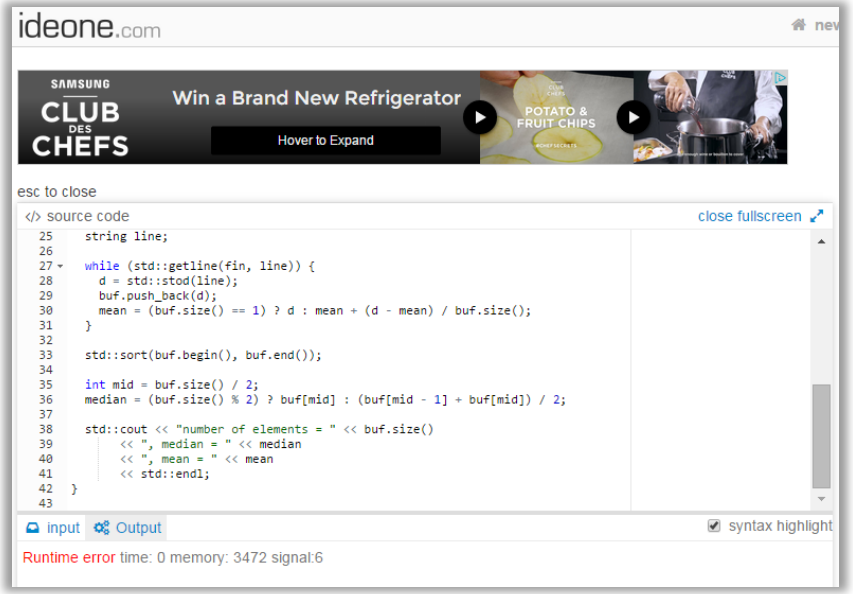
C++ Tool Chain (3)

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



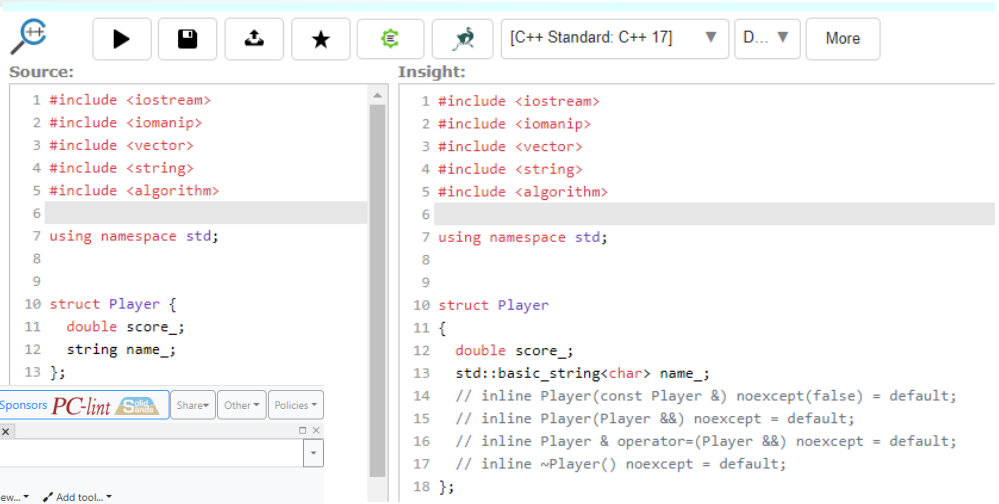
C++ Tool Chain (4)

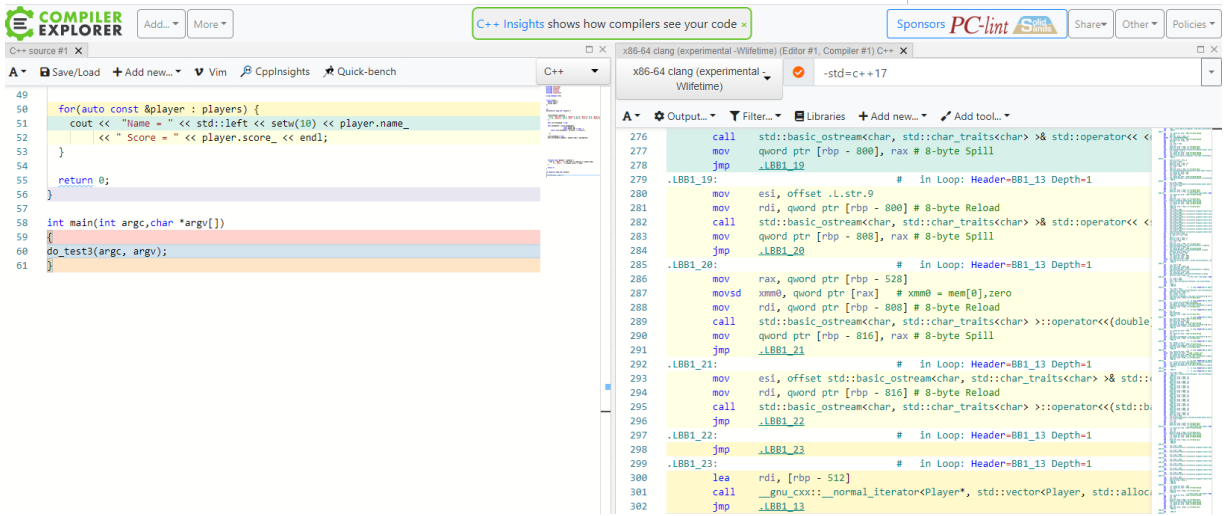
- Web based flow
  - gcc.godbolt.org (Clang, GCC, Intel ICC)
  - Rise4Fun (Microsoft VC++)
  - Rextester (Clang, GCC, VC++)
  - ideone.com (GCC)
- Limited to “simple code”
  - No file read or write



C++ Tool Chain (5)

- C++ Explorer
  - https://cppinsights.io
  - https://godbolt.org





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++20 -O3 -o
./build/mean_and_median.r.o mean_and_median.cpp
g++ -o mean_and_median_cpp ./build/mean_and_median.r.o
```

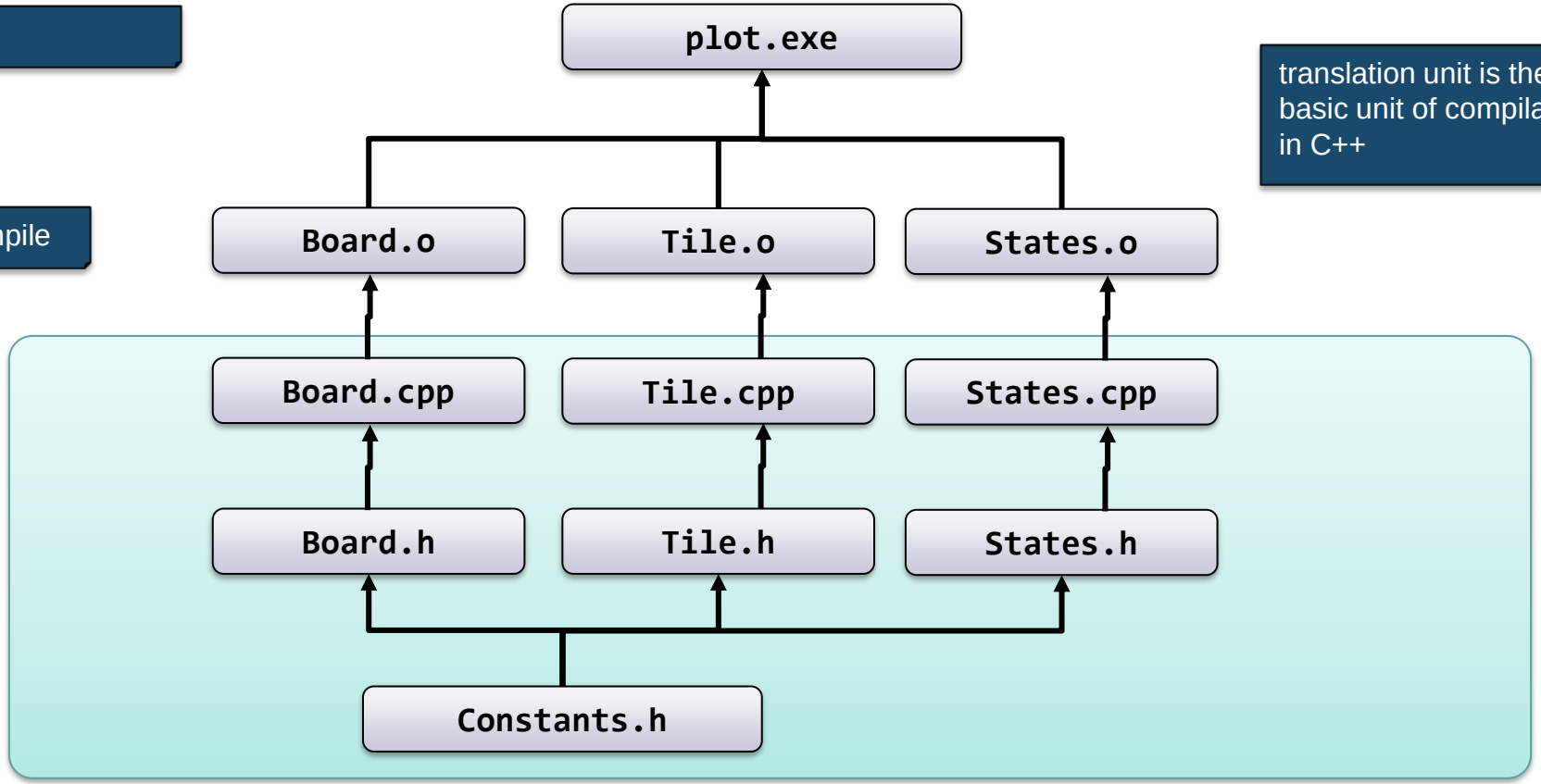
Revision Control Systems

- Many systems are available: rcs, clearcase, git, svn, ...
- We shall use git
  - easy to install and use
  - scalable from one person to teams of hundreds.
- Useful git links
  - Full documentation here
  - Good introduction: http://githowto.com/checking_status
  - Hundreds of video on youtube...

C++ Projects

Link

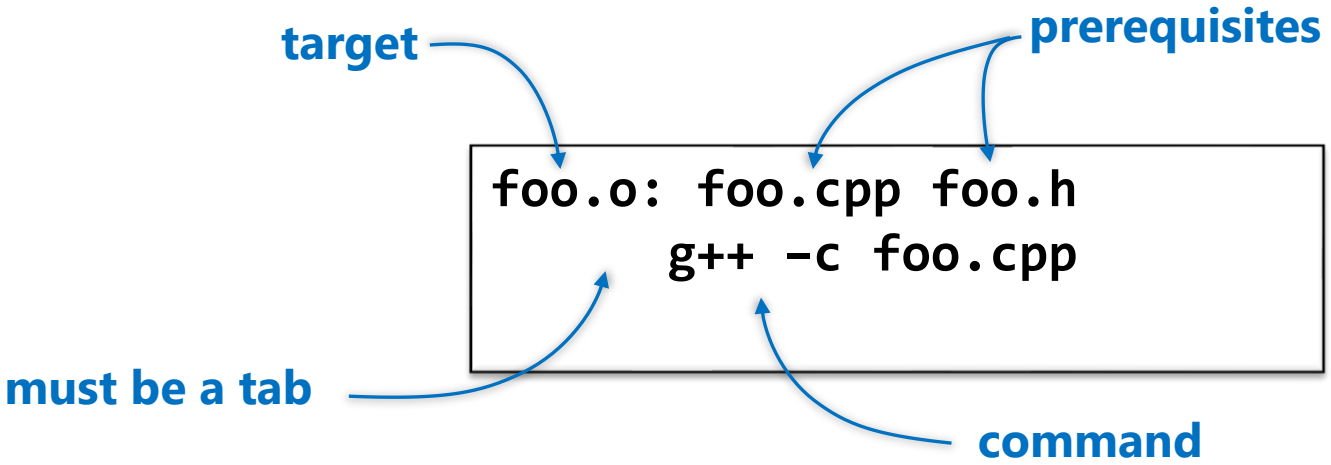
Compile



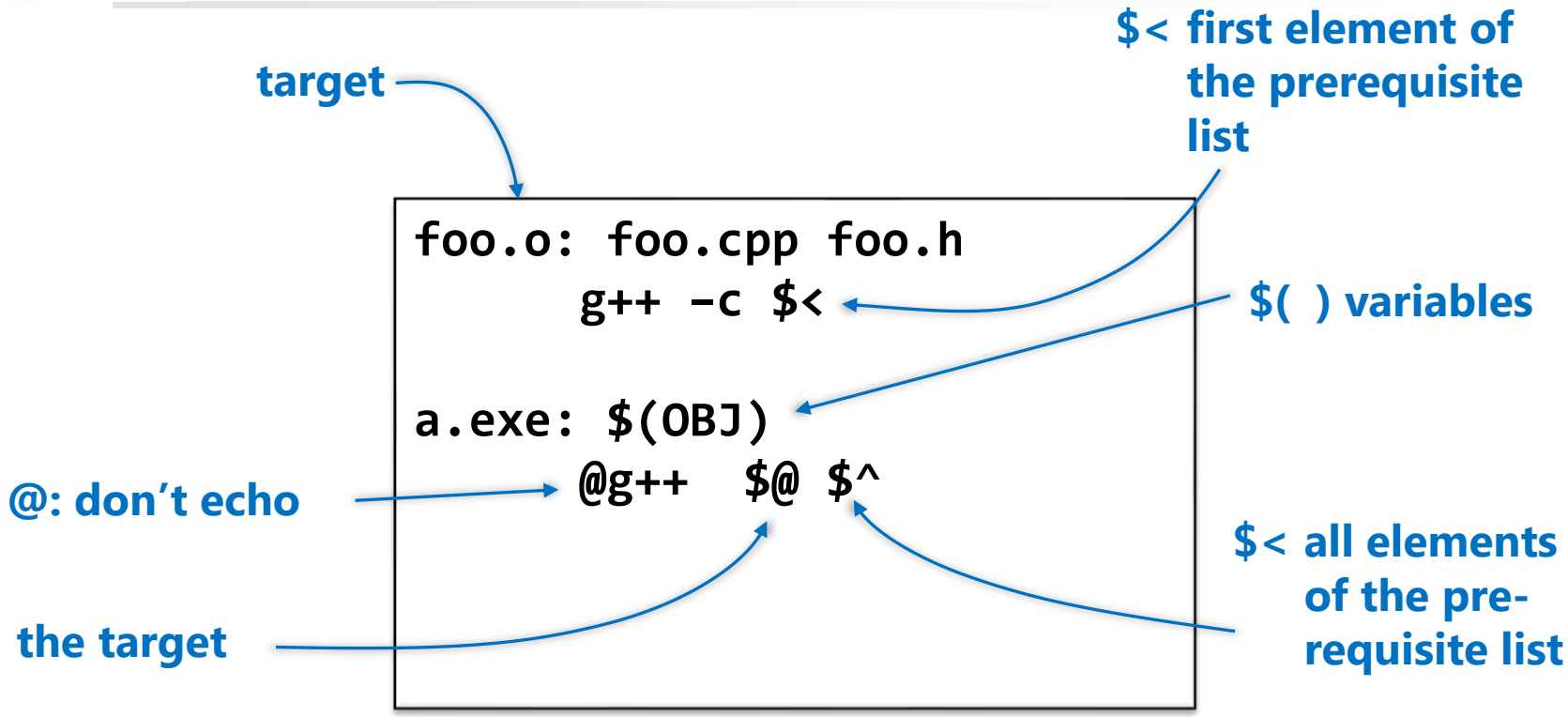
translation unit is the basic unit of compilation in C++

Makefile (1)

- A Makefile specifies how *out of date* files must be processed.
- The executable `make` reads the Makefile and invokes the rules needed to re-build the target.
- Full documentation: here



Makefile (2)


target
$< first element of the prerequisite list
foo.o: foo.cpp foo.h
g++ -c $<
$( ) variables
a.exe: $(OBJ)
@g++ $@ $^
@: don't echo
$< all elements of the pre-requisite list
the target

First Project: Install & Run

```
shell> git clone https://github.com/elec4/mean_and_median.git
Cloning into 'mean_and_median'...
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 20 (delta 4), reused 18 (delta 2), pack-reused 0
Unpacking objects: 100% (20/20), done.
Checking connectivity... done.
shell> cd mean_and_median
shell> make
[...]
shell> /bin/time -p ./mean_and_median data/data_100000.txt
number of elements = 99957, median = 1715.81, mean = 2124.64
real 0.12
user 0.01
sys 0.00
```

Using the debugger: GDB (1)

- Goal: find the type of variable `mid`

```
auto mid = vec.size() / 2;
```

- We must use an executable with debug information and call the debugger with that executable.

debug prompt

```
shell> make debug
[...]
shell> gdb --args mean_and_median.debug data/data_10.txt
GNU gdb (GDB) 7.10.1
Copyright (C) 2015 Free Software Foundation, Inc.
[...]
(gdb) run
```

Using the debugger: GDB (2)

- You may find the following useful
  - GDB quick reference sheet [here]
  - Most useful commands

start	Run the program with a temporary breakpoint at main()
b ***n***	Set breakpoint at line n
c	Continue debug after a breakpoint
n	Execute the next line of code
s	Execute the next line of code, if function call, step into that function.
l	List 10 lines of source code
p ***expr***	Evaluate the expression and print the result