Welcome and Getting Started

Lecture 1

Hartmut Kaiser

https://teaching.hkaiser.org/spring2024/csc3380/

Abstract

- Today, we'll outline the aims for this course and present a rough course plan. After some introduction, we'll start reviewing C++ by looking at the simplest possible C++ program and outline how it can be made into running code.

What is Computer Science

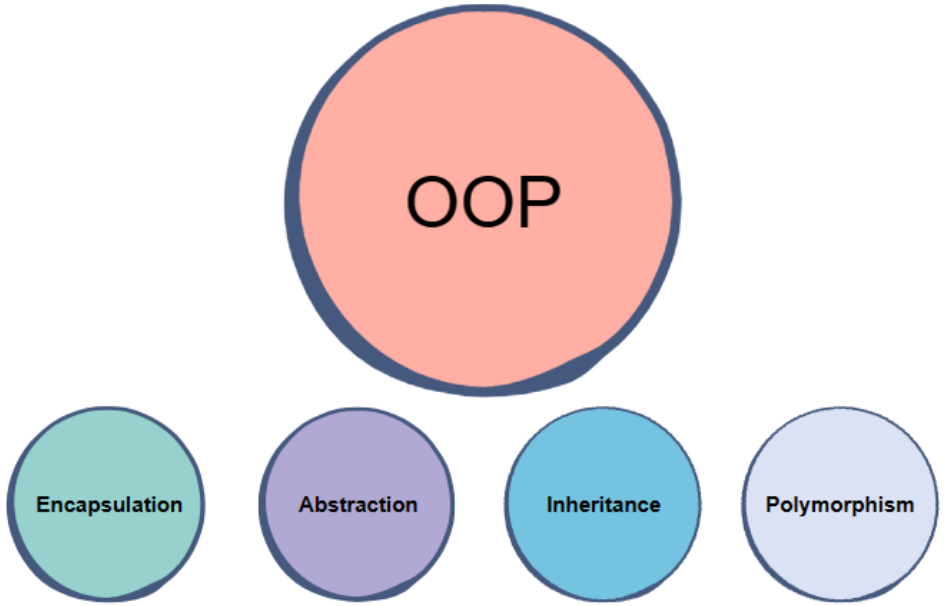
Techniques for managing complexity

- Black box abstraction
  - Primitive objects
    - Primitive procedures and primitive data
  - Means of combination
    - Procedure composition (functions) and compound data (types)
  - Means of abstraction
    - Procedure (function) definition - Algorithms
    - Data abstraction (types) – Data structures
  - Capturing common patterns
    - Higher order functions
    - Data as procedures, procedures as data

Techniques for managing complexity

- Conventional Interfaces
  - Generic operations
  - Large scale structure and modularity
  - Object oriented programming
  - Operations on aggregates

Object Oriented Programming

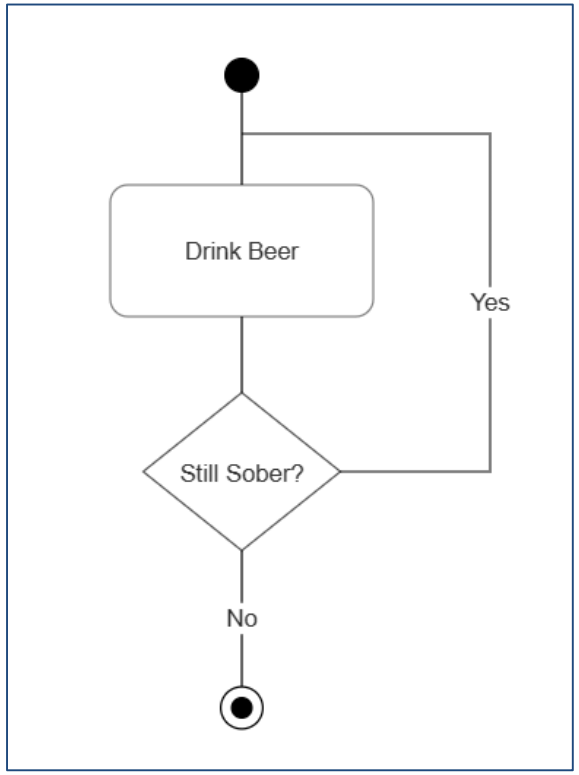


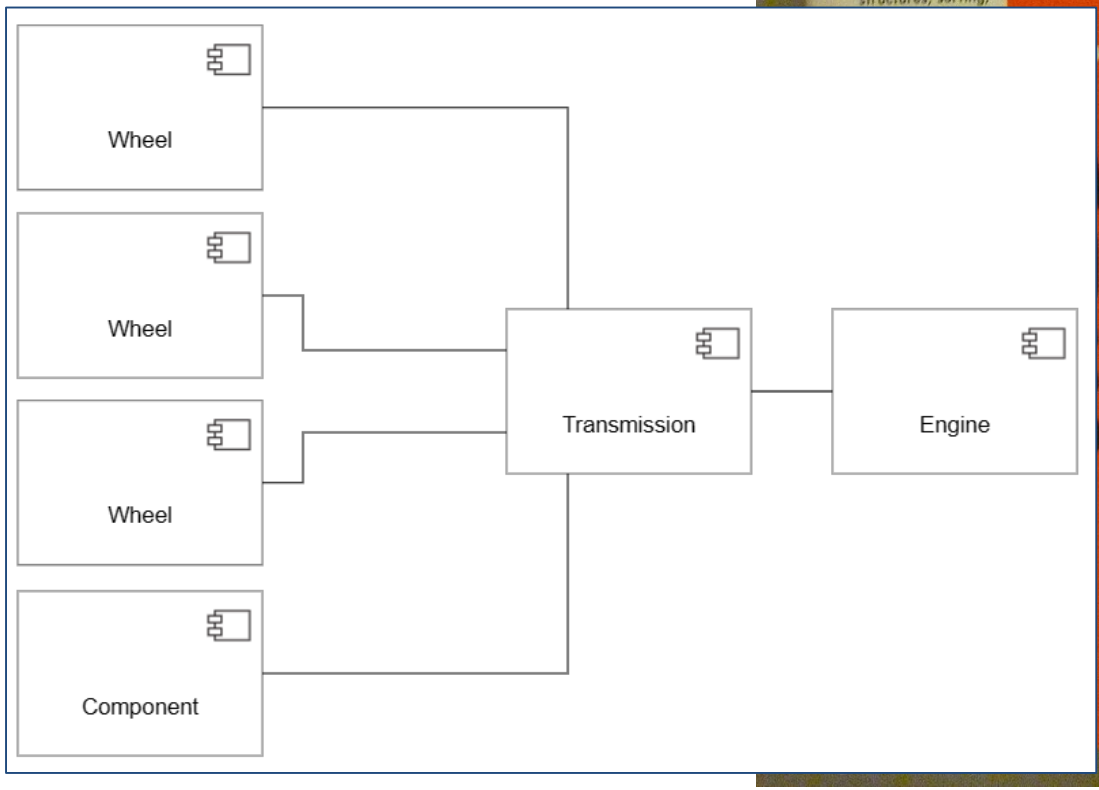
- The four pillars of object-oriented programming are:
- **Encapsulation:** containing information in an object, exposing only selected information
- **Abstraction:** only exposing high-level public methods for accessing an object
- **Inheritance:** child classes inherit data and behaviors from the parent class
- **Polymorphism:** many methods can do the same task

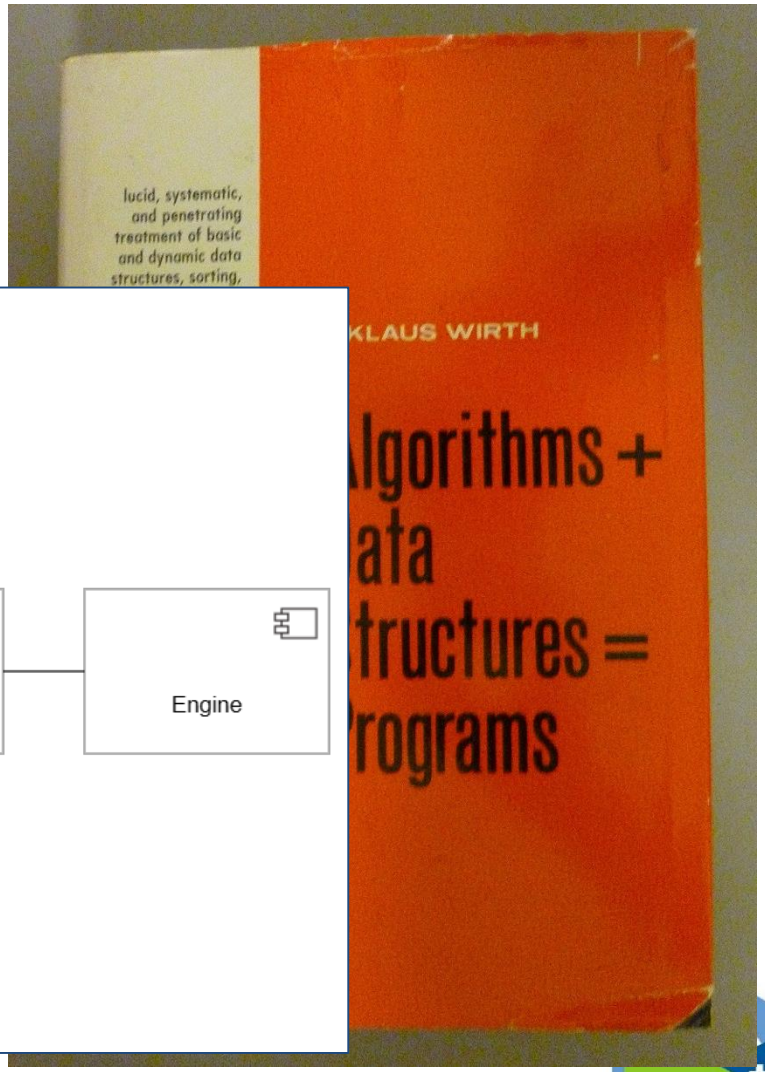
See also: [What is object-oriented programming? OOP explained in depth]

Course Overview

- Programs == Algorithms + Data structures







Building Abstractions

- Computer Science is a discipline that
  - Builds Abstractions
  - By managing complexity
  - With the goal of being able to write correct and performant code

Admin & Organizational

Admin & Organizational

- Congrats
  - Why I like programming
- Course:
  - Depth first introduction, C++ Standard Library, C++ Data structures and algorithms
  - https://teaching.hkaiser.org
  - hkaiser@cct.lsu.edu
  - Discord server: https://discord.gg/62U9ceEbN8
- Reading:
  - Stepanov's From Mathematics to Generic Programming
  - Koenig's Accelerated C++
  - Stroustrup's Programming - Principles and Practice Using C++
- Homework, project, quizzes, grading, honesty
  - Use Github classroom for assignments
  - Use VSCode as a tool

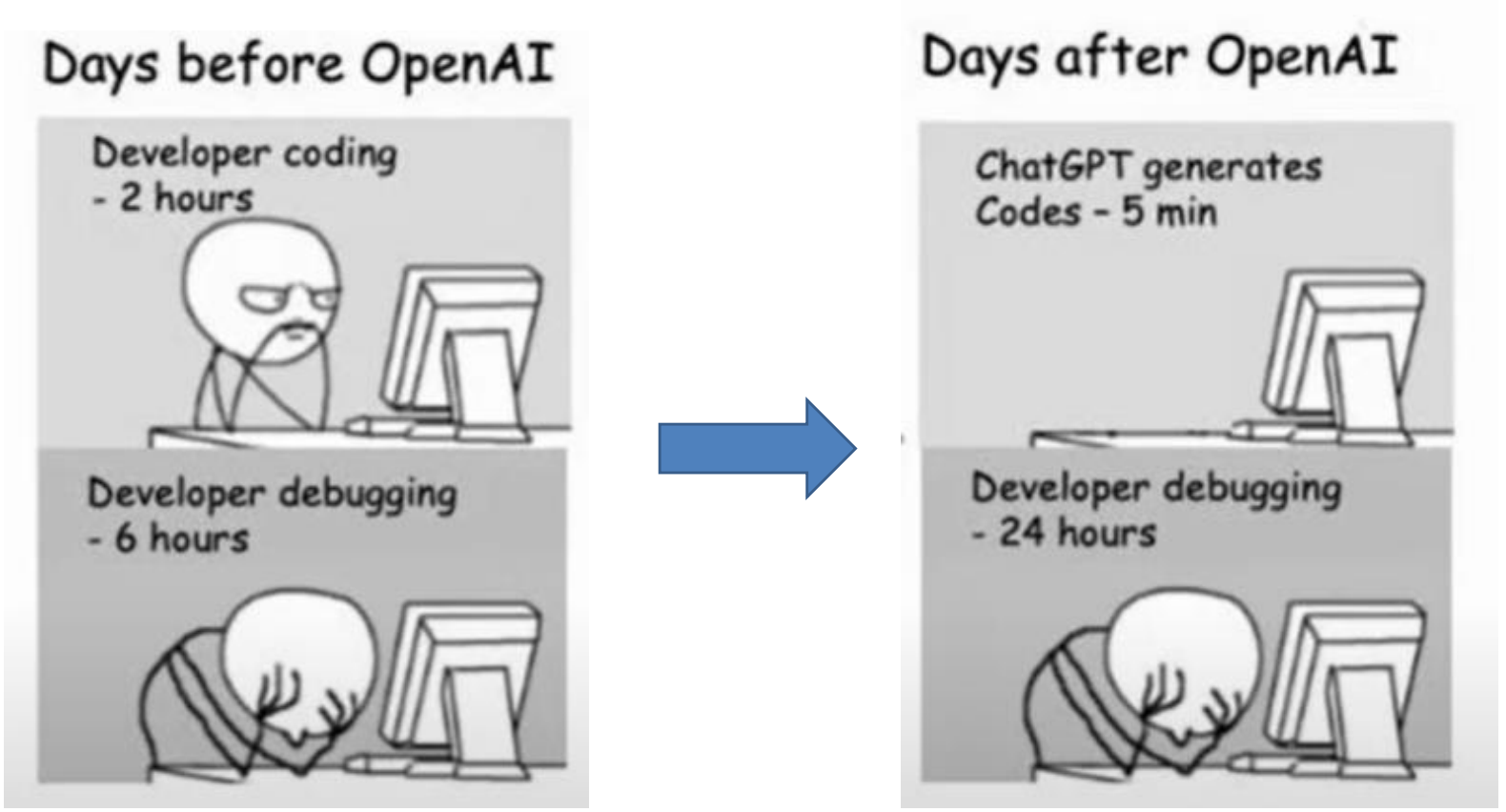
Ground Rules

- All information about the course: teaching.hkaiser.org
  - Following that site is expected and will positively impact overall grade
- Lectures
  - Attendance is expected and will positively impact overall grade
- Assignments
  - Up to five individual assignments focusing on writing code
    - Use of (standard C++) algorithms and (standard C++) data structures
  - Assignment 0 has been posted (due: January 29, 11:59 pm, no extension)
- Project:
  - Collaborative project of six students per group
    - Start forming your group now, form groups, select leader, send email
    - Remaining students will be arbitrarily combined into teams by January 31
  - During project presentations attendance will be checked
- All assignments and the project are hosted on GitHub (github.com) and managed through GitHub classroom (classroom.github.com)
  - You will need to create accounts on both

Honesty

- The LSU *Code of Student Conduct* defines plagiarism in Section 5.1.16:
  - "Plagiarism is defined as the unacknowledged inclusion of someone else's words, structure, ideas, or data. When a student submits work as his/her own that includes the words, structure, ideas, or data of others, the source of this information must be acknowledged through complete, accurate, and specific references, and, if verbatim statements are included, through quotation marks as well. Failure to identify any source (including interviews, surveys, etc.), published in any medium (including on the internet) or unpublished, from which words, structure, ideas, or data have been taken, constitutes plagiarism;"
- Plagiarism will not be tolerated and will be dealt with in accordance with and as outlined by the LSU Code of Student Conduct : https://www.lsu.edu/saa/students/codeofconduct.php

ChatGPT?


Days before OpenAI
Developer coding
- 2 hours
Developer debugging
- 6 hours
Days after OpenAI
ChatGPT generates
Codes - 5 min
Developer debugging
- 24 hours

ChatGPT

- The goal for this course is to train our brains, not ChatGPT
  - So, do yourself a favor and don't use it
  - There are too many software 'developers' out there who copy & paste their way to the next paycheck
- However, if you do use it:
  - Never use anything without carefully reviewing it
    - Assume what you got is wrong! Prove to yourself it is correct!
  - The skill of reading (and understanding) code becomes more important than ever
- Whatever you do, remember:
  - It's plagiarism if you submit the same code as your neighbor
  - No matter where you got it from, be it Google, ChatGPT, or your neighbors computer

Rough course outline

- Books:
  - Alexander A. Stepanov and Daniel E. Rose: From Mathematics to Generic Programming, ISBN 978-0321942043
  - Andrew Koenig, Barbara Moo: Accelerated C++ - Practical Programming by Example, ISBN 0-201-70353-X
  - Bjarne Stroustrup: Programming – Principles and Practice using C++, ISBN 978032154372
  - Stanley Lippmann, Josée Lajoie, Barbara E. Moo: C++ Primer 5th Ed., ISBN 0321714113
- Part I: The basics
  - Egyptian Multiplication
  - OOP and Generic Programming
- Part II: Organizing Programs and Data, Overview of Standard Template Library
  - Sequential data containers
  - Using library algorithms
  - Associative data containers
- Part III: Defining new types
  - Managing memory and low level data structures
  - Defining abstract data types
- Part IV: Object oriented programming
  - Inheritance and dynamic binding
  - Automatic memory management

Rough course outline

- Throughout
  - Program design and development techniques
  - Software development cycle, source code control system use
  - C++ language features
  - Background and related fields, topics, and languages

C++! But Why?

Why C++ ?

- You can't learn to program without a programming language
- The purpose of a programming language is to allow you to express your ideas in code
- C++ is the language that most directly allows you to express ideas from the largest number of application areas
- C++ is the most widely used language in engineering areas
  - http://www.research.att.com/~bs/applications.html
- Our society runs on software
  - Large parts of that software directly or indirectly rely on C/C++
    - Windows, Linux, Office Suite, Adobe Products, …
    - Self driving cars, industry equipment, …
    - Internet

Why C++ ?

- C++ is precisely and comprehensively defined by an ISO standard
  - C++17! Now also C++23!
  - And that standard is almost universally accepted
- C++ is available on almost all kinds of computers
- Programming concepts that you learn using C++ can be used fairly directly in other languages
  - Including C, Java, C#, and (less directly) Fortran
- Last but not least: C++ jobs are one of the best paid jobs available

A First C++ Program

```
// Our first C++ program

#include <iostream>

int main()     // main() is where a C++ program starts
{
       std::cout << "Hello, world!" << std::endl;
                        // output the 13 characters
                        // Hello, world! followed by a new line
       return 0;        // return a value indicating success
}

// quotes delimit a string literal
// NOTE: “smart” quotes “ ” will cause compilation problems.
//        so make sure your quotes are of the style " "
// \n is a notation for a new line
```

A deeper look

- Expressions
  - Compute something, yields result, may have side effects
  - Operands and operators – types!

```
std::cout << "Hello, world!" << std::endl;
```

- Scope
  - Part of the program in which a name has its meaning
    - Global scope
    - Namespace scope
    - Block scope

Details

- Program structure
  - Free form except string literals, #include, comments
- Types
  - Data structures and their operations
  - Built in and user defined
- Functions
  - Code structure
  - Helps isolating (abstracting) sub-functionalities
- Namespaces
  - Grouping related names

Details

- Special character literals
  - `\n` newline character
  - `\t` horizontal tabulator
  - `\b` backspace character
  - `\"` same as " but does not terminate string
  - `\'` same as ' but does not terminate character literal
  - `\\` same as \ but does not give special meaning to next character
- Definitions and headers
- The main() function
- Braces and semicolons
- Output

Hello, world!

- "Hello world" is a very important program
  - Its purpose is to help you get used to your tools
    - Compiler
    - Program development environment
    - Program execution environment
  - Type in the program carefully
    - After you get it to work, please make a few mistakes to see how the tools respond; for example
      - Forget the header
      - Forget to terminate the string
      - Misspell return (e.g. retrun)
      - Forget a semicolon
      - Forget { or }
      - …

Hello world

- It's almost all "boiler plate"
  - Only `std::cout << "Hello, world!\n"` directly does anything
- That's normal
  - Most of our code, and most of the systems we use simply exist to make some other code elegant and/or efficient
  - "real world" non-software analogies abound
- "Boiler plate," that is, notation, libraries, and other support is what makes our code simple, comprehensible, trustworthy, and efficient.
  - Would you rather write 1,000,000 lines of machine code?
- This implies that we should not just "get things done"; we should take great care that things are done elegantly, correctly, and in ways that ease the creation of more/other software:
  - Style Matters!

Number of Unique Elements

- Let's start with a sequence of integers:

```
int a[] = { 1, 3, 1, 4, 1, 5 };
```

- How many unique integers do we have?
  - Simple solution:

```
#include <iostream>
#include <set>

int main() {
    std::set<int> set_of_ints(a, a + 6);
    std::cout << set_of_ints.size() << std::endl;
}
```

- This solution is correct! But … very slow! Why?

Equality vs. Ordering

- `std::set` is implemented using Red-Black-Trees, which according to textbooks is the best way of implementing it
  - Will do `O(n log(n))` comparison operations, but with a large constant coefficient
  - Has to re-sort for each insertion without need
- It might appear, that finding unique elements does not require ordering, it just requires equality
  - But, actually we need a search or find
  - Equality gives us linear search `O(n)`, while sorting gives us binary search `O(log(n))` so we can find much, much faster.
- Correct solution is to use `std::unique`

Number of Unique Elements

- Sequence must be sorted, however. `std::unique` eliminates equal elements and returns reference to first duplicated element:

```
#include <algorithm>
#include <iostream>

int main() {
    std::sort(a, a + std::size(a));
    std::cout << std::unique(a, a + std::size(a)) - a << std::endl;
}
```

- For example:

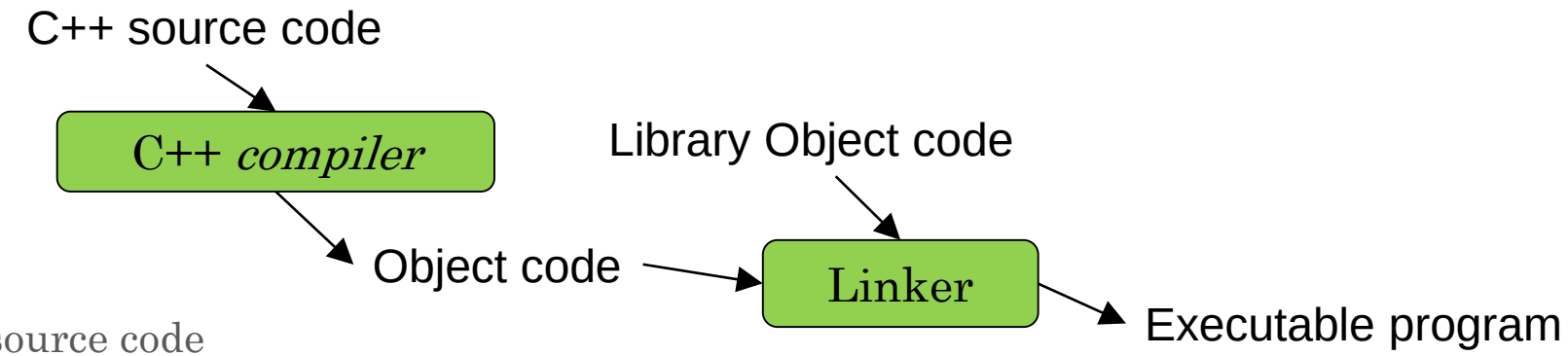
```
1 2 2 2
    ^
```

Use the correct STL data structures

- What container type should you use?
  - Whenever you can, use `std::vector`.
  - If you cannot, find a way so you can.
- Avoid any data structures except arrays. “Well aren’t there exceptions?” No, not for you.
  - Typically advanced data structures are slower than simple data structures.
  - Data structures which appear to be alright when textbook writers wrote their books, are no longer all right now.
  - Computers changed so much. Things no longer work as taught 10 years ago.
- Occasionally we will ask you to use other data structures
  - Mostly to make our point, though

A Word about Compilation

Compilation and Linking



- You write C++ source code
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  - Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  - E.g. input/output libraries, operating system code, and windowing code
- The result is an executable program
  - E.g. a .exe file on windows or an a.out file on Unix

See: Decoding C++ Compilation Process: From Source Code to Binary

CMake

- CMake is a family of tools
  - Building software
  - Testing software
  - Packaging software
- We will use it for building and testing
- CMake generates build systems files (Makefiles and or workspaces)
  - Those can be used to automatically build and test your code
- The user writes a single set of descriptive scripts
  - Define Targets and their inter-dependencies
- CMake is well integrated with many IDEs

Simplest Example

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.0)
project(Demo1)
add_executable(Demo1 demo1.cpp)
```

In source build:

```
% cd demo1
% ls
CMakeLists.txt  demo1.cpp
```

Simplest Example

```
% cmake .
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile fe
-- Detecting C compile fe
-- Check for working CXX
-- Check for working CXX
-- Detecting CXX compiler
-- Detecting CXX compiler
-- Detecting CXX compile
-- Detecting CXX compile
-- Configuring done
-- Generating done
-- Build files have been
```

```
% make
Scanning dependencies of target Demo1
[ 50%] Building CXX object
CMakeFiles/Demo1.dir/demo1.cpp.o
[100%] Linking CXX executable Demo1
[100%] Built target Demo1

% ./Demo1
Hello, world!
```

Homework and Projects

- We will use Visual Studio Code as our development environment
  - Assignment 0 will ask you to install everything and make it work
- We will use Github Classroom for managing your submission
  - Please go get a login on github.com, if you have none yet (this is part of assignment 0)
  - This will teach you the use of the Git source code control system along the lines
  - Special lecture will talk about development environment (git, cmake, etc.)
- Project will be a collaborative assignment
  - Get a sense how collaborative software development works

What is Programming?

So what is Programming?

- Conventional definitions
  - Telling a very fast moron exactly what to do
  - A plan for solving a problem on a computer
  - Specifying the order of a program execution
    - But modern programs often involve millions of lines of code
    - And manipulation of data is central
- Definition from another domain (academia)
  - A … program is an organized and directed accumulation of resources to accomplish specific … objectives …
    - Good, but no mention of actually doing anything
- The definition we'll use
  - Specifying the structure and behavior of a program, and testing that the program performs its task correctly and with acceptable performance
    - Never forget to check that "it" works
  - Analysis, design, programming (coding), testing
- Software == one or more programs

Programming

- Programming is fundamentally simple
  - Just state what the machine is to do
- So why is programming hard?
  - We want "the machine" to do complex things
    - And computers are nitpicking, unforgiving, dumb beasts
  - The world is more complex than we'd like to believe
    - So we don't always know the implications of what we want
  - "Programming is understanding"
    - When you can program a task, you understand it
    - When you program, you spend significant time trying to understand the task you want to automate
  - Programming is part practical, part theory
    - If you are just practical, you produce non-scalable unmaintainable hacks
    - If you are just theoretical, you produce toys

LSU CENTER FOR COMPUTATION & TECHNOLOGY

STE||AR GROUP