CMSC202
Computer Science II

Lecture 01 –
Introduction and C++ Primer

CMSC 202 Faculty

Based on slides by Chris Marron and Katie Gibson

Course Overview

Spring 2025

- CMSC 202 will use the following tools:
  - Blackboard
  - YouTube
  - Discord (office hours and tutoring)
  - ITE 240 – computer lab
- Discord will be synchronous
  - We will be able to talk one on one and share our screens for debugging
  - Please download the app on your laptop/desktop (not the web interface)
  - We will email you an invitation in the next week or so

Course Information

- Second course in the CMSC programming sequence
  - Preceded by 201 (Computer Science I)
  - Followed by 341 (Data Structures)
- CMSC must pass with a B or better
- CMPE must pass with a C or better
- Only two attempts are allowed in CMSC courses (regardless of major)

Quick Note About Grades

- Students are **not** allowed to retake a class if they have taken its successor
- If you are a CMSC major and received a “C” in 201, you **must** retake 201 before this class (Except in Spring 2020)
  - If you receive a grade in this class, you can no longer be a computer science major at UMBC!
- Students are only allowed two attempts in CMSC 201 or CMSC 202
  - A “W” counts as an attempt!

Labs / Discussion

- Labs
  - 100 total points
  - Best 10 of 13 labs
- Labs 2 – 13 Prelab quizzes
  - 4 points
  - One attempt at the quiz
  - Comes out on Friday at 9am
  - Due on Monday by 10am (for everyone – regardless of when your lab is!)
- Labs
  - 6 points
  - TA will be available during schedule time on Discord for help

Lab 1

- We do NOT have in-person lab this week or next week
  - January 27th – January 30th
  - February 3rd – February 6th
- Lab 1 is the only online lab
  - It will be released this weekend
  - Due Sunday, February 9th at 11:59pm on GL
- Lab 2
  - Prelab quiz
    - Open on Friday, February 7th and close Monday, February 10th at 10am
  - Meet in-person during your regularly schedule lab
    - Monday, February 10th – Thursday, February 13th

Review of the Syllabus

- Grading Criteria
- Course Policies
- Attendance
- Communication
- Academic Integrity
- Blackboard

Development Environment

- You will use the GL Linux systems and GCC (GNU Compiler Collection) suite for development.
- You will learn to be semi-literate in Linux and shell usage.
- You will learn to use a text editor — Emacs is recommended.
- You may use IDEs such as Eclipse or XCode, but support will not be provided, and…

Your programs must compile and function correctly on the GL Linux systems.

What the Course is About

- An introduction to:
  - Object-oriented programming (OOP) and object-oriented design (OOD)
  - Basic software engineering techniques
- Emphasis on proper program design
- Tools
  - C++ programming language, GCC (Gnu Compiler)
  - Linux (GL system)

Challenges

- Getting used to the Linux environment (tends to hit transfer students hardest).
- Starting the projects early.
- CMSC 202 is much more difficult than CMSC 201 – you will need to be more self-sufficient.
- Waiting too late to seek help.
- Thinking all that matters is the projects.
  - Practice programming outside of the projects!

Introduction to C++

Today's Objectives

- To discuss the differences between the Python and C++ programming languages
  - Interpreted vs compiled
  - More restrictions on programming "style"

- To begin covering the basics of C++
  - Classes
  - Object-Oriented Programming

Why C++?

How Old Are Programming Languages?

Plankalkül 1945
Short Code 1949
FORTRAN 1957
ALGOL 1958
LISP 1958
COBOL 1959
BASIC 1964
PL/I 1965
SNOBOL4 1967
SIMULA 67 1967
Pascal 1971
C 1972
Prolog 1972
Smalltalk 1972
ML 1977
Icon 1979
Ada 1980
C++ 1983
Objective-C 1983
Erlang 1986
Perl 1987
Haskell 1990
Python 1991
Ruby 2/24/93
Java 1995
JavaScript 1995
PHP 3 1998
C# 2000
D 2001
Scala 2003
Clojure 2007
Go 2008
Rust 2010
Kotlin 2011
TypeScript 2012
Julia 2012
Swift 2014
Goaldi 2015

Why C++ for CMSC 202?

- Popular modern OO language
- Wide industry usage
- Used in many types of applications
- Desirable features
  - Object-oriented
  - Portable (not as much as Java, but fairly so)
  - Efficient
  - Retains much of its C origins

TIOBE for January 2025

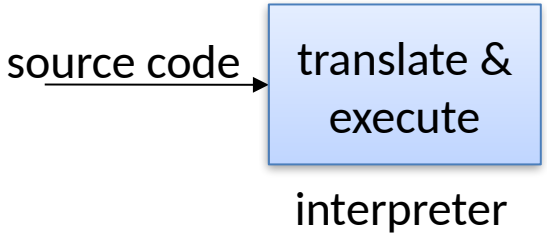
Jan 2025	Jan 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.28%	+9.32%
2	3	↑	C++	10.29%	+0.33%
3	4	↑	Java	10.15%	+2.28%
4	2	↓	C	8.86%	-2.59%
5	5		C#	4.45%	-2.71%
6	6		JavaScript	4.20%	+1.43%
7	11	⇈	Go	2.61%	+1.24%
8	9	↑	SQL	2.41%	+0.95%
9	8	↓	Visual Basic	2.37%	+0.77%
10	12	↑	Fortran	2.04%	+0.94%
11	13	↑	Delphi/Object Pascal	1.79%	+0.70%
12	10	↓	Scratch	1.55%	+0.11%
13	7	⇊	PHP	1.38%	-0.41%
14	19	⇈	Rust	1.16%	+0.37%
15	14	↓	MATLAB	1.07%	+0.09%

From https://www.tiobe.com/tiobe-index/

Compiled Languages

Interpreters, Compilers, & Hybrids

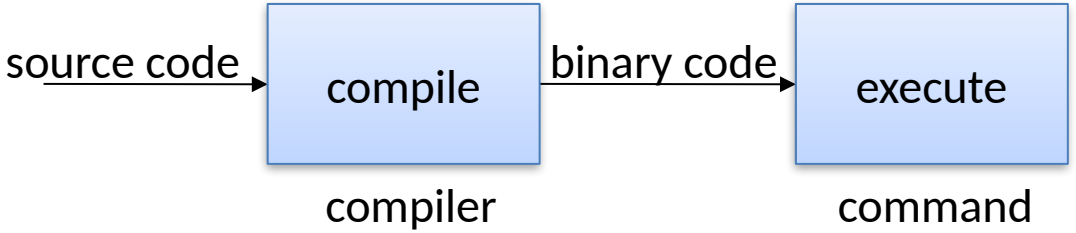
Interpreted Languages (e.g. JavaScript, Perl, Ruby)



Interpreter translates source into binary and executes it

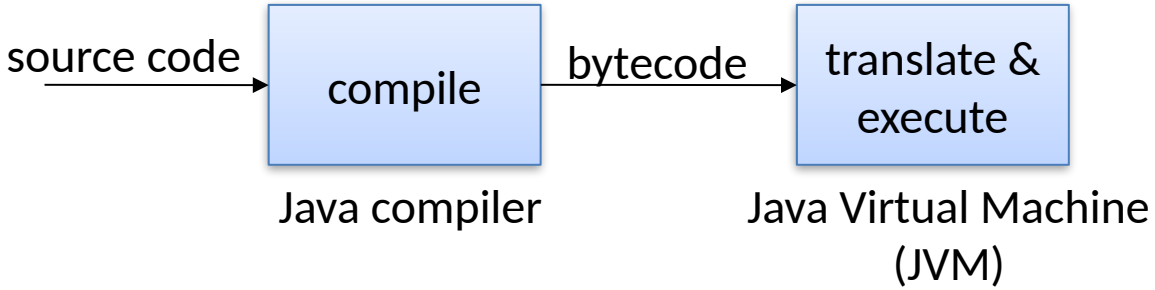
Small, easy to write
Interpreter is unique to each ***platform (operating system)***

Compiled Languages (e.g. C, C++)



Compiler is platform dependent

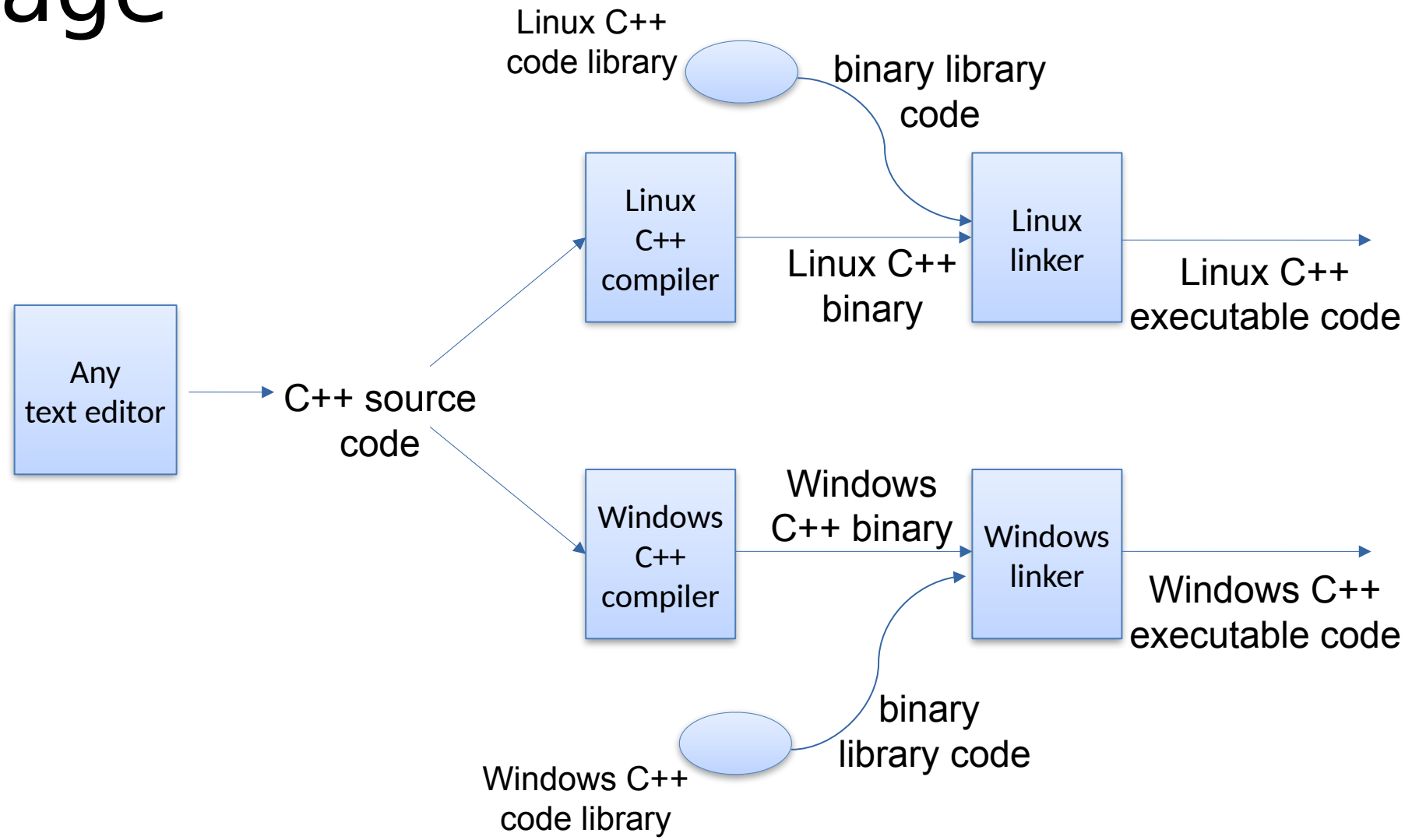
Many other models: e.g., Java (Python is stranger still):



Bytecode is platform independent

JVM is an interpreter that is platform dependent

C++ Compilation and Linkage



Language Versions

Python vs C++ Syntax

Python

```
print "Hello, world"
quotient = 3 / 4
if (quotient == 0):
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

C++

```
#include <iostream>
 using namespace std;

 int main() {
   int quotient;
   cout << "Hello, world";
   quotient = 3 / 4;
   if (quotient == 0) {
     cout << "3/4 == 0";
     cout << " in C++";
   } else {
     cout << "3/4 != 0";
   }
   return 0;
 }
```

What do we know about this chunk of Python code?

What is the output of the Python code?

What is the output of the C++ code?

Python vs C++ Syntax

Python

```
print "Hello, world"
quotient = 3 / 4
if (quotient == 0):
    print "3/4 == 0",
    print "in Python"
else:
    print "3/4 != 0"
```

C++

```
#include <iostream>
 using namespace std;

 int main() {
   int quotient;
   cout << "Hello, world";
   quotient = 3 / 4;
   if (quotient == 0) {
     cout << "3/4 == 0";
     cout << " in C++";
   } else {
     cout << "3/4 != 0";
   }
   return 0;
 }
```

What did you notice are the differences?

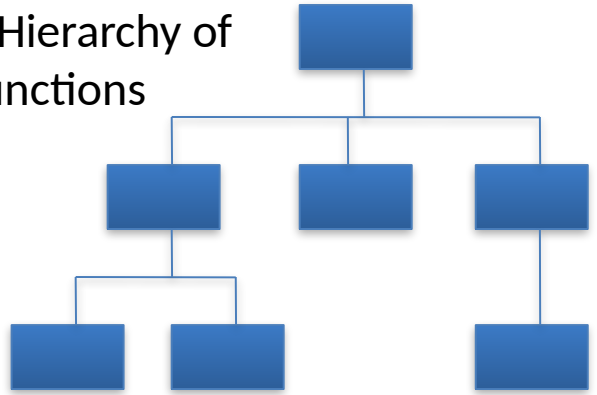
1. Must have a "`main()`" function
2. Statements end with "`;`"
3. Variables must be declared
4. "`if/else`" syntax different
5. Statement blocks demarcated by "`{ . . . }`"
6. Versions matter! (Python 2.x vs Python 3.x)

Procedural vs OOP Languages

Procedural vs OOP

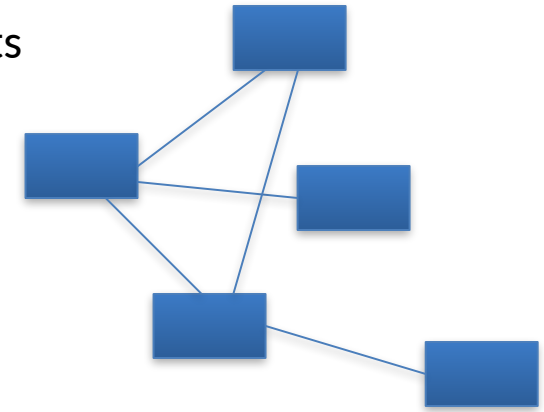
- Procedural
  - Modular units: functions
  - Program structure: hierarchical
  - Data and operations are not bound to each other
  - Examples:
    - C, Pascal, Basic, Python

A Hierarchy of Functions



- Object-Oriented (OO)
  - Modular units: objects
  - Program structure: a graph
  - Data and operations are bound to each other
  - Examples:
    - C++, Java, Python (huh?!)

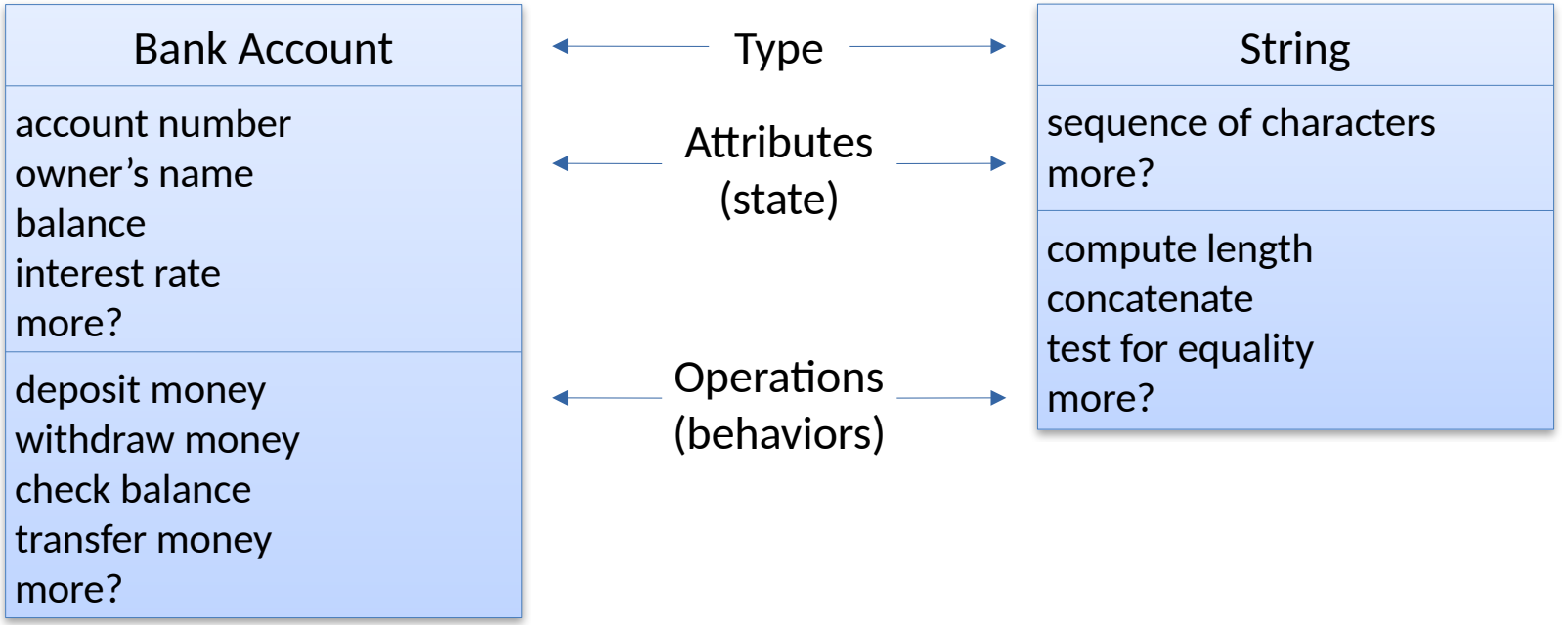
A Collection of Objects



Classes and Objects

Classes

- First off, what is a class?
  - A data type containing:
    - Attributes – make up the object's state
    - Operations – define the object's behaviors



Objects

- An ***object*** is a particular instance of a class

Smith's Account	Doe's Account	Jones's Account
12-345-6 Jake Smith $1,250.86 1.5%	65-432-1 John Doe $5.50 2.7%	43-261-5 Jane Jones $825.50 2.5%

- For any of these accounts, one can…
  - Deposit money
  - Withdraw money
  - Check the balance
  - Transfer money

Identifiers and Variables

C++ Identifiers and Variables

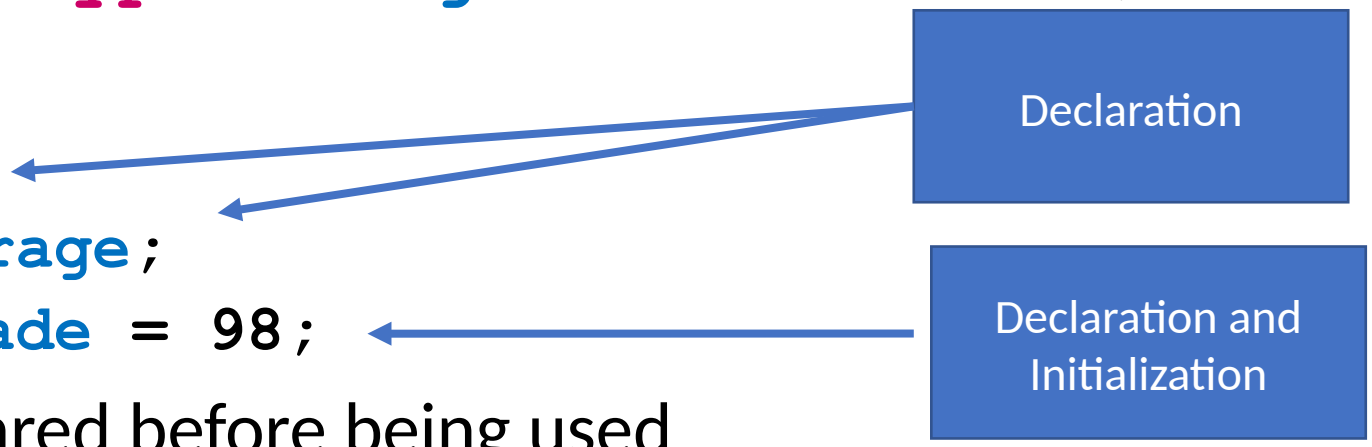
- C++ Identifiers
  - Can’t use keywords/reserved words
  - Case-sensitivity and validity of identifiers
  - Meaningful names!
  - Used for variables, functions, class names, and more

- Variables
  - A memory location to store data for a program
  - **Generally, in C++ we declare all data before use in program**

Variable Declaration

- Syntax: `<type> <legal identifier>;`
- Examples:

```
int sum;
float average;
double grade = 98;
```



- Must be declared before being used
- Must be declared to be of a given type (e.g. int, float, char, etc.)

Don't forget the semicolon at the end!

Declaring a Variable

- When we declare a variable, we tell the compiler:
  - When and where to set aside memory space for the variable
  - How much memory to set aside
  - How to interpret the contents of that memory; AKA, the specified data type
  - What name we will be referring to that location by: its identifier, or name

Initializing Variables

- When you declare a variable, a certain number of bytes in memory get allocated to represent that variable.

```
int myValue; //Allocates 4 bytes of memory for the int
```

- The 4 byes of memory allocated above are NOT automatically set to 0 so they can have garbage data in memory. As such, we try to initialize our variables.

```
int myValue = 0; //Allocates 4 bytes and sets to 0
```

Assignment Operator (=)

- You can initialize data in declaration statement
  - Results may be "undefined" if you don't initialize!

```
int myValue = 0;
```

- Assigning data during execution
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
  - Example: **`distance = rate * time;`**

  Lvalue: "**`distance`**"

  Rvalue: "**`rate * time`**"

Naming Conventions

- Naming conventions are rules for names of variables to improve readability
  - CMSC 202 has its own standards, described in detail on Blackboard
    - Start with a lowercase letter
    - Indicate "word" boundaries with an uppercase letter
    - Restrict the remaining characters to digits and lowercase letters

  `topSpeed` `bankRate1` `timeOfArrival`

- Note: variable names are still case sensitive!

Simple Data Types

Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	−32,768 to 32,767	Not applicable
`int`	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

Simple Data Types

Important Data Types

Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	−32,768 to 32,767	Not applicable
`int`	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

More Simple Data Types

long double	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
char	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
bool	1 byte	true, false	Not applicable

```
int main () {
  bool mytest = true;
  if(mytest)
    cout << "mytest is true" << endl;
}
```

Same as:

```
if(mytest == true)
```

Data Types

- One of the big changes from Python to C++
- Variables can only be of one type
  - A string cannot be changed into a list
  - A tuple cannot be changed into a dictionary
  - An integer is always an integer – forever
- A variable's type must be explicitly declared

Data Assignment Rules

- Compatibility of Data
  - Some data types are compatible
    - All numbers are compatible but you may lose precision
    - `intVar = 2.99;` ☾ 2 is assigned to **`intVar`** (ints do not store decimals)
  - Other data types are not compatible and cause type mismatches
    - Cannot place value of one type into variable of another type

Literal Data

- Literals
  - Examples:

```
2           // Literal constant int
5.75        // Literal constant double
'Z'         // Literal constant char
"Hello World\n"  // Literal constant string
```

- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!

Live Coding

Lec1 –> variables.cpp

C++ Primer

A Sample C++ Program

```
1   #include <iostream>
2   using namespace std;

3   int main( )
4   {
5       int numberOfLanguages;

6       cout << "Hello reader.\n"
7            << "Welcome to C++.\n";

8       cout << "How many programming languages have you used? ";
9       cin >> numberOfLanguages;

10      if (numberOfLanguages < 1)
11          cout << "Read the preface. You may prefer\n"
12               << "a more elementary book by the same author.\n";
13      else
14          cout << "Enjoy the book.\n";

15      return 0;
16  }
```

Sample Program Usage

SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 0
Read the preface. You may prefer
a more elementary book by the same author.
```

User types in 0 on the keyboard.

SAMPLE DIALOGUE 2

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 1
Enjoy the book
```

User types in 1 on the keyboard.

Live Coding

Lec1 –> hello.cpp

Announcements

- Course policy acknowledgement (10 pts)
  - On Blackboard
  - Due on Sunday, February 9th by 11:59pm
- Lab 1 (10 pts)
  - No in-person lab this week or next
  - Due on Sunday, February 9th by 11:59pm on GL
  - Will be released on Blackboard
  - No prelab quiz for lab 1

Next Time: C++ Primer Continued