Scientific Programming with C++

USE OF 3RD PARTY LIBRARIES IN RESEARCH

Learning objectives

- After this lecture and related assignments, you will…
  - Have an idea of how source code turns into usable programs
  - Have an idea of how libraries work in C++
  - Be introduced to the inclusion of 3rd party libraries in your programs

Introduction: source code to computer programs (1/2)

- C++ projects consist of ”translation units” that must be compiled
  - Source code we wrote
  - Source code we included
- The compiler creates object code files (.obj) from translation units
  - Human-readable source code to machine-readable object code
  - In our project, just from our code
    - Code we included is already available as .obj (.o on Unix)
- The linker links together different (compiled) object code files
  - Allows the use of e.g. the C++ standard library
  - Creates the final executable file (or a library)

Introduction: source code to computer programs (2/2)

- Two programs involved
  - The compiler
    - Source code to object code
  - The linker
    - Connects object code files to create the executable
- Object code and executables are not portable on different operating systems

Libraries

- What is a library?
  - Source code that others have written for our use
  - An archive of object files created by the linker
- Examples
  - C++ standard library
    - You have been using this
  - 3rd party libraries
- The C++ standard library has many great general functionalities
  - But research work often requires very specialized classes
    - Only found in 3rd party libraries

Using libraries

- If you have the full source code
  - You can compile the source code of the library with your source code
- If you have the library as a header and static or shared/dynamic library file
  - You need to link the library to your program’s object code
  - Static libraries are packed into your executable
    - Results in one big file
  - Dynamic libraries are referenced to your executable at build time
    - But used at program run time
    - Results in a not-so-big executable and a separate dynamic library file (which may be shared by multiple different programs)

Why use 3rd party libraries?

- Solving research problems often involves complex and chained calculations
  - Solving the activation levels of muscles during walking from trajectories of motion capture markers
    - Creation of a subject-specific musculoskeletal model, calculation of joint angles, calculation of joint moments based on the angles, estimation of muscle activity given the muscle parameters…
  - Predicting future values in a time series based on previous known values
    - Choosing and training a prediction model, testing on an independent test set to evaluate prediction accuracy, predicting from real-world data
  - And so forth
- Implementing the computations involved yourself would take a long time
  - Using 3rd party libraries with existing algorithms for the problem saves time

Installing 3rd party libraries

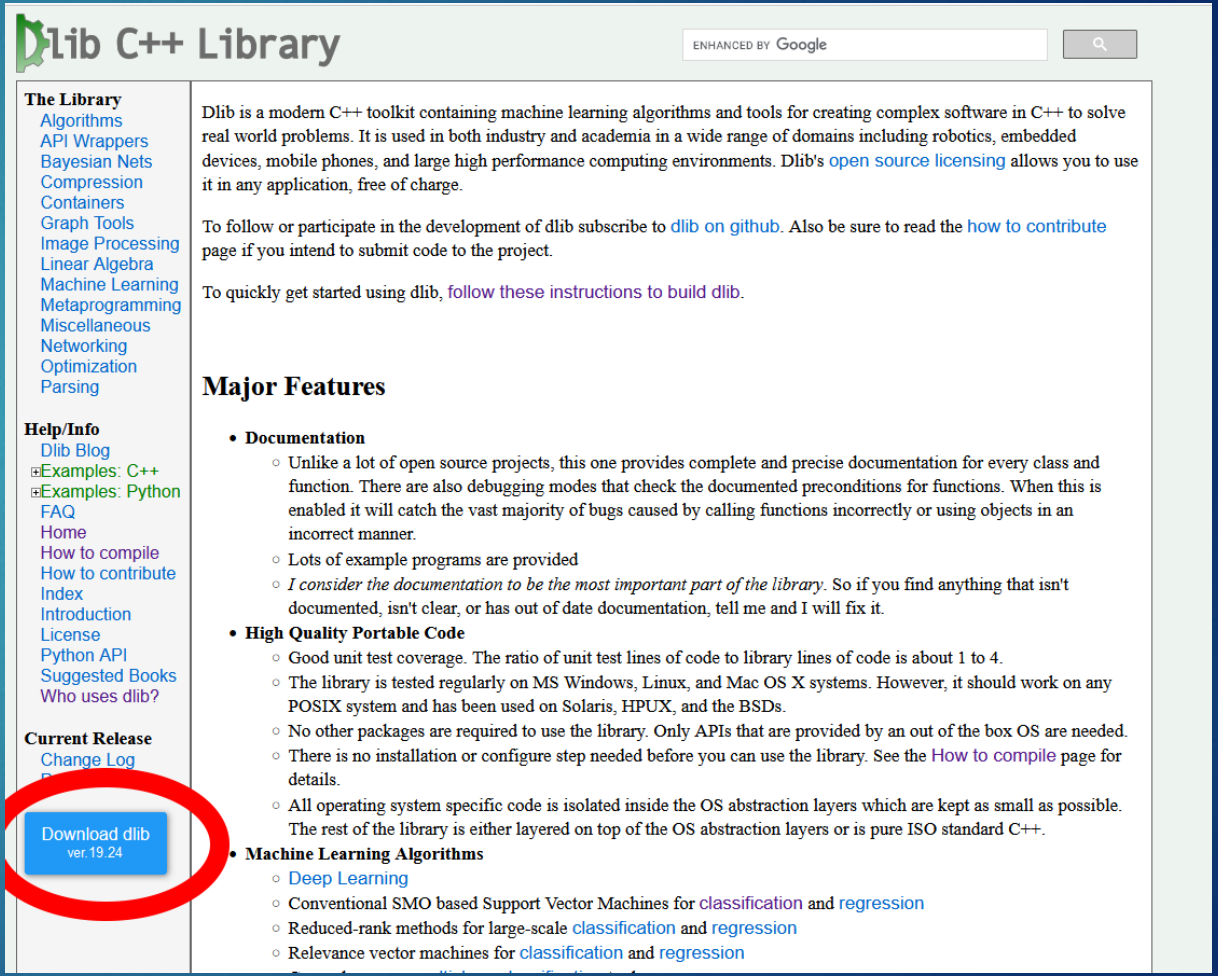
- Somewhat inconvenient as there is no universal dependency manager like in Python
  - For interesting discussion, see for example
    - "Why is it such an abysmal pain to use libraries in C++ compared to pretty much anything else?"
      - https://www.reddit.com/r/cpp/comments/ix9n1u/why_is_it_such_an_abysmal_pain_to_use_libraries/
    - "C++ libraries are impossible to install"
      - https://www.reddit.com/r/cpp/comments/ozg80h/c_libraries_are_impossible_to_install/
- Adapting a 3rd party library usually involves
  - 1. Downloading the library
  - 2. Configuring your project to be able to find the library and link relevant library files (often with helper programs)
  - 3. Troubleshooting, changing some configuration parameters and trying again

dlib

- We will use dlib (http://dlib.net/)
  - “Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems.“
- Relatively easy to install for many headers
  - No need to explicitly link libraries
  - No need for external programs to configure projects in IDE
  - “In most cases, to use this library all you have to do is extract it somewhere, make sure the folder *containing* the dlib folder is in your include path, and finally add dlib/all/source.cpp to your project. “
  - http://dlib.net/compile.html, “Compiling C++ examples without CMake”

Configuring dlib on Windows & Visual Studio

- Download dlib from the website
- Extract the zipped package on your hard drive



The Library
Algorithms
API Wrappers
Bayesian Nets
Compression
Containers
Graph Tools
Image Processing
Linear Algebra
Machine Learning
Metaprogramming
Miscellaneous
Networking
Optimization
Parsing

Help/Info
Dlib Blog
⊞Examples: C++
⊞Examples: Python
FAQ
Home
How to compile
How to contribute
Index
Introduction
License
Python API
Suggested Books
Who uses dlib?

Current Release
Change Log

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

To follow or participate in the development of dlib subscribe to dlib on github. Also be sure to read the how to contribute page if you intend to submit code to the project.

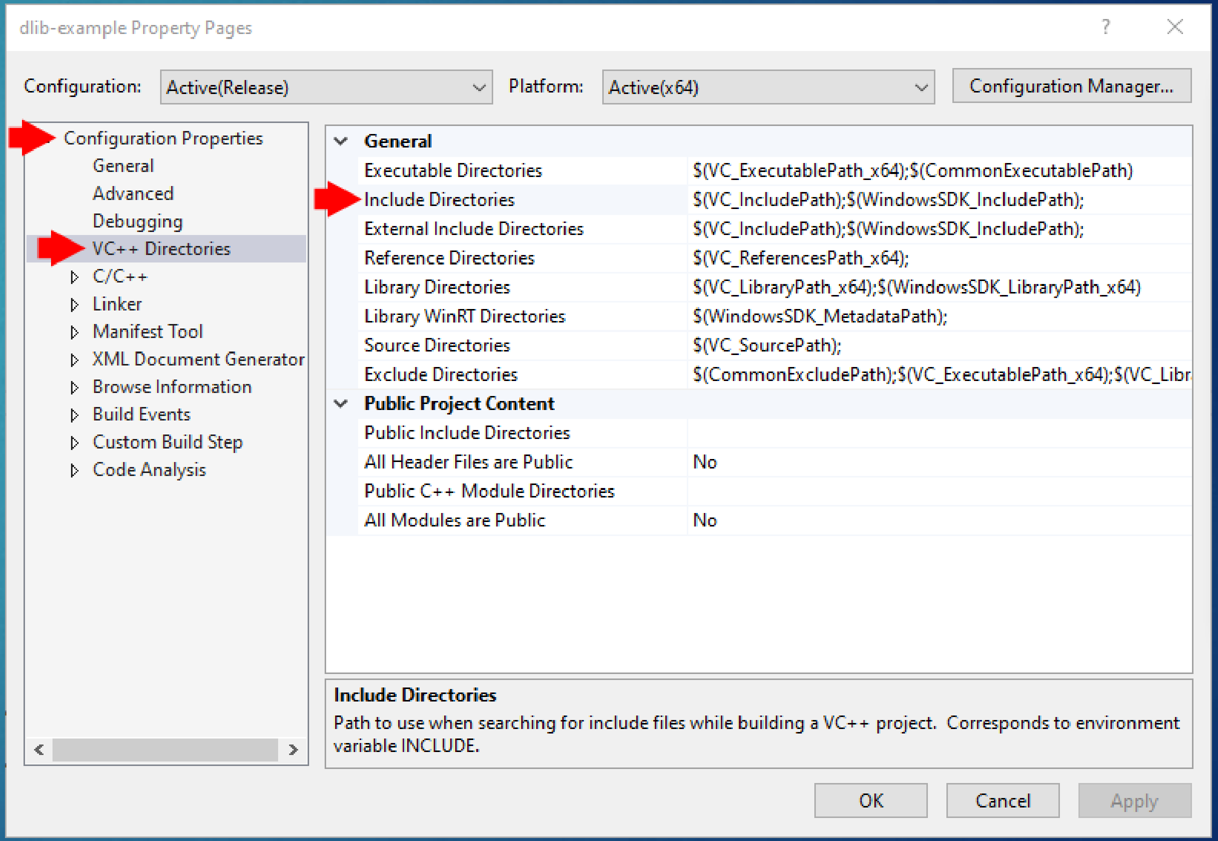
To quickly get started using dlib, follow these instructions to build dlib.

Major Features

- **Documentation**
  - Unlike a lot of open source projects, this one provides complete and precise documentation for every class and function. There are also debugging modes that check the documented preconditions for functions. When this is enabled it will catch the vast majority of bugs caused by calling functions incorrectly or using objects in an incorrect manner.
  - Lots of example programs are provided
  - *I consider the documentation to be the most important part of the library.* So if you find anything that isn't documented, isn't clear, or has out of date documentation, tell me and I will fix it.
- **High Quality Portable Code**
  - Good unit test coverage. The ratio of unit test lines of code to library lines of code is about 1 to 4.
  - The library is tested regularly on MS Windows, Linux, and Mac OS X systems. However, it should work on any POSIX system and has been used on Solaris, HPUX, and the BSDs.
  - No other packages are required to use the library. Only APIs that are provided by an out of the box OS are needed.
  - There is no installation or configure step needed before you can use the library. See the How to compile page for details.
  - All operating system specific code is isolated inside the OS abstraction layers which are kept as small as possible. The rest of the library is either layered on top of the OS abstraction layers or is pure ISO standard C++.
- **Machine Learning Algorithms**
  - Deep Learning
  - Conventional SMO based Support Vector Machines for classification and regression
  - Reduced-rank methods for large-scale classification and regression
  - Relevance vector machines for classification and regression

Configuring dlib on Windows & Visual Studio

- Download dlib from the website
- Extract the zipped package on your hard drive
- Create a new project in Visual Studio
- Set include directories
  - "make sure the folder *containing* the dlib folder is in your include path"



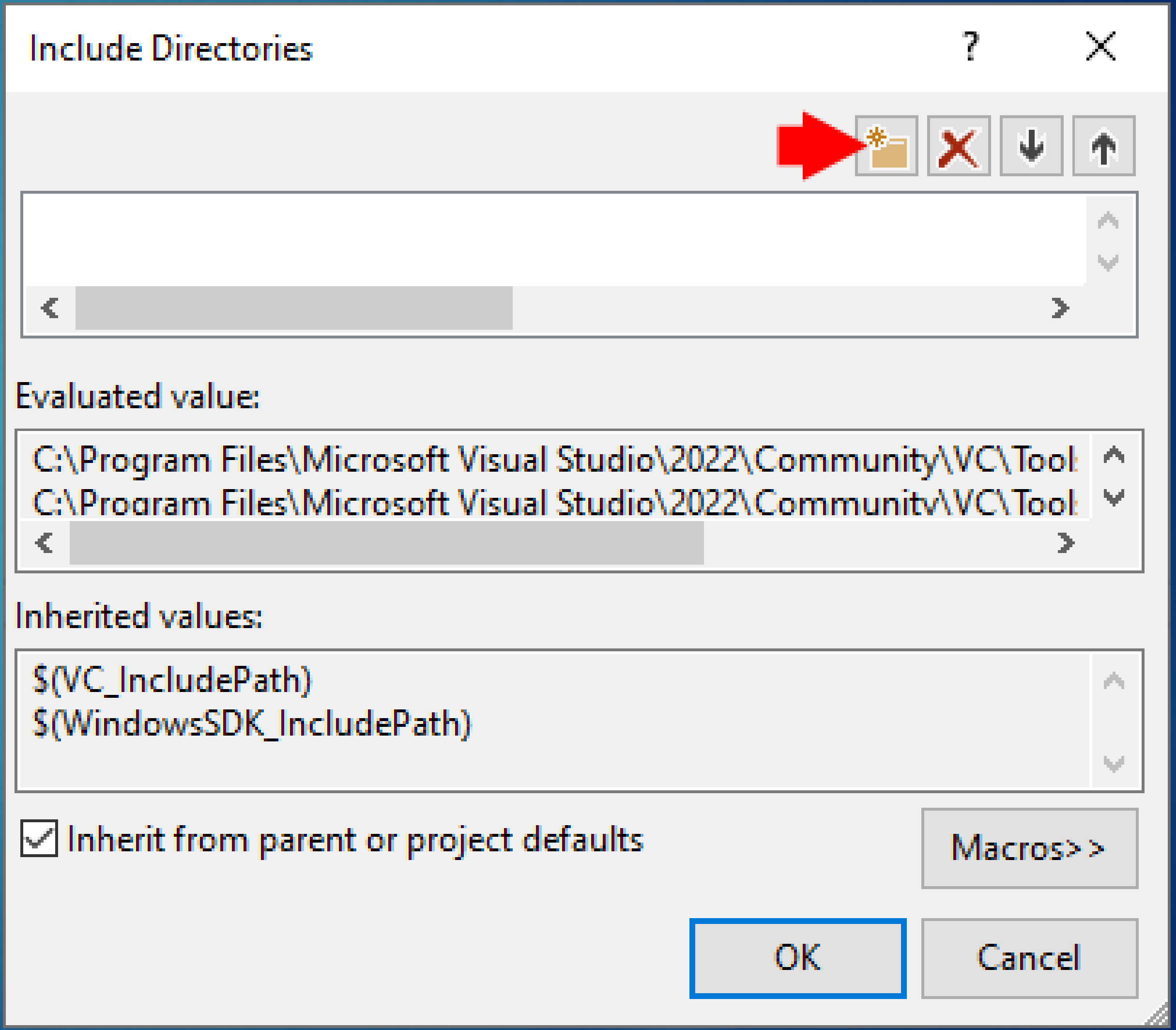
Configuration: Active(Release) Platform: Active(x64)

General	
Executable Directories	$(VC_ExecutablePath_x64);$(CommonExecutablePath)
Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
External Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x64);
Library Directories	$(VC_LibraryPath_x64);$(WindowsSDK_LibraryPath_x64)
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(CommonExcludePath);$(VC_ExecutablePath_x64);$(VC_Libr
Public Project Content	
Public Include Directories	
All Header Files are Public	No
Public C++ Module Directories	
All Modules are Public	No

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

Configuring dlib on Windows & Visual Studio

- Download dlib from the website
- Extract the zipped package on your hard drive
- Create a new project in Visual Studio
- Set include directories
  - "make sure the folder *containing* the dlib folder is in your include path"



Include Directories

Evaluated value:

```
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tool
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tool
```

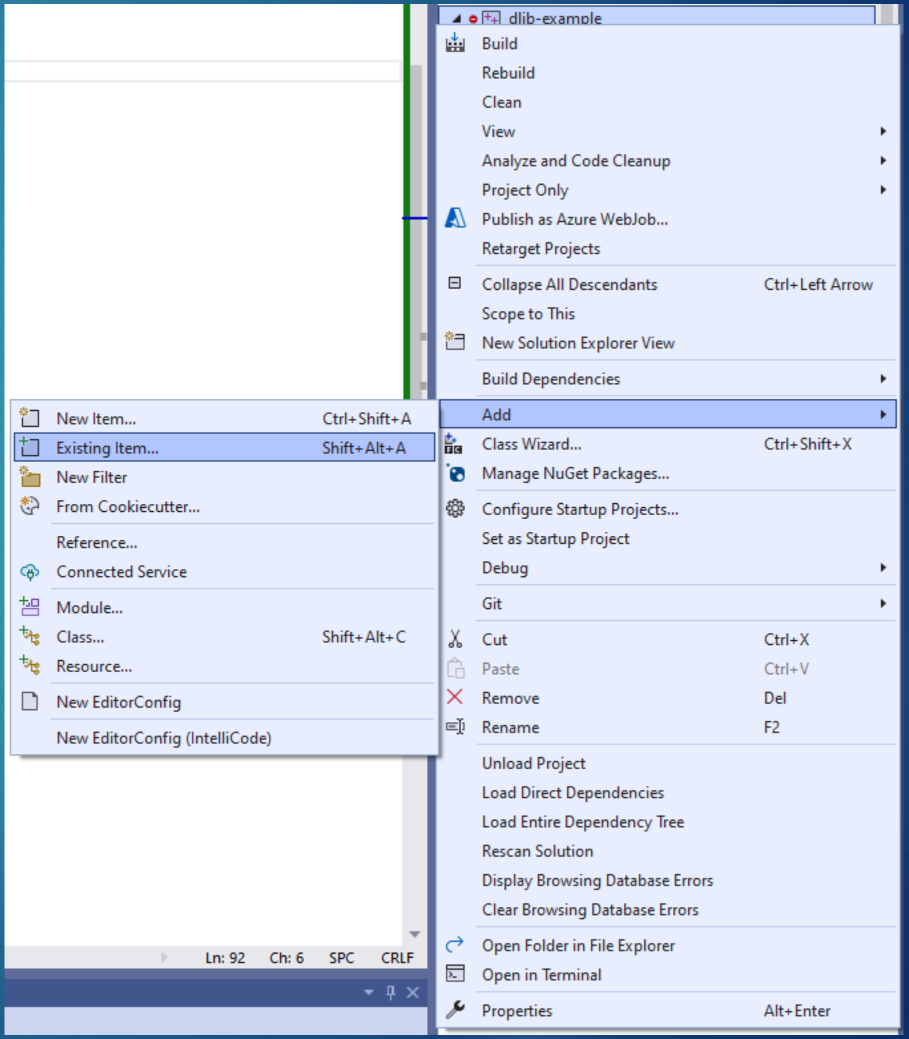
Inherited values:

```
$(VC_IncludePath)
$(WindowsSDK_IncludePath)
```

☑ Inherit from parent or project defaults

Configuring dlib on Windows & Visual Studio

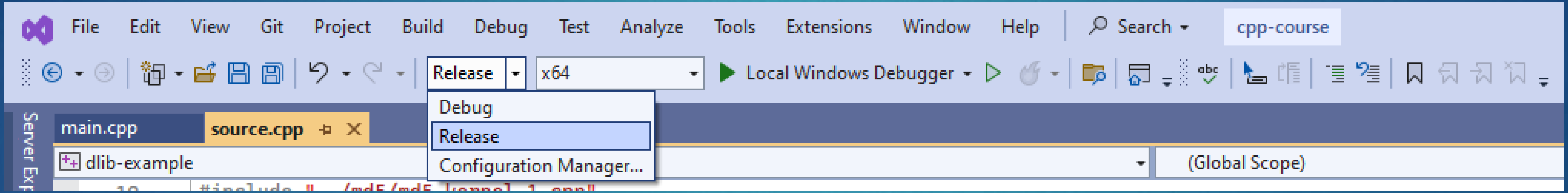
- Download dlib from the website
- Extract the zipped package on your hard drive
- Create a new project in Visual Studio
- Set include directories
  - “make sure the folder *containing* the dlib folder is in your include path”
- Add dlib/all/source.cpp to your project
  - Project -> Add -> Existing Item…



Configuring dlib on Windows & Visual Studio

- If building the project fails, make sure solution configuration is “Release”

fatal error C1128: number of sections exceeded object file format limit: compile with /bigobj



Configuring dlib on Windows & Visual Studio

- In your own .cpp file, include the relevant headers

```
#include "C:/Users/StudentMcStudentson/Downloads/dlib-19.24/dlib/mlp.h"
#include "C:/Users/StudentMcStudentson/Downloads/dlib-19.24/dlib/clustering.h"
#include "C:/Users/StudentMcStudentson/Downloads/dlib-19.24/dlib/matrix.h"
```

- You should now be able to use dlib in your own code
- If not, refer to the documentation (http://dlib.net/compile.html)