Creative Coding II

01 Introduction and Setup

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Contents

- Introduction to C++ & openFrameworks
- Practical Examples
- Build Process and IDEs | Development Process
- Practical Examples
- Exercises

C++

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Why C++?

- Improve & expand on technical understanding
- C++ is a compiled & statically typed programming language
- C++ supports low-level (== close to the hardware) as well as high-level programming (== close to the human) concepts

Why C++?

- Improve & expand on design & code expressiveness

- C++ is a high-level programming language with a rich set of syntactic and semantic elements that support developers in
  - focusing on concept development
  - expressing ideas directly in code

Why C++?

- Improve & expand on general knowledge
- C++ is particularly well-known for its support of
  - writing very efficient (real-time) programs
  - object-oriented programming
  - generic programming

History of C++

- Bjarne Stroustrup started the development of C++ in 1979 at AT&T Bell Labs as an extension to the C language & first called it „C with classes“

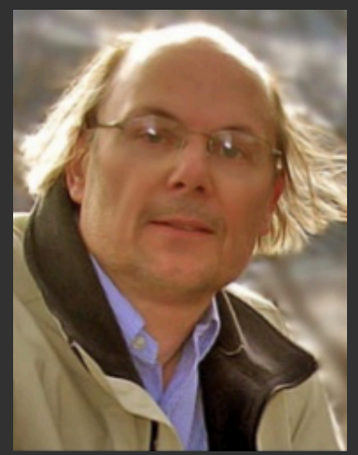
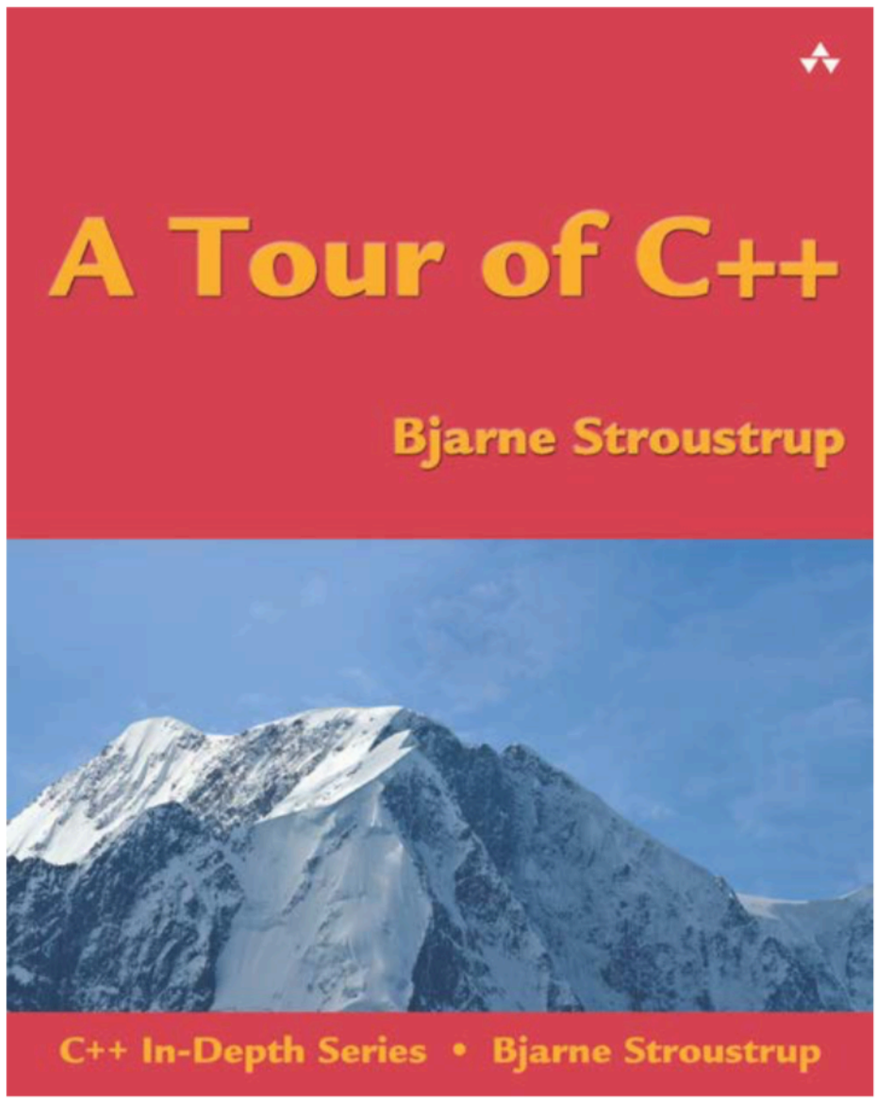


Image credit: Bjarne Stroustruop (2014): **The Essence of C++**. Morgan Stanley, Columbia University, Texas A&M University

- C++ is continuously being improved
- ANSI standard ensures portability
- Major versions C++98, C++11, C++14, …, C++23, C++26

What does C++ offer?

- C++ in Four slides
  - Map to hardware
  - Classes
  - Inheritance
  - Parameterized types

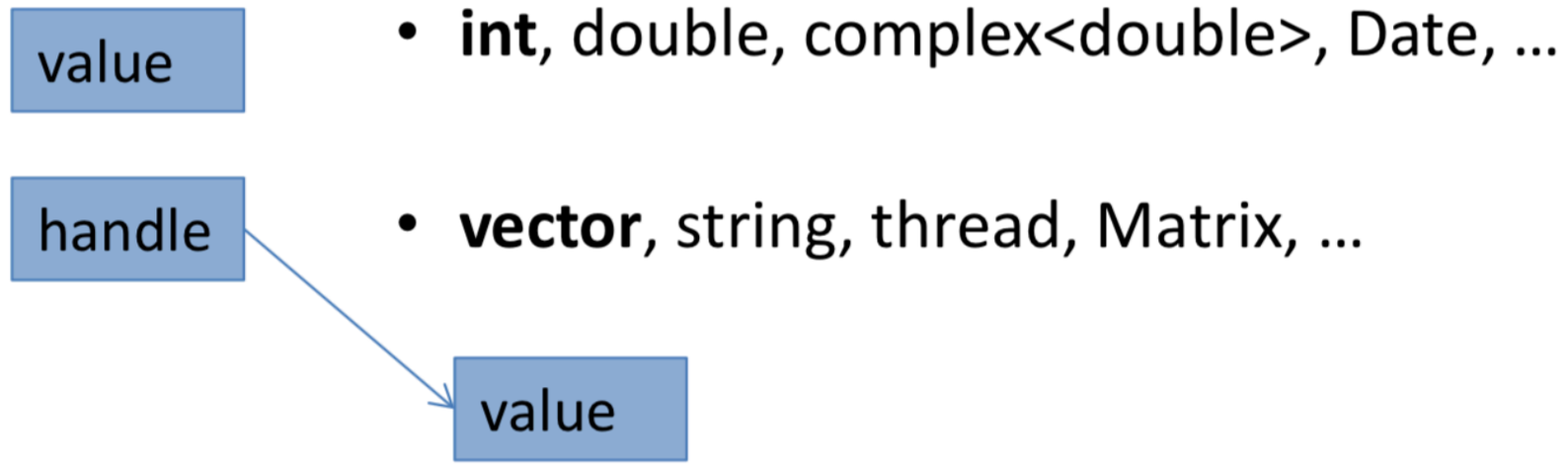


- If you understand **int** and **vector**, you understand C++
  - The rest is “details” (1,300+ pages of details)

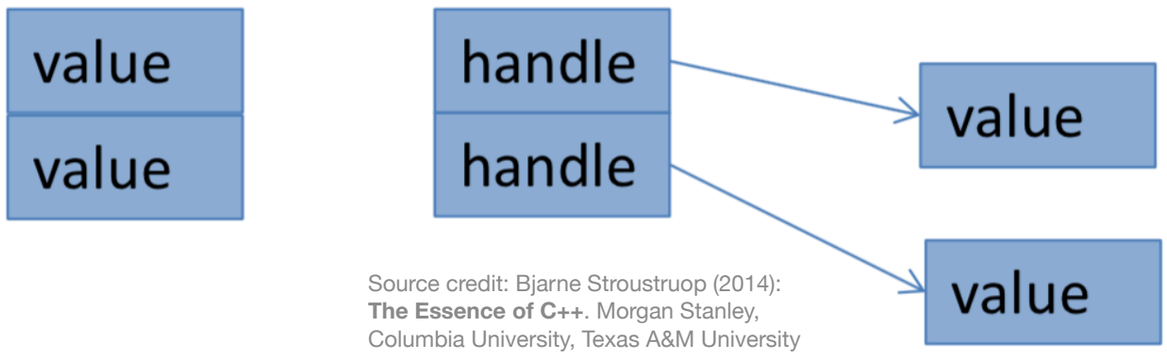
Source credit: Bjarne Stroustruop (2014): **The Essence of C++.** Morgan Stanley, Columbia University, Texas A&M University

Map to Hardware

- Primitive operations => instructions
  - +, %, ->, [], (), …



- Objects can be composed by simple concatenation:
  - Arrays
  - Classes/structs

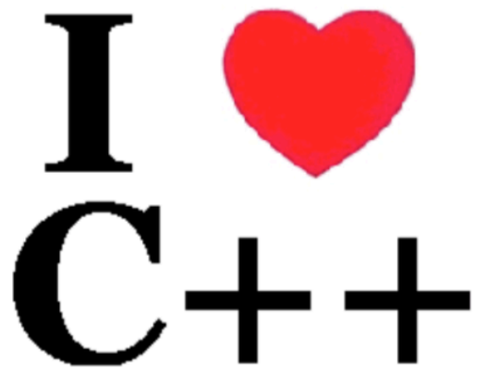


Source credit: Bjarne Stroustruop (2014): **The Essence of C++**. Morgan Stanley, Columbia University, Texas A&M University

Classes: Construction/Destruction

- From the first week of “C with Classes” (1979)

```
class X {                    // user-defined type
public:              // interface
        X(Something);    // constructor from Something
        ~X();                // destructor
        // …
private:             // implementation
        // …
};
```



“A constructor establishes the environment for the members to run in; the destructor reverses its actions.”

Source credit: Bjarne Stroustruop (2014): **The Essence of C++.** Morgan Stanley, Columbia University, Texas A&M University

Abstract Classes and Inheritance

- Insulate the user from the implementation

```
struct Device {                                  // abstract class
        virtual int put(const char*) = 0;        // pure virtual function
        virtual int get(const char*) = 0;
};
```

- No data members, all data in derived classes
  - "not brittle"
- Manipulate through pointer or reference
  - Typically allocated on the free store ("dynamic memory")
  - Typically requires some form of lifetime management (use resource handles)
- Is the root of a hierarchy of derived classes

Source credit: Bjarne Stroustruop (2014): **The Essence of C++**. Morgan Stanley, Columbia University, Texas A&M University

Parameterized Types and Classes

- Templates
  - Essential: Support for generic programming
  - Secondary: Support for compile-time computation

```
template<typename T>
class vector { /* … */ };                    // a generic type

vector<double> constants = {3.14159265359, 2.54, 1, 6.62606957E-34, };   // a use


template<typename C>
void sort (C& c) { /* … */ }                 // a generic function

sort(constants);                             // a use
```

Source credit: Bjarne Stroustruop (2014): **The Essence of C++**. Morgan Stanley, Columbia University, Texas A&M University

openFrameworks

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Why openFrameworks?

- openFrameworks is free, open source C++ framework that has been strongly influenced by the Processing environment

- openFrameworks targets easy development of real-time applications & is primarily oriented for use in creative and experimental projects

- openFrameworks takes care of creating a graphical window, listening for mouse and keyboard events, etc., so developers can start with expressing their ideas fairly quickly

Overview

- openFrameworks is highly extensible using **addons**
  - **ofxAddon** are open source & generally built by members of the openFrameworks community
  - However, many addons are not maintained on a regular basis

- openFrameworks is **cross-platform** (supporting OS X, Windows, Linux, iOS, Android & Linux ARM devices such as Raspberry Pi)

- Finally, it has a very friendly & active community

Download and Installation

- Go to the http://openframeworks.cc/download/ page
- Download the latest public release (v0.12.0) for your target OS
- You will download a *.zip file
- Choose a folder on your OS where you want to develop stuff
- Unzip the folder
- That’s it! Everything else is done with the IDE

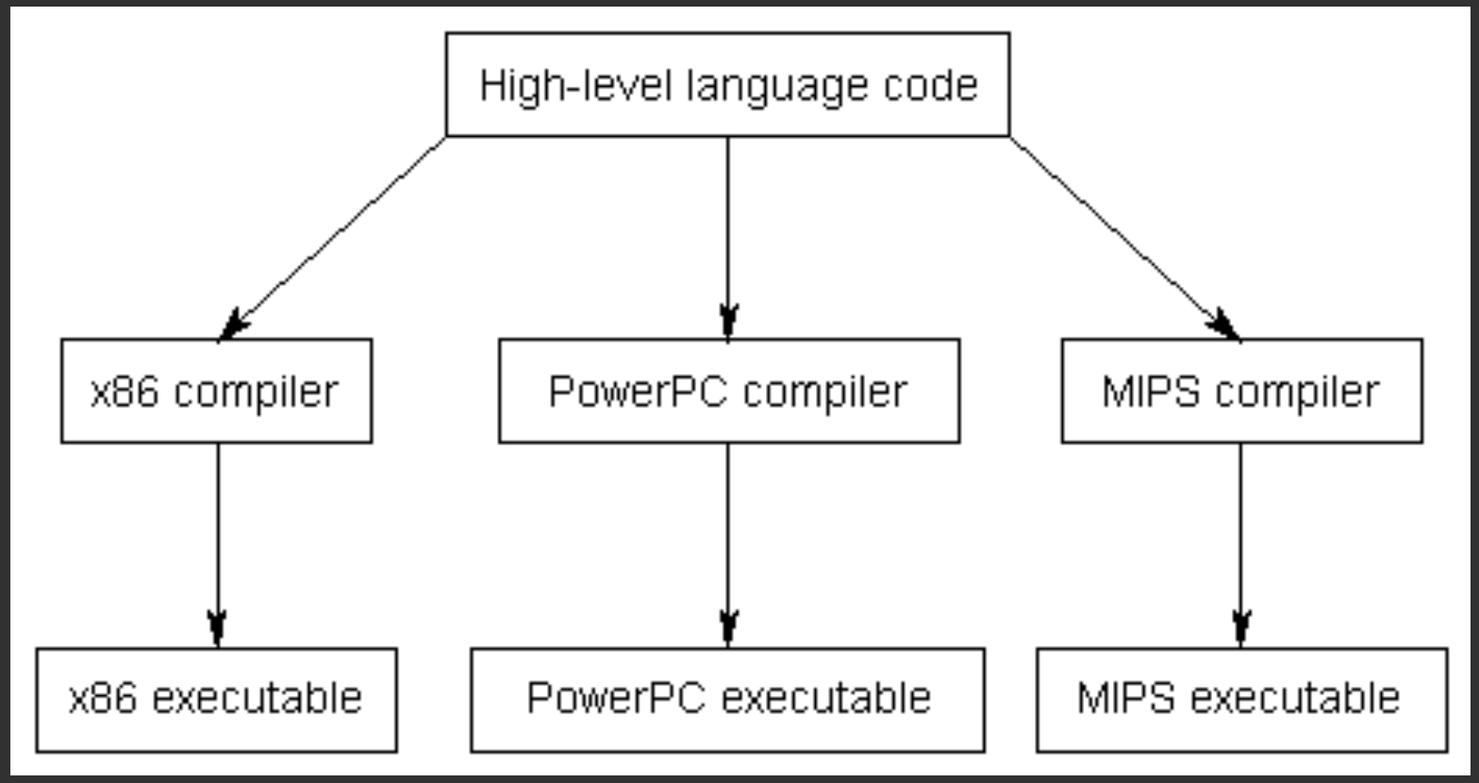
Practical Examples

The Build Process

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

The Build Process

- The build process takes care of turning source code into executable software programs written in machine code



simplified diagram

Image credit: http://www.learncpp.com | 0.2 — Introduction to programming languages

Program Execution

- Software programs („executables“) are essentially sets of instructions that tell the computer how to operate

- The instructions must be written in machine code for the computer to understand and execute them

- C++ is a compiled language which means it has to be explicitly translated into machine code
  => this is done during the build process

Machine Code

- Machine code is a very low-level programming language
- Each instruction set is composed of bits, binary numbers
  - For example: `10110000 01100001`

- This type of machine code is hard to work with for humans
- Also, the instruction sets are different per CPU
  - MacOS, Windows, Linux executables / binaries differ

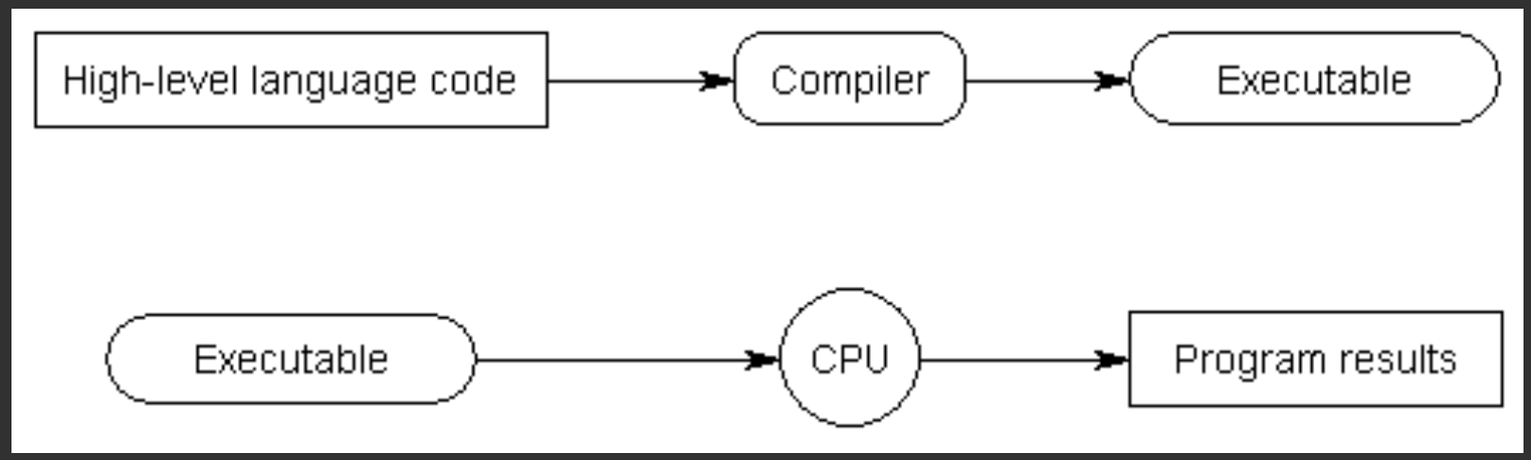
High-level Language

- High-level programming languages like C++ can be easily read and written by humans
  - For example: “`string name = \`Hello, world!\`;`“

- Compiled languages require compilers and linkers to translate the high-level language source code into machine code
- This process differs essentially from how programs written in interpreted languages can be executed and run

Compiled Language

- A compiled language (e.g. C++) has to be actively translated into machine code so that the program can be executed.
- A „loader“, which is a program provided by the specific platform, can then load and execute the executable



Interpreted Language

- An interpreted language (e.g. Python, JS, Java) uses an interpreter that does not compile source code in machine code but directly executes the instructions itself and is not necessarily optimized for the platform

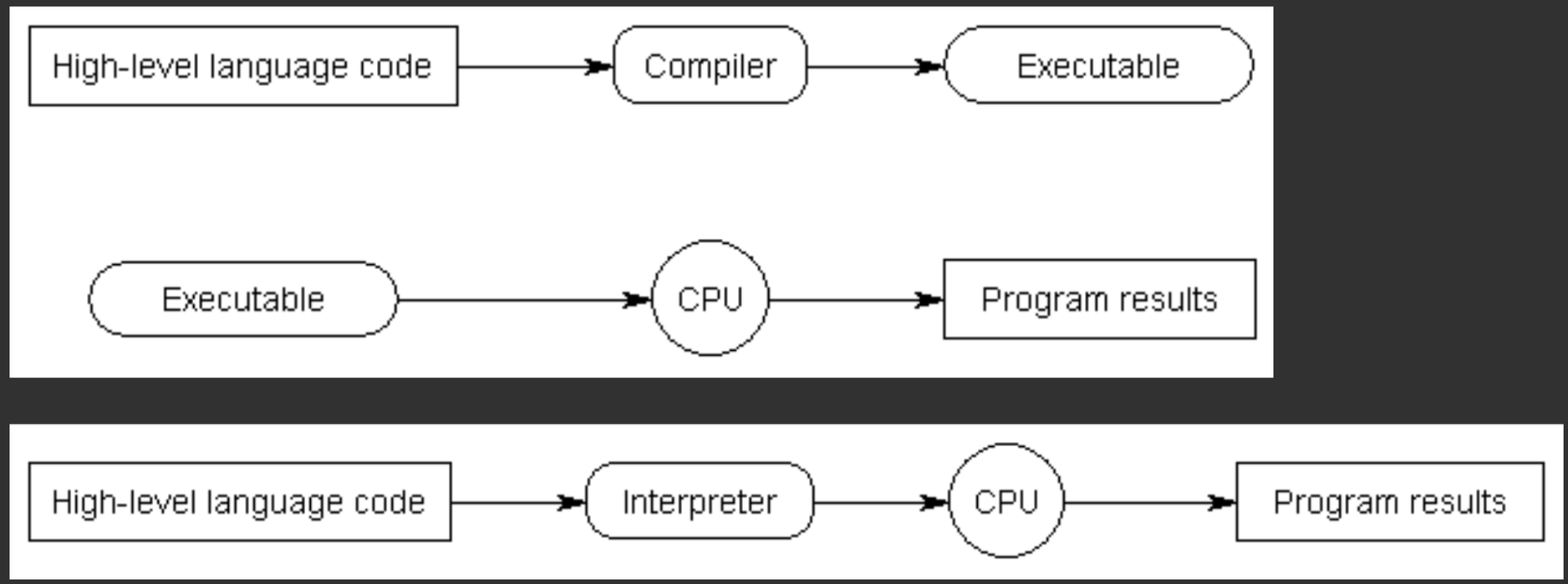
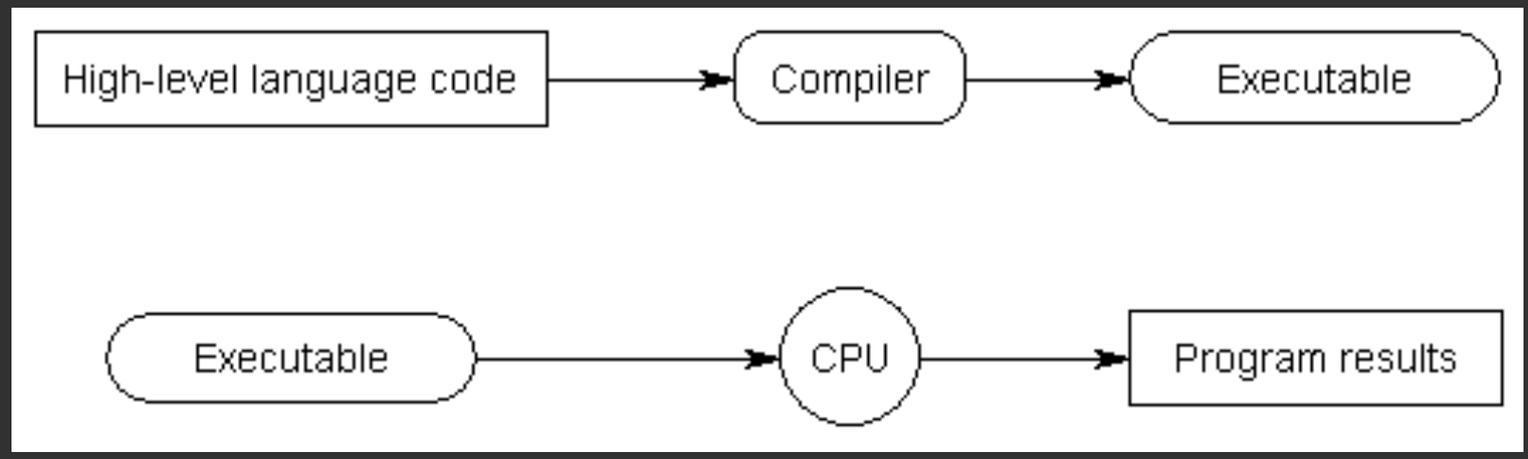


Image credit: http://www.learncpp.com | 0.2 — Introduction to programming languages

Compiler vs Interpreter

- Interpreted language programs are therefore generally still slower in program execution than compiled language programs that are executed directly by the CPU



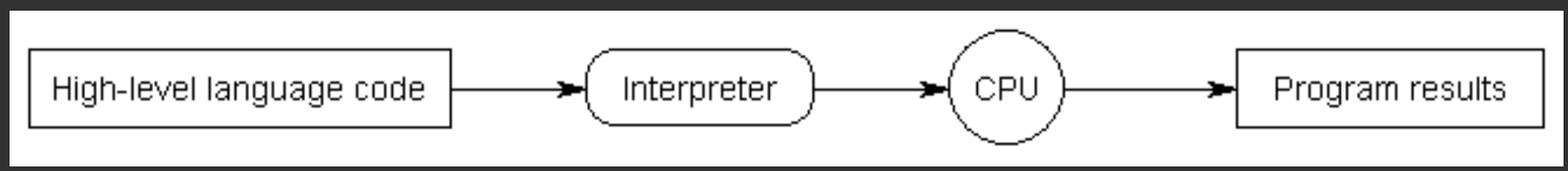


Image credit: http://www.learncpp.com | 0.2 — Introduction to programming languages

Development and Build Process

The build process is comprised of

- Preprocessing — generates enhanced source code
- Compilation — generates object files
- Linking — links all object files and generates the executable

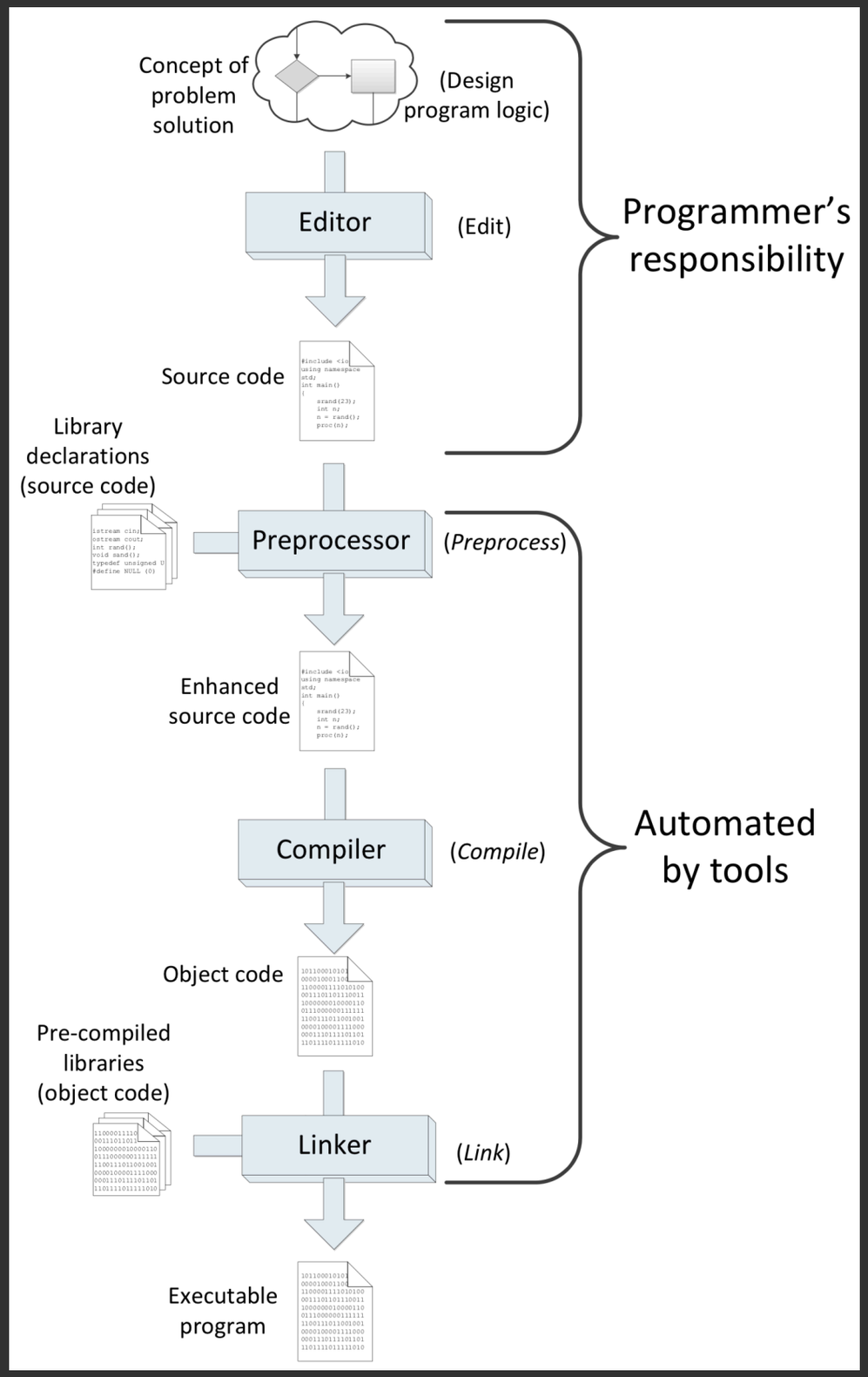
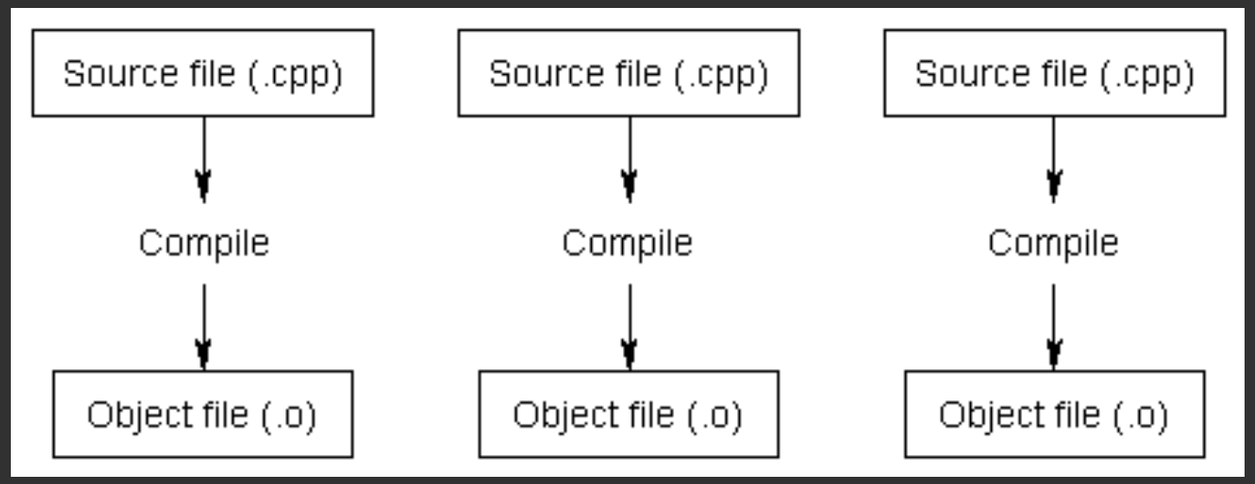
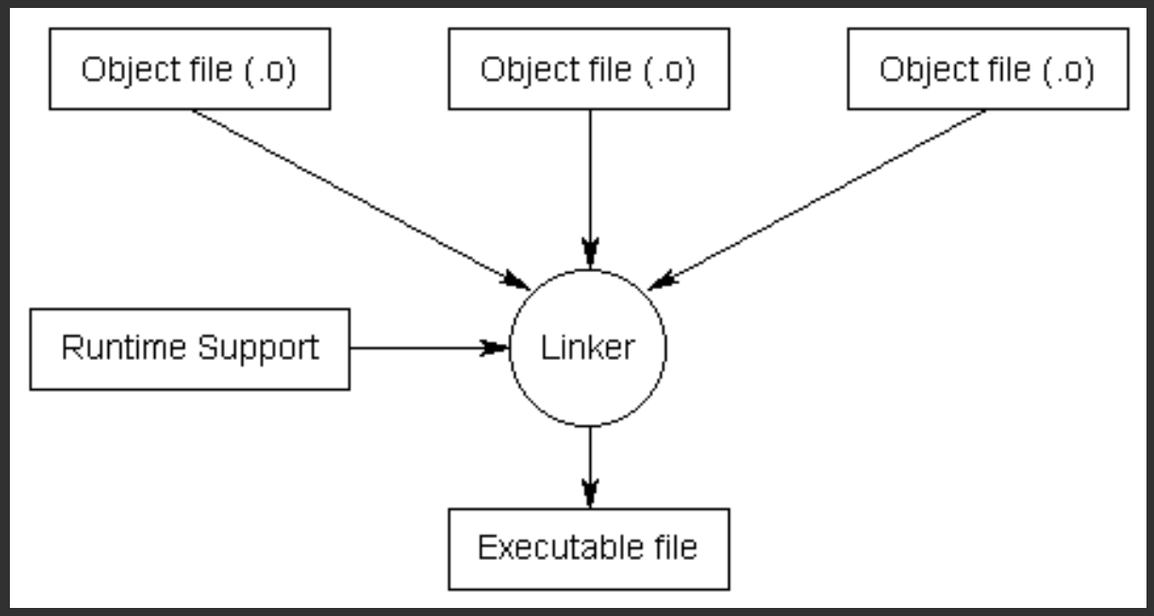


Image credit: Richard L. Halterman (2017): **Fundamentals of C++ Programming**. Online print.

Build Process Stages

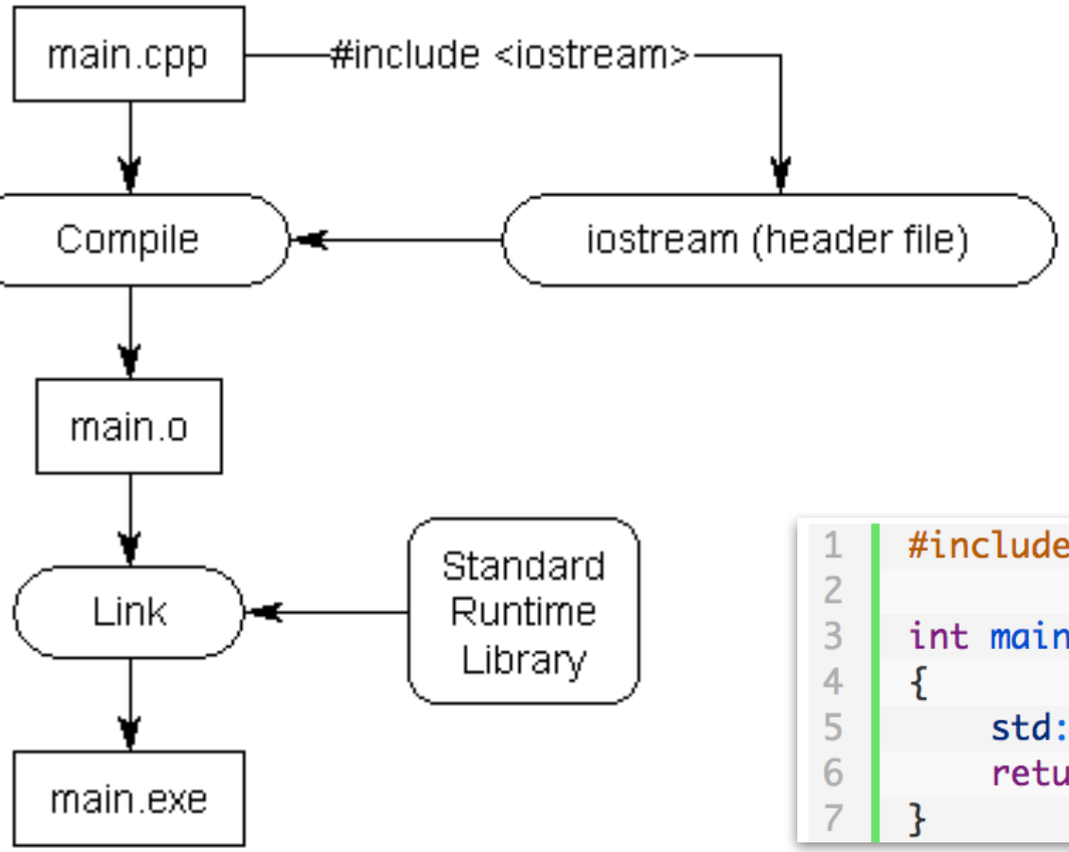
- The build process stages
  - preprocessing — generates enhanced source code
  - compilation — generates assembler files & then object files
  - linking — links all object files & generates the executable





Images credit: http://www.learncpp.com | 0.2 — Introduction to development

Header & Definition (cpp) Files

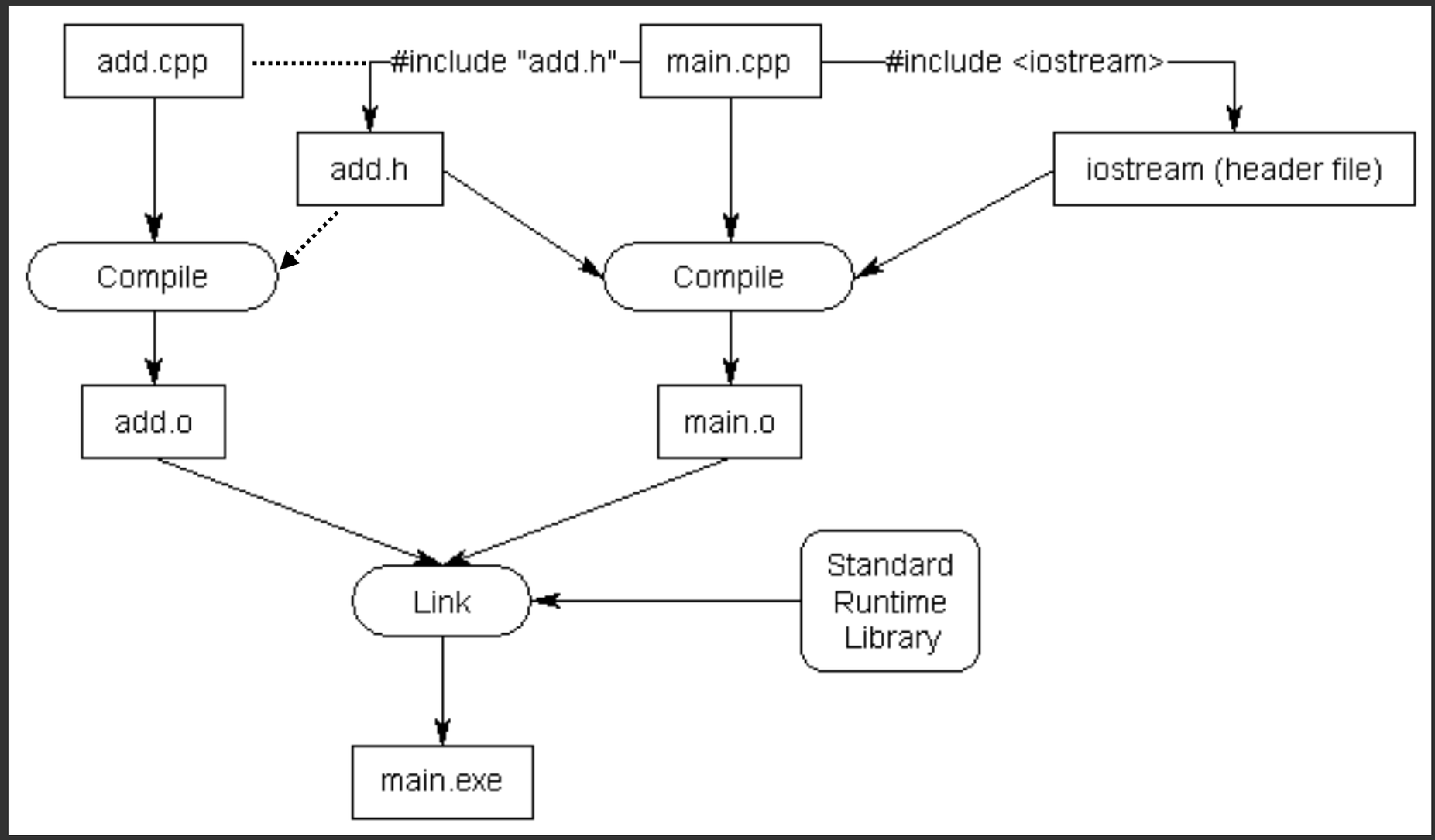



```cpp
#include <iostream>

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

Source: https://www.learncpp.com/cpp-tutorial/header-files/

Header & Definition (cpp) Files



Source: https://www.learncpp.com/cpp-tutorial/header-files/

Header & Definition (cpp) Files

- C++ code is written with an editor or IDE (integrated development environment) and saved to files of the following file extensions
  - source code definition files: *.cpp
  - source code header files: *.h

- The responsibility of the developer is to follow the language syntax & rules so that no errors occur during compilation — or the build process to be more precise

A Simple Example

- A first „Hello world!“ example
  - Line 1 — a preprocessor directive
  - Lines 3-7 — the main function
- Every C++-program requires a **main function**
- Main functions are usually stored in **main.cpp**

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello world!";
6      return 0;
7  }
```

A Simple Example

- Compiling and linking the program using the command line (indicated by "$"):
  - `$ g++ -o mainApp.exe main.cpp` | Windows
  - `$ g++ -o mainApp main.cpp` | MacOS

```cpp
#include <iostream>

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

Error Types

- Compiler errors occur during compilation
  - usually refer to syntax errors in the code
  - compiler error code is usually very helpful to identify and fix the compiler error
- Linker errors occur during linking of the object files and are often much harder to fix
  - usually refer to missing implementation details of the code like, i.e., missing function definitions or libraries (path errors)

Integrated Development Environment

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Working with an IDE

- IDE is short for **Integrated Development Environment**

- IDEs are rich programming environments that contain a compiler, a linker, a debugger, etc., as well as many features like code completion, syntax highlighting, version control, etc.

- Most prominent IDEs are
  - Microsoft Visual Studio Community for Windows
  - XCode for macOS

Windows

- MS Visual Studio Community is the free version version for developing C++ applications on Windows

- There is a specific openFrameworks plugin for visual studio that can be used instead of the Project Generator

- Download and install MS Visual Studio Community & follow this setup guide http://openframeworks.cc/setup/vs/

macOS

- XCode is the standard IDE for developing C++ applications on macOS

- XCode comes with Command line tools that need to be installed separately

- Download and install XCode & the command line tools follow this guide http://openframeworks.cc/setup/xcode/

Alternatives

- For working with openFrameworks, alternative IDEs are mainly
  - VisualStudioCode on all platforms
    https://code.visualstudio.com
  - Code::Blocks on Windows
    http://www.codeblocks.org
  - Qt Creator on all platforms (MacOS, Windows, Linux)
    https://www.qt.io/qt-features-libraries-apis-tools-and-ide/#ide

- Setup guides are available via the openframeworks.cc page

Important Terms

- A Project contains all files, resources, info to build one or more software products
  - *.xcodeproj (‘xcodeproject’ file on MacOS)
  - *.sln (‘solution’ file on Windows)
  - ‚Makefile’ and ‚make‘ command
- A Target is the actual executable file (or lib, DLL, etc.)
- A Product is the actual software product including one or more targets & being organized by the project (file)

Important Terms

- Build process terms
  - Build — preprocess, compile, link the sources
  - Run — (build), load & run the executable
  - Clean — remove all temporary (i.e., object) files that were generated during build process

- Build settings
  - Contain information about how the individual aspects of the build process should be performed

Important Terms

- Build configurations: Debug & Release Mode

- Debug mode provides additional debug information about your program that tell you about where & why something crashed
- It is particular useful during development but slows down the build & execution processes

- Release mode removes any additional debug information
- It creates a small & fast app & should be used after the development — for the **release candidate**

Bibliography

Bibliography

- Last access on websites: 19 May 2024
- Martin, Robert C. (2009): Clean Code. Upper Saddle River, NJ: Prentice Hall.
- Boccara, Jonathan: Fluent C++. Online blog. www.fluentcpp.com
- Stroustrup, Bjarne. Website. http://www.stroustrup.com
- https://learncpp.com