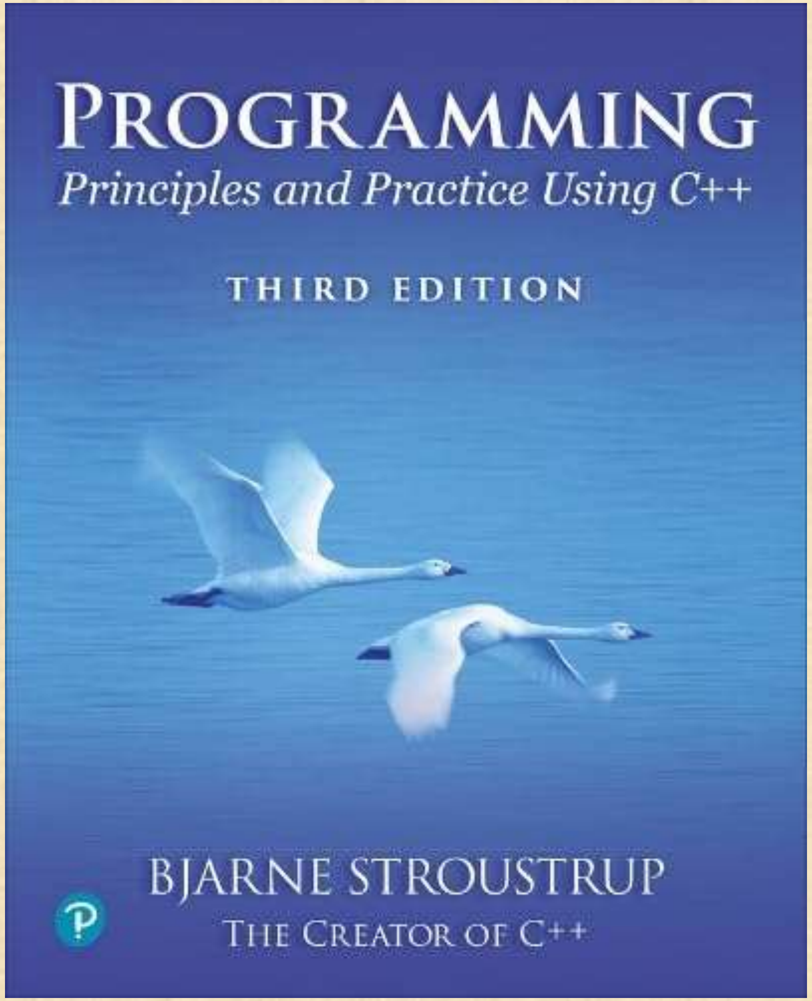
Chapter 1 – Hello, World!



PROGRAMMING

Principles and Practice Using C++

THIRD EDITION

BJARNE STROUSTRUP

THE CREATOR OF C++

Programming is learned
by writing programs.
– Brian Kernighan

Abstract

Today, we'll outline the aims for this course and present a rough course plan. We'll introduce the basic notion of programming and give examples of areas in which software is critical to our civilization. Finally, we'll present the simplest possible C++ program and outline how it can be made into running code.

Overview

- Course aims and outline
- Uses of software
- The first program: "Hello, world!"
- Compilation
- What is programming?

This is a course

- In Programming
- For beginners
  - who want to become professionals
    - i.e., people who can produce systems that others will be happy using
  - who are assumed to be bright
    - Though not (necessarily) geniuses
  - who are willing to work hard
    - Though do need sleep occasionally, and take a normal course load
- Using the C++ programming language

Not!

- A Washout course
  - "If you can get into the science/engineering parts of a university, you can handle this course"
- A course in
  - The C++ programming language
- For students
  - who want to become language lawyers
    - We try not to get bogged down in technical obscurities
  - who are assumed to be a bit dim and fairly lazy
    - We try not to spoon feed
- Using
  - Some untested software development methodologies and a lot of unnecessarily long words

The Aims

- Teach/learn
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities
- After the course, you'll be able to
  - Write small colloquial C++ programs
  - Read much larger programs
  - Learn the basics of many other languages by yourself
  - Proceed with an "advanced" C++ programming course
- After the course, you will ***not*** (yet) be
  - An expert programmer
  - A C++ language expert
  - An expert user of advanced libraries

The Means

- Lectures
  - Attend every one
- Notes/Chapters
  - Read a chapter ahead (about one per lecture)
  - Read the chapter again after each lecture
  - Feedback is welcome (typos, bugs, suggestions, etc.)

The Means (Cont.)

- Work
  - Review questions in chapters
  - Review “Terms” in Chapters
  - Drills
    - Always do the drills
    - Always do the drills before the exercises
  - Exercises
- Course specific
  - Projects
    - That’s where the most fun and the best learning takes place
  - Quizzes
  - Exams

Cooperate on Learning

- Except for the work you hand in as individual contributions, we ***strongly*** encourage you to collaborate and help each other
- If in doubt if a collaboration is legitimate: ask!
  - Don’t claim to have written code that you copied from others
  - Don’t give anyone else your code (that you are to hand in for a grade)
  - When you rely on the work of others, explicitly list all of your sources – i.e., give credit to those who did the work
- Don’t study alone when you don’t have to
  - Form study groups
  - Do help each other (without plagiarizing)
- Go to your TA's office hours
  - Go prepared with questions
  - The only stupid questions are the ones you wanted to ask but didn’t

Rough course outline

- Part I: The basics
  - Types, variables, strings, console I/O, computations, errors, vectors, functions, source files, modules, classes
- Part II: Input and Output
  - Text I/O
  - Graphical output
  - Graphical User Interface
- Part III: Data structures and algorithms
  - Free store, pointers, and arrays
  - Lists, maps, sorting and searching, vectors, templates
  - The STL
- Part IV: Broadening the view (Web only, possibly just self study)
  - Software ideals and history
  - Text processing, numerics, embedded systems programming, testing, C, etc.

Rough course outline (Cont.)

- Throughout
  - Program design and development techniques
  - C++ language features
  - Background and related fields, topics, and languages

Promises

- ***Detail***: We will try to explain every construct used in this course in sufficient detail for real understanding
  - There is no "magic"
- ***Utility***: We will try to explain only useful concepts, constructs, and techniques
  - We will not try to explain every obscure detail
- ***Completeness***: The concepts, constructs, and techniques can be used in combination to construct useful programs
  - There are, of course, many useful concepts, constructs, and techniques beyond what is taught here

More Promises

- ***Realism***: The concepts, constructs, and techniques can be used to build "industrial strength" programs
  - i.e., they have been used to …
- ***Simplicity***: The examples used are among the simplest realistic ones that illustrate the concepts, constructs, and techniques
  - Your exercises and projects will provide more complex examples
- ***Scalability***: The concepts, constructs, and techniques can be used to construct large, reliable, and efficient programs
  - i.e., they have been used to …

Feedback request

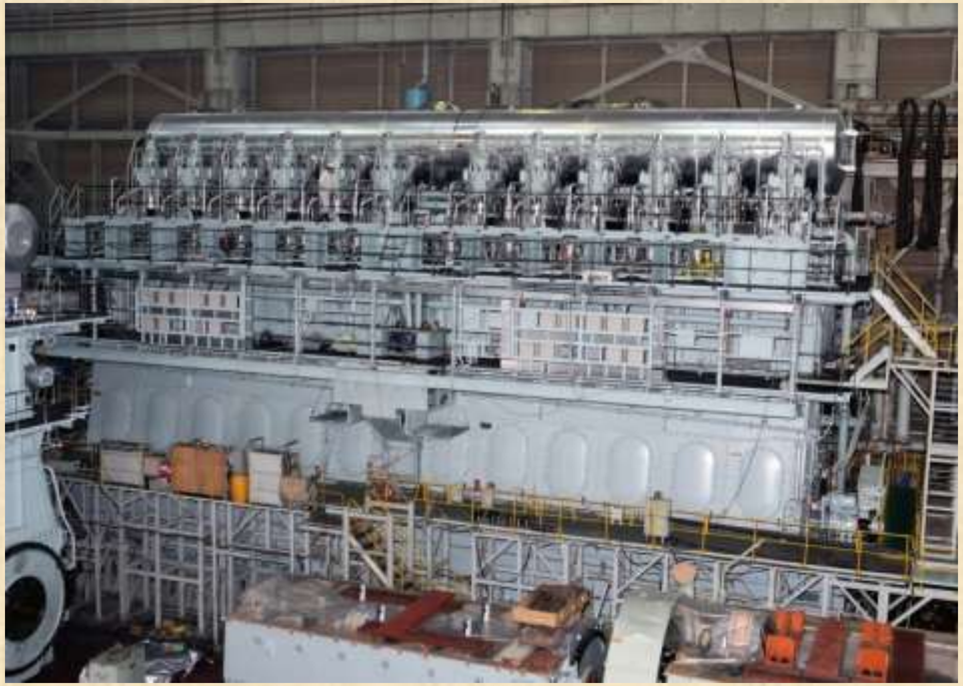
- Please mail questions and constructive comments to
  - ???
- Your feedback will be most appreciated
  - Style, contents, detail, examples, clarity, conceptual problems, exercises, missing information, depth of presentation, etc.
- Book support website
  - www.stroustrup.com/Programming
- Local course support website
  - ???

Why programming?

- Our civilization runs on software
  - Most engineering activities involve softwar

- Note: most programs do not run on things that look like a PC
  - a screen, a keyboard, a box under the table

Ships



- Design
- Construction
- Management
- Loading
- Scheduling
- Route planning

- Monitoring
- Engine
- Hull design
- Pumps

Aerospace

- Communication
- Control
- Display
- Routing

- Signal processing
- “Gadget” control
- telemetry

Cars

(distributed computers with wheels)

- Gas, diesel, hybrid, electric
- Self-driving
- Navigation
- Entertainment

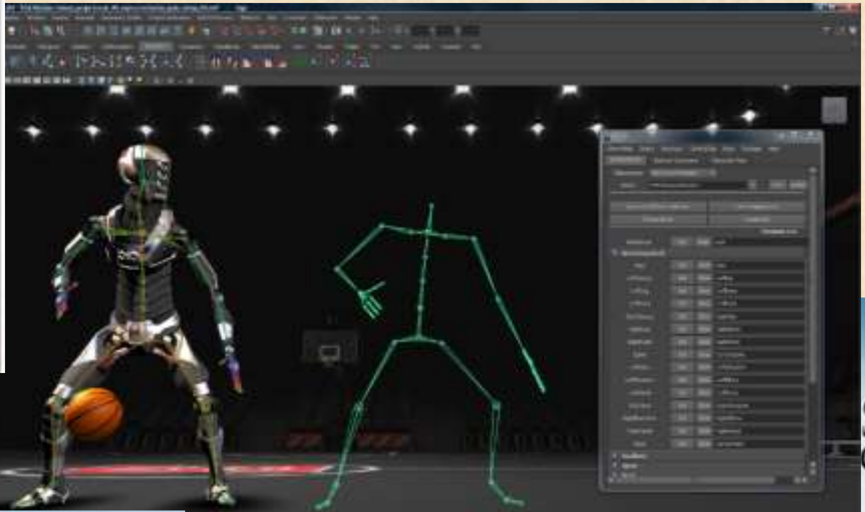
- Design
- Monitorin
- Steering
- Breaks

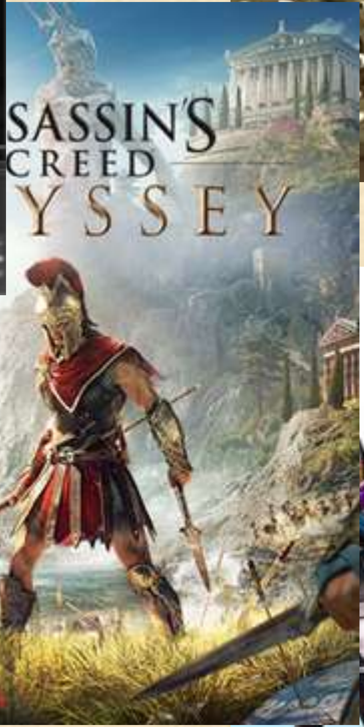
Phones

- Voice quality
- User interfaces
- Billing
- Mobility

- Switching
- Reliability
- Provisioning
- Images
- Videos
- apps

hics, animation, and games





Commerce, finance, and governance

- Banks
- Stock exchanges
- Currency exchange
- Social services
- Medical records
- Taxes
- Online stores

Medicine

- Scanners
- Vaccine development and production
- Analysis (blood, tissue)
- General research

- Genomics
- Materials design
- Simulations
- ???

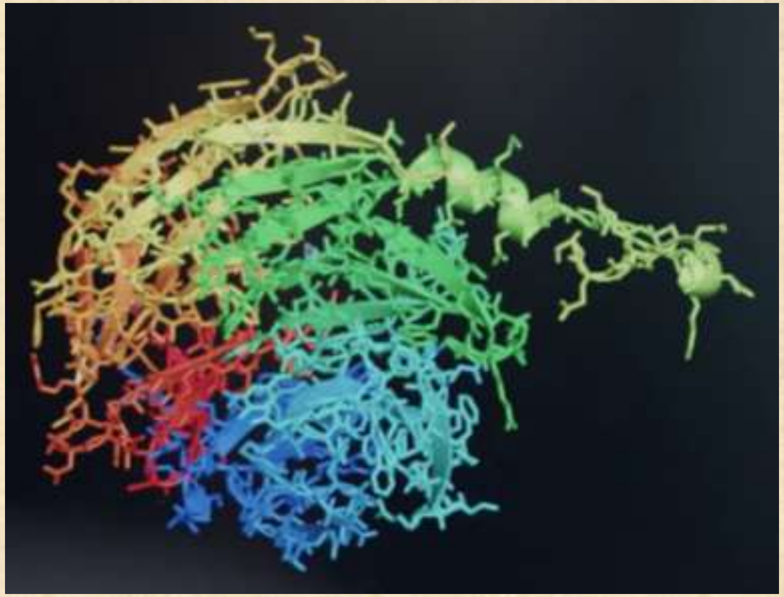
Energy

- Control
- Monitoring
- Analysis
- Design

- Communications
- Visualization
- Manufacturing
- Transport

Science

- Physics
- Biology
- Engineering
- Astronomy



Foundations

- Microelectronics
- Computers
- Routers
- Networking
- Manufacturing
- Wireless

- Operating systems
- Browsers
- Virtual machines
- Search engines

Laptops, tablets, workstations, servers, …

- There’s a lot more to computing than games, word processing, browsing, and spreadsheets!

Why C++ ?

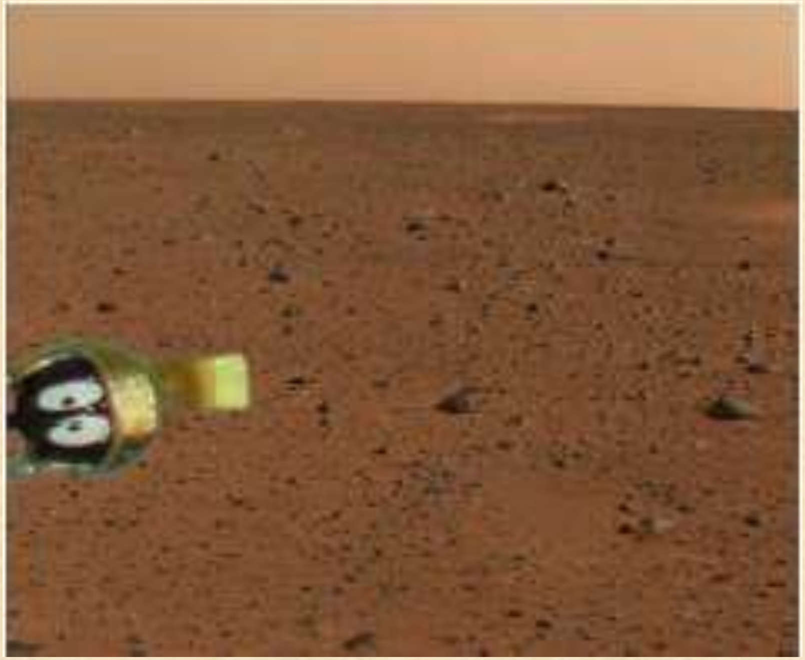
- You can’t learn to program without a programming language
- The purpose of a programming language is to allow you to express your ideas in code
- C++ is the language that most directly allows you to express ideas from the largest number of application areas
- C++ is the most widely used language in engineer areas
  - http://www.stroustrup.com/applications.html
- Other languages often use C++ for computation intensive tasks
  - E.g., Python doing AI
- The implementation of many languages are C++ programs
  - E.g., Java and Javascript

Why C++ ?

- C++ is precisely and comprehensively defined by an ISO standard
  - And that standard is almost universally accepted
  - The most recent standard is ISO C++ 2023
- 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

Where is C++ Used?

- Just about everywhere



- C++ plays a major part in all the examples and photos used here
  - See www.stroustrup.com/applications.html
  - Note: a large system is not written exclusively in one language

A first program – complete

```
// a first program:

import std;                     // get the standard library facilities

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

  // note the semicolons; they terminate statements
  // braces { … } group statements into a block
  // main( ) is a function that takes no arguments ( )  and returns an
  integer result
```

A first program – older style

```
// a first program:


#include <iostream>;               // get the library facilities
 needed for now


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

 // the std:: says that cout comes from the standard library
```

A first program – use while learning

```
// a first program:

#include “PPP.h”              // get PPP support

int main()                    // main() is where a C++ program starts
{
  cout << "Hello, world!\n";      // 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 compiler problems.
// \n is a notation for a new line

// for PPP.h, see www.stroustrup.com/Programming
```

A first program – older style

```
// a first program:

#include “PPPheaders.h"   // get the PPP support on older C++ implementations

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

  // note the semicolons; they terminate statements
  // braces { … } group statements into a block
  // main( ) is a function that takes no arguments ( )
  //      and returns an int (an integer value) to indicate success or failure
```

Hello, world!

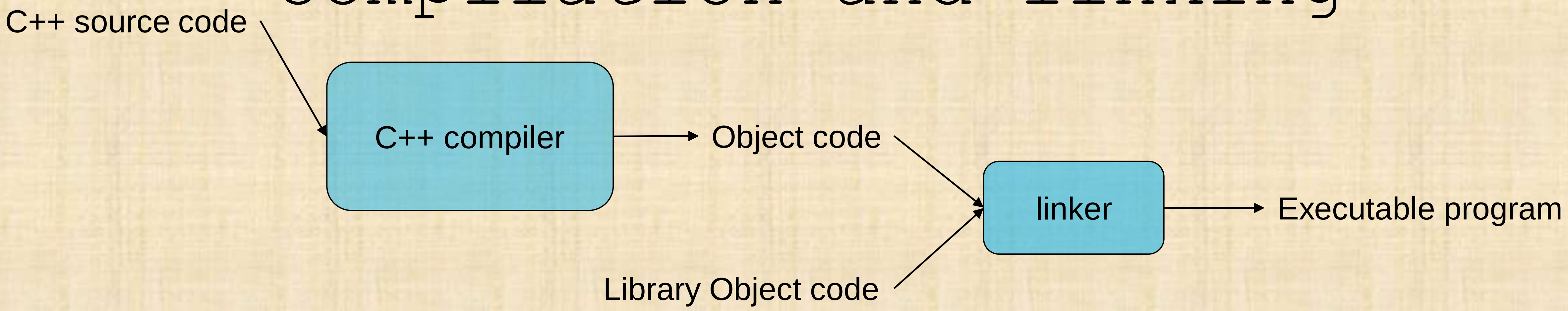
- "Hello world" is a very important program
  - Its purpose is to help you get used to your tools
    - Compiler
    - Program development environment
    - Program execution environment
  - Type in the program **carefully**
    - After you get it to work, please make a few mistakes to see how the tools 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 **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!

Compilation and linking



- You write C++ source code (e.g. **Hello.cpp**)
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (e.g., **Hello.o**)
  - sometimes called machine code
  - Object code is simple enough for a computer to "understand"
- The linker links your code to other code needed for it 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 Linux

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
- 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

Support

- Rely on local support, if available
- A C++ implementation (good and free)
  - Clang, GCC, Microsoft, or other
  - Always use the most recent version
    - don’t suffer problems that “they” have already fixed
    - Have the contemporary language features and library components available
- A Software development environment
  - Visual studio, visual studio code, X-code, or other
  - Or work from the command line
- Online compiler: e.g., https://godbolt.org/
- Online reference manual: https://en.cppreference.com/w/
- Supporting libraries: www.stroustrup.com/programming.html

The next lecture

- Will talk about types, values, variables, declarations, simple input and output, very simple computations, and type safety.