Advanced Programming

ACSE-5: Lecture 1

Adriana Paluszny
Royal Society University Research Fellow

Why programming?

- “Our civilization runs on software” Bjarne Stroustrup
- Code runs on a variety of hardware (not only PCs)

The Aims

- Learning objectives
  - 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
- After the course, you will ***not*** (yet) be
  - A C++ language expert
  - An expert user of advanced libraries
  - But you will be on your way!

The Means

- Lectures
  - Attend every one
  - Few slides, and lots of live coding
  - Teaching 50 min, Break 10 min (3x)
  - Try to follow what is being done, you will get the final code after class, ask and interrupt if you want more/other details or don't follow or understand!
- Notes/Chapters
  - Extra mile: Read a chapter per week
    - Bjarne Stroustrup: Programming -- Principles and Practice Using C++
  - Feedback is welcome (typos, suggestions, etc.)
- Assignments
  - "That's where the most fun and the best learning takes place"
  - 3 Assignments (individual or in pairs) 23.3% each
  - Exam/Coursework in-class (individual): Last class 30%

Cooperate on Learning

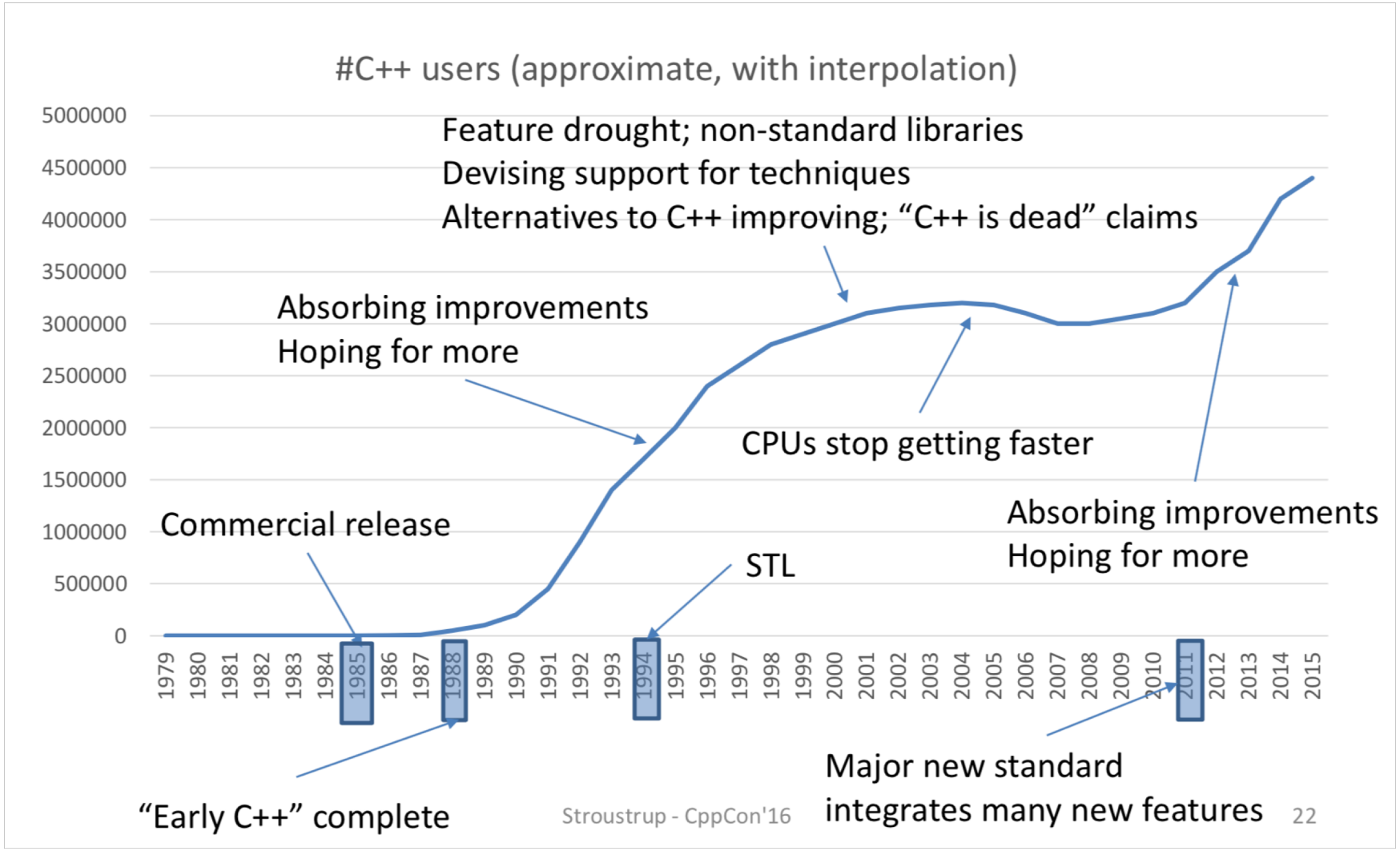
Except for the work you hand in as individual contributions, we ***strongly*** encourage you to collaborate and help each other

But don’t copy code from the internet, try to write it yourself

Why C++ ?

- 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
- Abstraction + Hardware control
- C++ is the most widely used language in engineering areas
  - Finance, Ships, Aviation, Aircrafts, Phones, Energy (Oil, Gas, ..), Visualisation, Manufacturing, Comms, Games, Mars Rover
  - Adobe, Google file system, Bloomberg, Microsoft OS, MS Office, MS Explorer, Visual Studio, Mozilla Firefow, mySQL

“C++ Success” by Bjarne Stroustrup



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++ 2017
- 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
- C++ is changing and evolving all the time, and it is very fast!

Rough course outline (1)

- 14/01 – Introduction: C++, gcc/Intel compilers, MSVC IDE, compiling and linking, executables. Data types. (Adriana Paluszny)
- 17/01 – Functional programming. Functions: passing by value and reference. Recursion vs. iteration. Input/output. Pointers and References. Introduction to containers. (Steven Dargaville)
- 21/01 - Standard template library (STL). Introduction to objects (using objects). STL Containers: vectors, lists, maps. MSVC Debugger. Plotting with C++ (Gnuplot). (Adriana Paluszny)
- 24/01 – Object oriented programming (creating objects). Classes, constructor, destructor, copy constructor, members, Boolean operators, mutators, accessors. Introduction to Inheritance, Polymorphism & Encapsulation in C++. Making objects STL compatible. (Adriana Paluszny)
- 28/01 – 30-minute C++ History Trip. C++18. The Standards Committee. Programming paradigms. Agile. Introduction to UML. Roles in programming teams (Architect vs. programmer). (Adriana Paluszny)

Rough course outline (2)

- 31/01 – Memory management with C++. Safety and housekeeping. Applied to linear systems and matrices. BLAS/LAPACK. Second Assignment. (Steven Dargaville)
- 4/2 – Memory management and optimisation. Scaling. Overwriting. Introduction to templates. (Steven Dargaville)
- 7/2 – Polymorphism in C++. Sparse CSR Formats. Introduction to PETSc. (Steven Dargaville)
- 11/2 – Templates. Reference Counting. Dense and iterative methods for matrix inversion. (Steven Dargaville)
- 14/2 – Wavelets (Steven Dargaville). An introduction to image processing and filters. (Adriana Paluszny)
- 18/2 – An introduction to the architecture of a commercial Medical Imaging cloud-based software. (Adriana Paluszny + External Speaker(s))
- 21/2 – Computer-based programming class test.

Rough course outline (Cont.)

- Throughout
  - Program design and development techniques
  - C++ language features
  - Background and related fields, topics, and languages
  - Two instructors (Adriana and Steven)
  - Two GTAs (Robin and Cristina)

Feedback request

- Please mail questions and constructive comments to

  apaluszn@imperial.ac.uk

Or send anonymous feedback through the SuggestionOx

https://www.suggestionox.com/r/UcN8ib

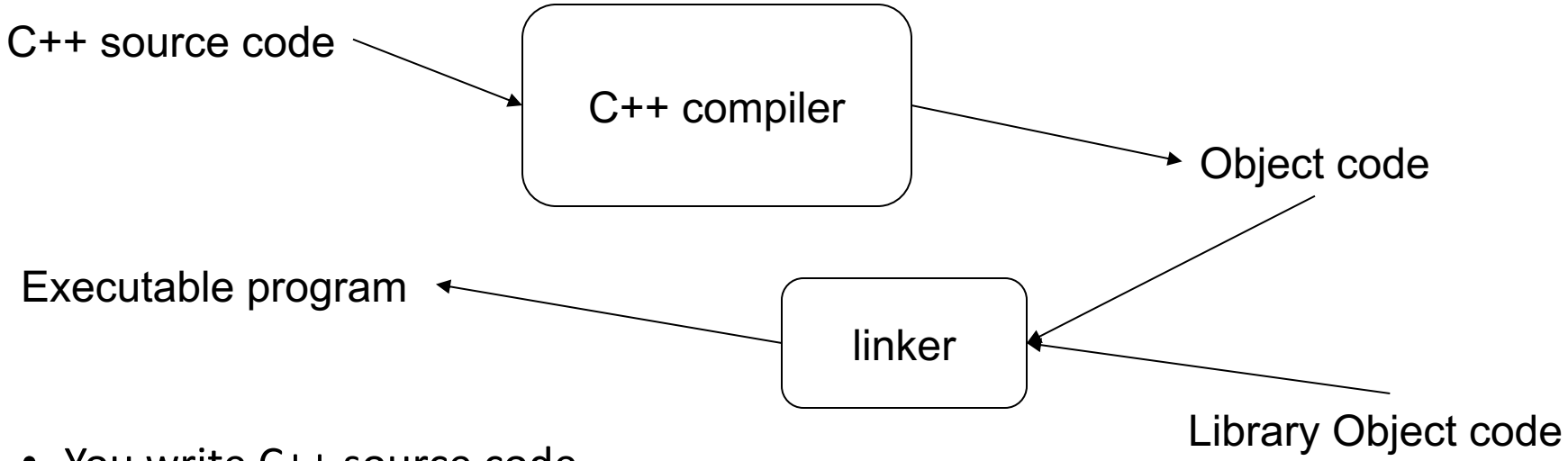
- Your feedback will be most appreciated
  - On style, contents, detail, examples, clarity, conceptual problems, exercises, missing information, depth, etc.

- Course support website https://github.com/msc-acse/ACSE-5

A first program – complete

```
// 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
}

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

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

Source files

header.h:

Who am I?

Interfaces to libraries
(declarations)

source.cpp:

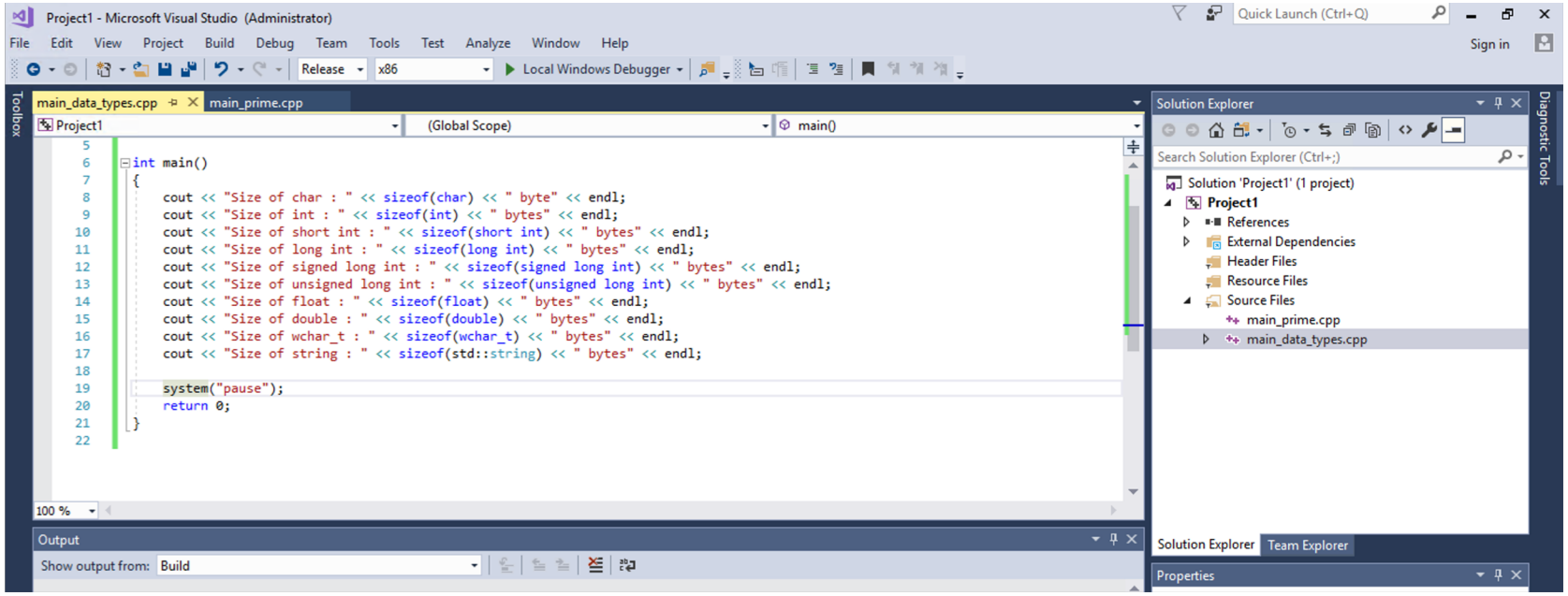
What am I?

```
#include <string>
#include “MyFile.h”

My code
My data
(definitions)
```

An IDE = MSVC Community 2017

Integrated development environment

Beyond a file editor it provides support for the construction of the makefiles = Editor + Compiler + Linker + Debugger + Profiler



Objects, types, and values aka. “Data types”

After Chapter 3 of Bjarne Stroustrup

www.stroustrup.com/Programming

Overview

- Strings and string I/O
- Integers and integer I/O
- Types and objects
- Type safety

Go to MSVC

Live coding …

[the following slides will be for reference]

Integers and Strings

- Strings
  - **cin >>** reads a word
  - **cout <<** writes
  - **+** concatenates
  - **+= s** adds the string **s** at end
  - **++** is an error
  - **-** is an error
  - …

- Integers and floating-point numbers
  - **cin >>** reads a number
  - **cout <<** writes
  - **+** adds
  - **+= n** increments by the int **n**
  - **++** increments by **1**
  - **-** subtracts
  - …

The type of a variable determines which operations are valid and what their meanings are for that type

(that's called “overloading” or “operator overloading”)

Names

- A name in a C++ program
  - Starts with a letter, contains letters, digits, and underscores (only)
    - **x, number_of_elements, Fourier_transform, z2**
    - Not names:
      - **12x**
      - **timetomarket**
      - **main line**
    - Do not start names with underscores: **_foo**
      - those are reserved for implementation and systems entities
  - Users can't define names that are taken as keywords
    - E.g.:
      - **int**
      - **if**
      - **while**
      - **double**

Names

- Choose meaningful names
  - Abbreviations and acronyms can confuse people
    - **mtbf, TLA, myw, nbv**
  - Short names can be meaningful
    - (only) when used conventionally:
      - **x** is a local variable
      - **i** is a loop index
  - Don't use overly long names
    - Ok:
      - **partial_sum**
        element_count
        staple_partition
    - Too long:
      - **the_number_of_elements**
        remaining_free_slots_in_the_symbol_table

Types and literals

- Built-in types
  - Boolean type
    - **bool**
  - Character types
    - **char**
  - Integer types
    - **int**
      - **and short** and **long**
  - Floating-point types
    - **double**
      - and **float**
- Standard-library types
  - **string**
  - **complex<Scalar>**

- Boolean literals
  - **true false**
- Character literals
  - **'a', 'x', '4', '\n', '$'**
- Integer literals
  - **0, 1, 123, -6, 034, 0xa3**
- Floating point literals
  - **1.2, 13.345, .3, -0.54, 1.2e3, .3F**
- String literals **"asdf"**,
  "Howdy, all y'all!"
- Complex literals
  - **complex<double>(12.3,99)**
  - **complex<float>(1.3F)**

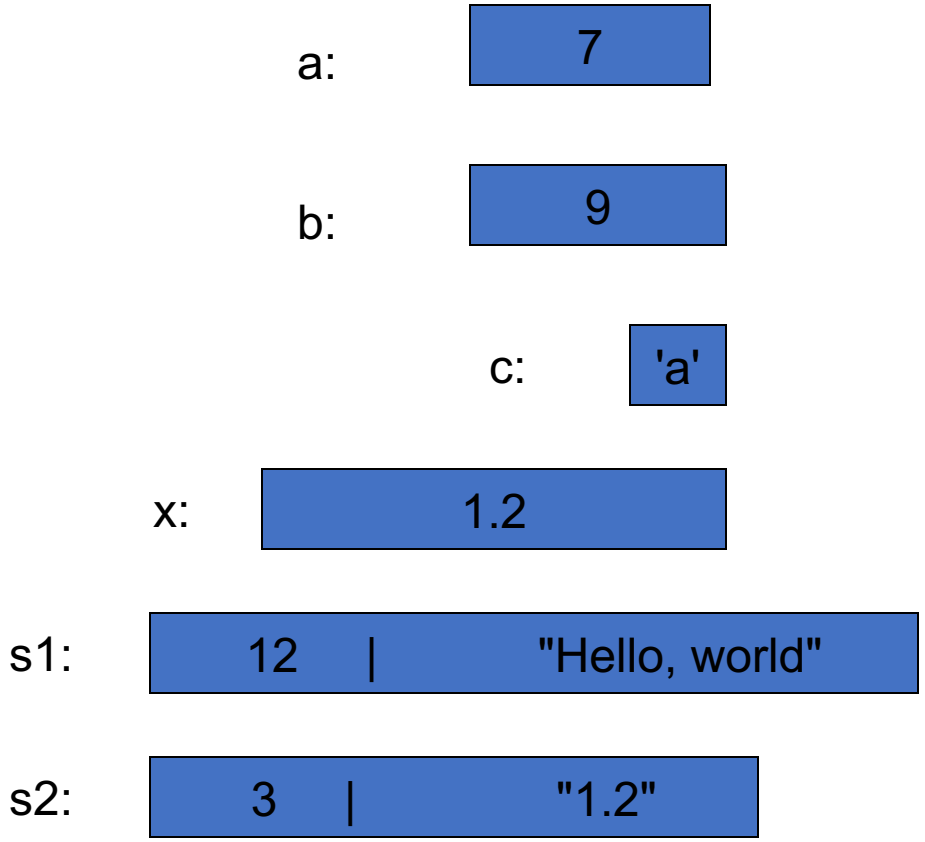
If (and only if) you need more details, see the book!

Types

- C++ provides a set of types
  - E.g. **bool**, **char, int, double**
  - **Called "built-in types"**
- C++ programmers can define new types
  - Called "user-defined types"
  - We'll get to that eventually
- The C++ standard library provides a set of types
  - E.g. **string**, **vector**, **complex**
  - Technically, these are user-defined types
    - they are built using only facilities available to every user

Declaration and initialization

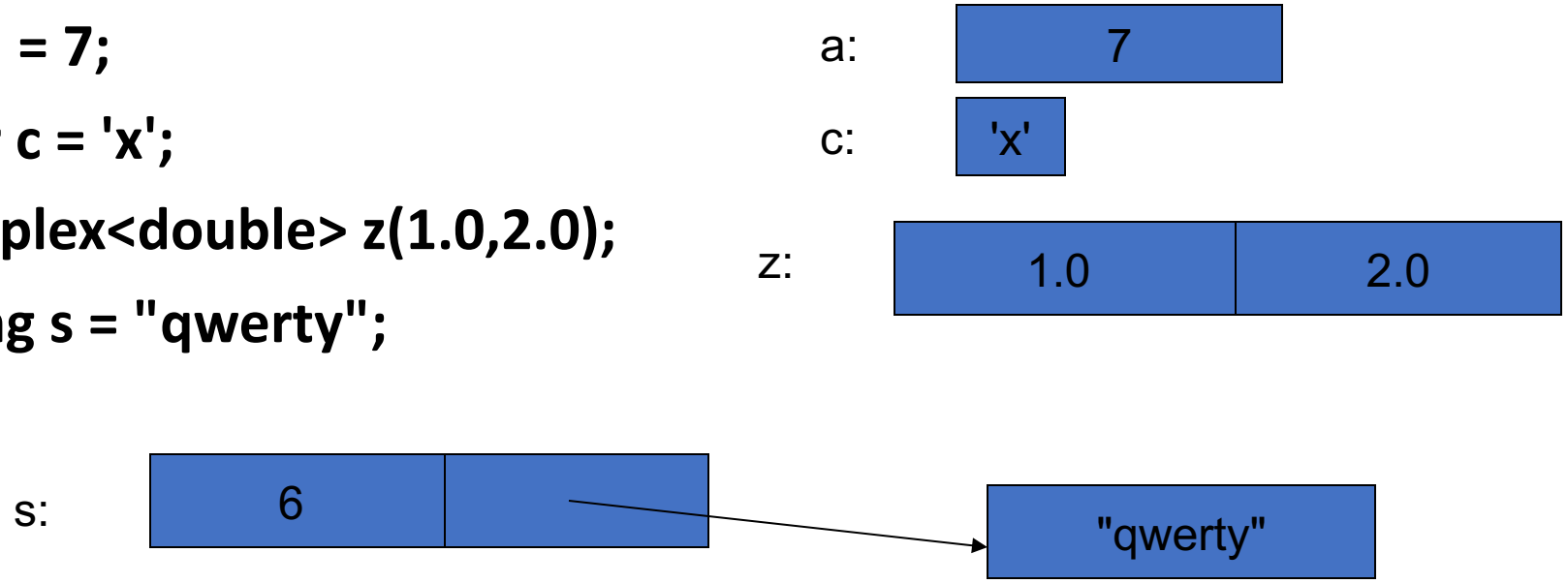
```
int a = 7;

int b = 9;

char c = 'a';

double x = 1.2;

string s1 = "Hello, world";

string s2 = "1.2";
```



Objects

- An object is some memory that can hold a value of a given type
- A variable is a named object
- A declaration names an object

```
int a = 7;
char c = 'x';
complex<double> z(1.0,2.0);
string s = "qwerty";
```



Type safety

- Language rule: type safety
  - Every object will be used only according to its type
    - A variable will be used only after it has been initialized
    - Only operations defined for the variable's declared type will be applied
    - Every operation defined for a variable leaves the variable with a valid value
- Ideal: static type safety
  - A program that violates type safety will not compile
    - The compiler reports every violation (in an ideal system)
- Ideal: dynamic type safety
  - If you write a program that violates type safety it will be detected at run time
    - Some code (typically "the run-time system") detects every violation not found by the compiler (in an ideal system)

Quick guide to safety

Safe conversions

bool to **char**
bool to **int**
bool to **double**
char to **int**
char to **double**
int to **double**

Unsafe conversions

double to **int**
double to **char**
double to **bool**
int to **char**
int to **bool**
char to **bool**

Go to MSVC

Live coding from now on…

Homework

Read Chapters 1, 2, and 3 of Programming: Principles and Practice using C++ by Bjarne Stroustrup

Practice by solving the eleven short exercises at the end of Chapter 3.

This will start preparing you for the assignment.

Assignment 1 will be published on GitHub by Thursday.