C++ Programming

Lecture 0
Software Engineering Group

Philipp D. Schubert
C++ is easy.
It’s like riding a bike.
Except the bike is on fire,
and you’re on fire
and everything is on fire
because you’re in hell.
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. C++ compilers
5. A “Hello, World!” program
6. Setting up a development environment
7. Basic terms & concepts
Organization

- **Rooms**
  - Exercises: D1, Friday 14:15-15:45
  - Lecture: D1, Friday 16:00-17:30

- **Instructor**
  - Philipp Schubert  F1.116 (Heinz Nixdorf Institut)
  - E-Mail  philipp.schubert@upb.de
  - Web  https://www.hni.uni-paderborn.de/swt/lehre/cppp/

- **Prerequisite**
  - No programming experience
  - Knowledge of how to use a computer
    - Word-processing software
    - Operating system (Linux / Windows / Mac)
[Figure taken from maps.google.de]
Organization

- **Benefits**
  - Be confident to take advances courses that require C++
    - Like “Algorithms”
  - Ease to realize programming projects
  - Will be useful for computational thinking
  - Better understanding of how a computer works

- **Studium Generale (SG) EIM-I**
  - Students of computer science will not receive credit points
  - Students of electrical engineering will not receive credit points
  - When in doubt ask your examination office
  - All other students will receive 4 credit points
  - Everyone obtains a nice certificate for their CV


Get the book
theboostcpplibraries.com

... oder das Training
boost-cpp-master-class.eventbrite.de
Organization

- Some of you have not yet registered
  - This is not required
- If you are not registered in PAUL
  - Register on this list
- I will send emails with additional material
- You will receive an email tomorrow
  - Unique ID
  - Link to Google spreadsheet

<table>
<thead>
<tr>
<th>Name</th>
<th>Surname</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. A “Hello World” program
5. C++ compilers
6. Setting up a development environment
7. Basic terms & concepts
Course outline

- Basic introduction
  - History of C & C++
  - Compilers
  - Development environments
  - Basic terms and concepts

- Basic C++ programming
  - Primitive data types, strings, vectors, arrays, pointers
  - Expressions, statements
  - Structures, unions, enumerations
  - Functions, Classes
Course outline

- C++ **Standard Template Library (STL)**
  - IO, containers, generic algorithms
  - Dynamic memory
  - Smart pointers
- Advanced techniques
  - Copy control, standard class members
  - Operator overloading
  - Object-oriented programming
  - Templates and generic programming
Course outline

- Useful libraries
  - OpenMP, OpenCV, OpenCL, OpenGL, …
  - Qt
  - gtest
  - And other useful libraries
- Where to find the desired information you need
- Don’t reinvent the wheel
  - Use libraries
Literature


Additional material will be handed out in time.

Various different input-channels are important:

- Lecture
- Exercises
- I’ll try to make links to books and youtube
- Talk to each other
Exercises

- Weekly exercises (strict regulations)
  - Theoretical exercises
  - Practical exercises
- Results get evaluated
  - Achieve 50% of the points during semester
- Final (small) project
  - Solve a programming task
- Certificate (+ credit points)
  - Pass exercises + project solved
  - No final exams
- Do not plagiarize (plage)
- Questions so far?
What is C++?

TIOBE Programming Community Index

Source: www.tiobe.com

[Graph showing the TIOBE Programming Community Index from 2002 to 2016, with various programming languages like Java, C, C++, C#, Python, JavaScript, PHP, and others plotted over time.]
What is C++?

- An object-oriented programming language
- Generic Programming
- Template meta-programming
- Buffer overflows
- Classes
- Too big
- Host for DSLs
- A hybrid language

- Embedded systems
- Low level
- A random collection of features
- Class hierarchies
- Multi-paradigms
- A failed attempt to build Java
- It’s C
- Too complicated
Advice

- Don´t be frightened
- Learning a new language takes time
- Practice a lot
- Read a lot (books and C++ forums / as well as code)
- Do the exercises
- Ask yourself: Why does this work?
  - If you are curios about something → use google
    - … and share your knowledge and discuss with friends
- Programming will be fun when understood
History of the C++ language

- All started with BCPL
- B - a language to implement operating systems
- C – better than B
  - Brian Wilson Kernighan
  - Dennis MacAlister Ritchie
- C with Classes
  - Bjarne Stroustrup
- C++
  - Dynamically evolving
  - C++11
  - C++14
  - Currently there is much work on C++16 / C++17

[Figure and images taken from images.google.de/ and A Tour of C++, Stroustrup 2013]
History of the C++ language

- But why are we not learning C++16, C++17, …
- Adaption needs time
  - Concepts and ideas first
  - Compiler implementation follows
  - // void …
- Industry usually adapt ~ 5-10 years later
  - There are reasons for that
    - Concepts have to be proven as useful
    - Compilers have to mature over time

[Figure from A Tour of C++, Stroustrup 2013]
History of the C++ language

- BCPL, B, C,
  - Why not D after C?
  - C was and is still tremendously successful
  - Lots of existing code was / is still written in C
  - Don’t break compatibility!
  - Be an increment rather than a new language
  - A language called D exists
    - D is no longer compatible with C
  - Be aware: Modern C++ is not C
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. C++ compilers
5. A “Hello World” program
6. Setting up a development environment
7. Basic terms & concepts
What is a compiler?

Figure 1.1: A compiler

Figure 1.2: Running the target program

Are there other forms? Interpreter

![Diagram of an interpreter](image)

Figure 1.3: An interpreter

Even more: hybrid compilers

Figure 1.4: A hybrid compiler
GCC and Clang are language processing systems

- C++ is (usually) a compiled language
- C++ compilers are language processing systems/ compiler tool chains

Figure 1.5: A language-processing system
C++ compilers

- **Gnu Compiler Collection GCC**
  - Includes C and C++ front-ends
  - Standard on most Linux distr.
  - “Most used C/C++ compiler in the world”
  - Fist stable release was v 1.17 (1988)
  - Monolithic design
  - Written by bootstrapping
    - Written by *something else* until its powerful enough to compile itself

- **Clang**
  - Compiler front-end for C-like languages (including C and C++)
  - Used by Google, Apple, Oracle …
  - Started as a Ph.D. thesis by Chris Lattner
  - Stable version in 2009
  - Part of a reusable compiler infrastructure (LLVM project)
  - Written in C++

There are a lot more: Intel icc, IBM C++, MSVS C++, Oracle ++, Apple C++, Bloodshed Dev-C++, EDG C++
Remark on what follows

- Keep simple things simple,
  as simple as possible, but not simpler! (Einstein)

- Problem: where to start when learning a programming language?
  - In order to be able to start **at all** we have to …
    1. take certain things for granted
    2. learn the WHY over time
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. C++ compilers
5. A “Hello World” program
6. Setting up a development environment
7. Basic terms & concepts
A “Hello World” program

- Shortest valid C++ program

```
int main() { return 0; }
```
or
```
int main() {}
```

- A “Hello World” program
  - Uses a header file
  - A comment
  - `main()` function (with arguments)
  - Uses a namespace
  - `::` scope and `<<` streaming operator
  - Uses a string literal and a variable `cout`
  - `return 0;` a value that is returned to the OS
    - ‘0’ indicates success
    - Others than ‘0’ indicate failure

```
#include <iostream>

// This function prints Hello, World!
int main(int argc, char** argv) {
    std::cout << "Hello, World!\n";
    return 0;
}
```
A “Hello World” program

- Tell the compiler to translate ‘hello.cpp´ into executable machine code
- Command:
  - cc hello.cpp -o hello
  - You can execute the program ‘hello´ with ./hello
- Instead of cc use g++ or clang++

Edit a file e.g. ‘hello.cpp´ with the following content:

```cpp
#include <iostream>

int main(int argc, char** argv){
    std::cout << "Hello, World!\n";
    return 0;
}
```
A “Hello World” program

- Some useful compiler flags
  - `-Wall` turns on compiler warning
  - `-Wextra` turns on even more warnings
  - `-g` insert debugging symbols
  - `-Ox` turn on compiler optimization (x is a number: 0,1,2,3)
  - `-o` specify the output file
  - `-std=X` specify the C++ standard
    e.g. `-std=c++11` or `-std=c++14`

Edit a file e.g. `hello.cpp` with the following content:

```cpp
#include <iostream>
int main(int argc, char** argv) {
    std::cout << "Hello, World!\n";
    return 0;
}
```
A “Hello World” program

- #-directives are instructions for the preprocessor
  - Preprocessor runs over the program first
  - Then compiler starts
- `#include` directives just performs textual insertion
- `std::` is a namespace
  - Namespaces hold code
  - Helps to avoid collisions (e.g. variable names, function names, …)
  - From now on we are using the namespace of the STL
    - `using namespace std;`
    - We can use `cout` without prefix `std::`

```cpp
#include <iostream>

int main(int argc, char** argv) {
    std::cout << "Hello, World!\n";
    return 0;
}
```

Figure 1.5: A language-processing system

[Figure from Compilers: Principles, Techniques, & Tools, 2007]
A “Hello World” program

- Compiler option –S shows the assembly code
- `cc hello.cpp -S -o hello.as`

```c
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    std::cout << "Hello, World!\n";
    return 0;
}
```

```assembly
... // code still continues
```

Figure 1.5: A language-processing system

[Figure from Compilers: Principles, Techniques, & Tools, 2007]
A “Hello World” program

- Compile directly to binary
  - `cc hello.cpp -o hello`
- Content of hello looks like that

```cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
  std::cout << "Hello, World!\n; return 0;
```

# Figure 1.5: A language-processing system

[Figure from Compilers: Principles, Techniques, & Tools, 2007]
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. C++ compilers
5. A “Hello World” program
6. Setting up a development environment
7. Basic terms & concepts
Calling the compiler by hand is ‘wasteful’

- **Makefile**
  - A textfile containing rules that describe how to call the compiler
  - `make` processes the Makefile
    - Reads and identifies the rules
    - Executes them on-demand
  - Flexible and powerful
  - One Makefile per project
  - Hard to write for complex tasks
  - You see what’s going on
    - Nothing is hidden under the carpet

- **Integrated Development Environment (IDE)**
  - Handles the project and corresponding source files for you
  - Handles calls to the compiler
  - Often more pleasant than Makefiles
  - Will find syntax errors on-the-fly
  - More complex tasks are painful
    - Lack of control
    - Hides complexity

- I often use a combination of both!
Makefiles

- Using the compiler `by hand` is fiddly
- Use files containing the compiler commands
  - Makefile
    - Contains executable `targets`
    - Consist of a bunch of declarative rules
    - Processed by `make`
    - Flexible
    - Easy to use
    - Hard to write
      - There are books on `make`
  - Use the Makefiles from [https://www.hni.uni-paderborn.de/swt/lehre/cppp](https://www.hni.uni-paderborn.de/swt/lehre/cppp)

- Makefile:
  ```make
  PROGNAME := hello_world
  CC := g++
  FLAGS := -std=c++14
  FLAGS += -Wall
  all: main.cpp
      $(CC) $(FLAGS) *.cpp -o $(PROGNAME)
  clean:
      rm -f $(PROGNAME)
  ```

- hello.cpp:
  ```cpp
  #include <iostream>
  using namespace std;
  int main() {
      cout << "Hello, World!\n";
      return 0;
  }
  ```
Integrated Development Environment (IDE)

- Eclipse for C & C++
  - Full IDE
  - Windows / Linux / Mac

- Visual Studio Code
  - Compact editor
  - Windows / Linux / Mac

- Use what feels best for you
  - Depending on your programming level and experience

[Figures taken from eclipse.org and code.visualstudio.com]
Set up a development environment

- Set up an development environment?
  - I will provide a virtual machine
  - Password: cpppws1718
  - Ubuntu 16.04, ~12 GB (sorry)
  - Ships with everything that is needed!

- Remark on compiler errors
  - Errors are the default case
  - Don’t panic and read them
  - Read them carefully
  - Google will help
  - So does stack overflow (a programming forum)

```cpp
#include <iostream>
int main() {
    cout << "Hello, World!\n";
    return 0;
}
```
Contents

1. Organizational matter
2. Course outline
3. History of the C++ language
4. C++ compilers
5. A “Hello World” program
6. Setting up a development environment
7. Basic terms & concepts
Primitive / build-in data types

- Boolean types
  - `bool`
  - Can hold `true` or `false`

- Character types
  - `char`

- Integer types
  - `int`
  - Modifiers and sizes (integer types only)
    - `signed` and `unsigned`
    - `short`, `long`, `long long`

- Floating point types
  - `float`
  - `double`
  - `long double`

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in bits</th>
<th>Format</th>
<th>Value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>8</td>
<td>signed (one's complement)</td>
<td>-127 to 127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed (two's complement)</td>
<td>-128 to 127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 255</td>
</tr>
<tr>
<td>Integral</td>
<td>16</td>
<td>± 3.27 \times 10^4</td>
<td>-3,276,673 to 3,276,677</td>
</tr>
<tr>
<td></td>
<td></td>
<td>± 2.14 \times 10^9</td>
<td>-2,147,483,647 to 2,147,483,647</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned</td>
<td>0 to 6,553 \times 10^4</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>± 9.22 \times 10^{18}</td>
<td>-9,223,727,036,854,775,807 to 9,223,727,036,854,775,807</td>
</tr>
<tr>
<td></td>
<td></td>
<td>± 1.84 \times 10^{19}</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>± 1.7 \times 10^{308}</td>
<td>min subnormal: ± 4.940,656,456,412 \times 10^{-324}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min normal: ± 2,225,073,858,507,201,4 \times 10^{-308}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>max: ± 1.797,693,134,862,315,7 \times 10^{308}</td>
<td></td>
</tr>
</tbody>
</table>

[Figure taken from Wikipedia]
Integer encoding

- unsigned char
  - 1 byte = 8 bit
- Dual number encoding with unsigned

```
1 0 1 1 0 0 1 1
```

Decimal value:

\[ 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \]
\[ = 128 + 16 + 2 + 1 = 179 \]
Integer encoding

- **signed char** or **char**
  - 1 byte = 8 bit
- Two’s complement encoding with **signed** or as default

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

- Highest bit encodes sign
- Other bits encode value
- Here: sign bit 1, number is negative: take two’s complement (negate and add 1)

<table>
<thead>
<tr>
<th>-</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Take complement

Add one

Decimal value: $1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 8 + 4 + 1 = 77 \rightarrow -77$
Floating point number encoding

IEEE-754 single-precision binary floating-point format

value \( = (-1)^{\text{sign}} \times \left( 1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} \right) \times 2^{e_{127}} \)

IEEE-754 double-precision binary floating-point format

value \( = (-1)^{\text{sign}} \left( 1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{e_{1023}} \)

Remark

- Use double as default
- Floating point numbers are not distributed equidistant

[Figures from Wikipedia]
Comments in C++

- Comments tell other people what your code does
- Comments tell you what your code does
- Code can be hard to understand

Examples

- `// a single-line comment`
- `/*
   A multi-line
   comment
   */`
- `/* ... */ ... * ... */ this is wrong`
Integer literals in C++

- 100 // int decimal
- 123456 // int decimal
- 5L // long, decimal
- 123u // unsigned int, decimal
- 777uL // unsigned long, decimal
- -020 // int, octal
- 0x1fff // int, hexadecimal
- 0x1ffful // unsigned long, hexadecimal
Character literals in C++

- 'A'  // character A
- '*'  // symbol *
- '\0'  // end of a string
- '\n'  // new line
- '\t'  // tabulator
- '\''  // apostroph
- '\\'  // backslash

String literals in C++

- "This is a string literal!"  // a string literal
  - More on strings later
Floating-point literals in C++

-9.876 // double
123.456E-7 // double
1e12 // double
.001 // double
1.23f // float
1.23L // long double
Defining variables in C++

- Variables have a
  - Type
  - Name
  - Optionally an initial value

```
int i = 42;
int j;
int k = 10, l = 42, m;
double d = 1;
double e;
double f = 1.23456;
float g = 12.5f;
float h = 42.13;
char c = 'A';
char c[] = "A string";
char *c = "Another string";
char x = -10;
unsigned int ui = 123;
unsigned int uj = -13; // DON'T!!!
```

- Initialize your variables, unless you know what you are doing!
Variables in C++

- **unsigned int** `uj = -13;  // DON'T!!!**
  - Dangerous
  - Integer overflow

- C++ is famous for undefined behavior
  - C++ standard allows undefined behavior in some situations
    ```cpp
    int i;
    int j = i + 42;
    ```
  - Anything can happen
  - Depends on the compilers implementation
  - Why?
    - Allowing that, compilers can produce faster machine code
Variables in C++

- **auto** Keyword
  - Automatic type deduction
  - Compiler finds the correct type
  - Use `typeid(x).name()` to check in doubt
    - You need `#include <typeinfo>`
  - Always be verbose
    - If type name gets ‘too long’ or type is obvious use `auto`
- What type is x?
  - `auto x = 13L;` // long
  - `auto x = 1.2345;` // double

```cpp
#include <vector>
// C++98 style 😊
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
for (std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    cout << *it << endl;
}
//using C++11 ilist & range for-loop
std::vector<int> w = {1, 2, 3};
for (auto i : w) {
    cout << i << endl;
}
```
Just for fun

- `int a = 1;`
- `int b(2);`
- `int c{3};`
- `int d = {4};`
- `auto i = 5;`
- `auto j(6);`
- `auto k{7};`
- `auto l = {8};`
# IOstreams

- `#include <iostream>`
  - Part of the STL
  - Content lives in namespace `std`
- Use `std::` or `using namespace std;`
- Important variables
  - `cin` standard input stream
  - `cout` standard output stream
  - `cerr` standard error stream
  - `clog` general information
  - `<<` and `>>` are streaming operators defined on the stream variables

## Example

```cpp
#include <iostream>

using namespace std;

int main() {
    int i = 0;
    cout << "Enter an integer: " ;
    cin >> i;
    cout << "The value of i is: " << i << "\n";
    return 0;
}
```
# Essential UNIX commands

[https://www.tjhsst.edu/~dhyatt/superap/unixcmd.html](https://www.tjhsst.edu/~dhyatt/superap/unixcmd.html)

<table>
<thead>
<tr>
<th>Command</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>ls</strong></td>
<td><strong>ls</strong></td>
<td>Lists files in current directory</td>
</tr>
<tr>
<td></td>
<td><strong>ls -aIF</strong></td>
<td>List in long format</td>
</tr>
<tr>
<td>2. <strong>cd</strong></td>
<td>cd tempdir</td>
<td>Change directory to tempdir</td>
</tr>
<tr>
<td></td>
<td>cd ..</td>
<td>Move back one directory</td>
</tr>
<tr>
<td></td>
<td>cd ~/dhyatt/web-docs</td>
<td>Move into dhyatt's web-docs directory</td>
</tr>
<tr>
<td>3. <strong>mkdir</strong></td>
<td>mkdir graphics</td>
<td>Make a directory called graphics</td>
</tr>
<tr>
<td>4. <strong>rmdir</strong></td>
<td>rmdir emptydir</td>
<td>Remove directory (must be empty)</td>
</tr>
<tr>
<td>5. <strong>cp</strong></td>
<td>cp file1 web-docs</td>
<td>Copy file into directory</td>
</tr>
<tr>
<td></td>
<td>cp file1 file1.bak</td>
<td>Make backup of file1</td>
</tr>
<tr>
<td>6. <strong>rm</strong></td>
<td>rm file1.bak</td>
<td>Remove or delete file</td>
</tr>
<tr>
<td></td>
<td>rm * .tmp</td>
<td>Remove all file</td>
</tr>
<tr>
<td>7. <strong>mv</strong></td>
<td>mv old.html new.html</td>
<td>Move or rename files</td>
</tr>
<tr>
<td>8. <strong>more</strong></td>
<td>more index.html</td>
<td>Look at file, one page at a time</td>
</tr>
<tr>
<td>9. <strong>lpr</strong></td>
<td>lpr index.html</td>
<td>Send file to printer</td>
</tr>
<tr>
<td>10. <strong>man</strong></td>
<td>man ls</td>
<td>Online manual (help) about command</td>
</tr>
</tbody>
</table>
And now?

- Quick demo of the environment and a how to write “Hello, World!” program
  1. VS Code + Makefile
  2. Eclipse C/C++
  3. How to get a C++ job?

- Password for the ccpp machine: ccppws1718
Recap

- Course outline
- What is C++?
- History of C++
- Compilers
- Hello, World!
- Build-in types
- Information encoding
- Variables
- IO streams

- Any questions?
Thank you for your attention

Questions?