CMSC202
Computer Science II for Majors

Lecture 01 –
Introduction and C++ Primer

Dr. Katherine Gibson

Course Overview

Course Information

- Second course in the CMSC intro sequence
  - Preceded by 201
- CS majors must pass with a B or better
- CMPE majors must get at least a C

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
  - 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!

About Me

- Dr. Katherine Gibson
  - Education
    - BS in Computer Science, UMBC
    - PhD, University of Pennsylvania
  - Likes
    - Video games
    - Dogs

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)

Review of the Syllabus

- Grading Criteria
- Course Policies
- Attendance
- Communication
- Academic Integrity
- Professor Marron's website (for assignments)

http://www.csee.umbc.edu/courses/undergraduate/202/spring16_marron/

Announcement: Note Taker

A peer note taker has been requested for this class. A peer note taker is a volunteer student who provides a copy of his or her notes for each class session to another member of the class who has been deemed eligible for this service based on a disability. Peer note takers will be paid a $200 stipend for their service. Peer note taking is not a part time job but rather a volunteer service for which enrolled students can earn a stipend for sharing the notes they are already taking for themselves.

If you are interested in serving in this important role, please fill out a note taker application on the Student Disability Services website or in person in the SDS office in Math/Psychology 213.

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

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.

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!

Why C++ for CMSC 202?

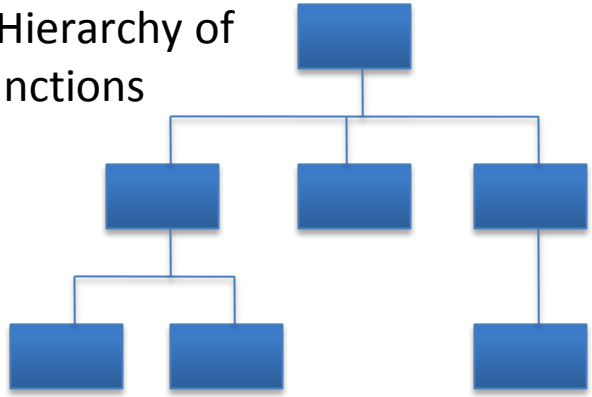
- Popular modern OO language
- Wide industry usage
- Used in many types of applications
- Desirable features
  - Object-oriented
  - Portable (not as much as Java, but fairly so)
  - Efficient
  - Retains much of its C origins

Procedural vs OOP

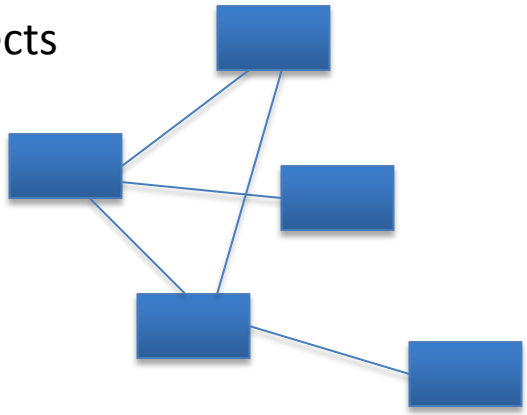
- Procedural
  - Modular units: functions
  - Program structure: hierarchical
  - Data and operations are not bound to each other
  - Examples:
    - C, Pascal, Basic, Python

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

A Hierarchy of Functions

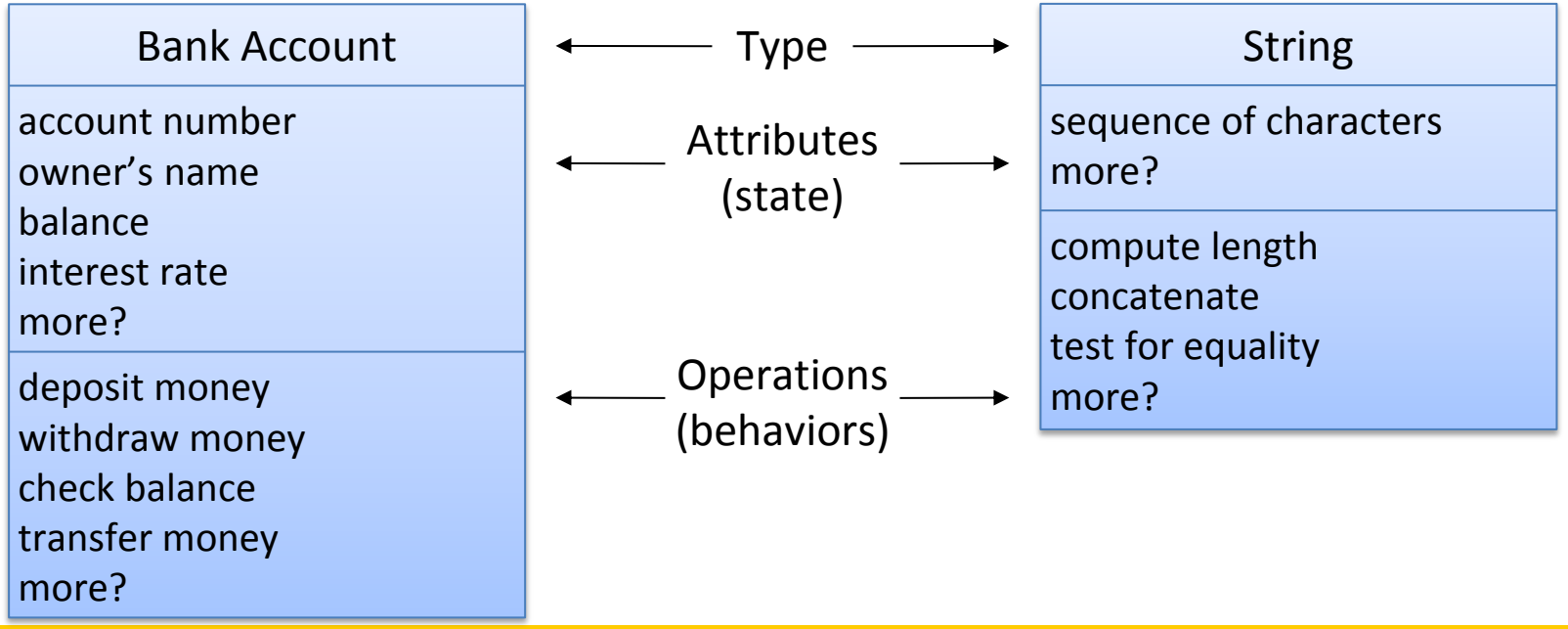


A Collection of Objects



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

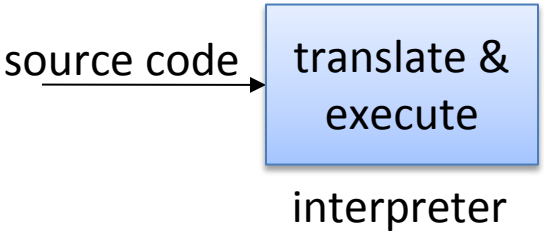
- An ***object*** is a particular instance of a class

Marron's Account	Chang's Account	Gibson's Account
12-345-6 Chris Marron $1,250.86 1.5%	65-432-1 Richard Chang $5.50 2.7%	43-261-5 Katherine Gibson $825.50 2.5%

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

Interpreters, Compilers, & Hybrids

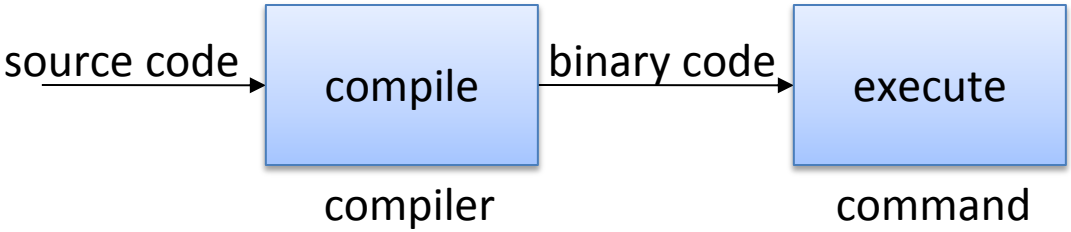
Interpreted Languages (e.g. JavaScript, Perl, Ruby)



Interpreter translates source into binary and executes it

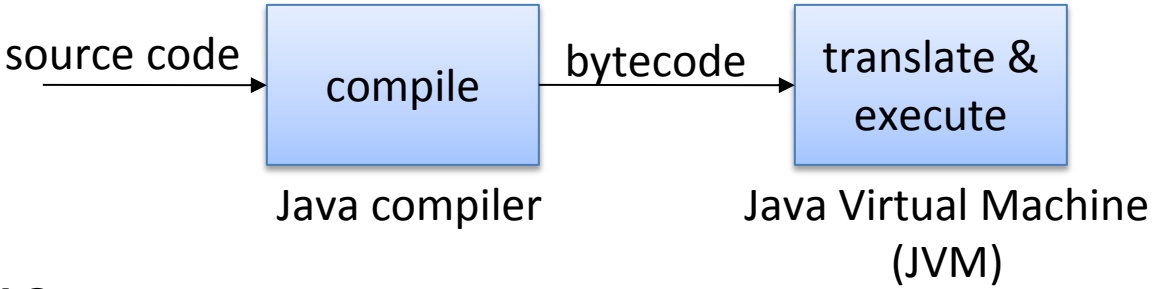
Small, easy to write
Interpreter is unique to each ***platform (operating system)***

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



Compiler is platform dependent

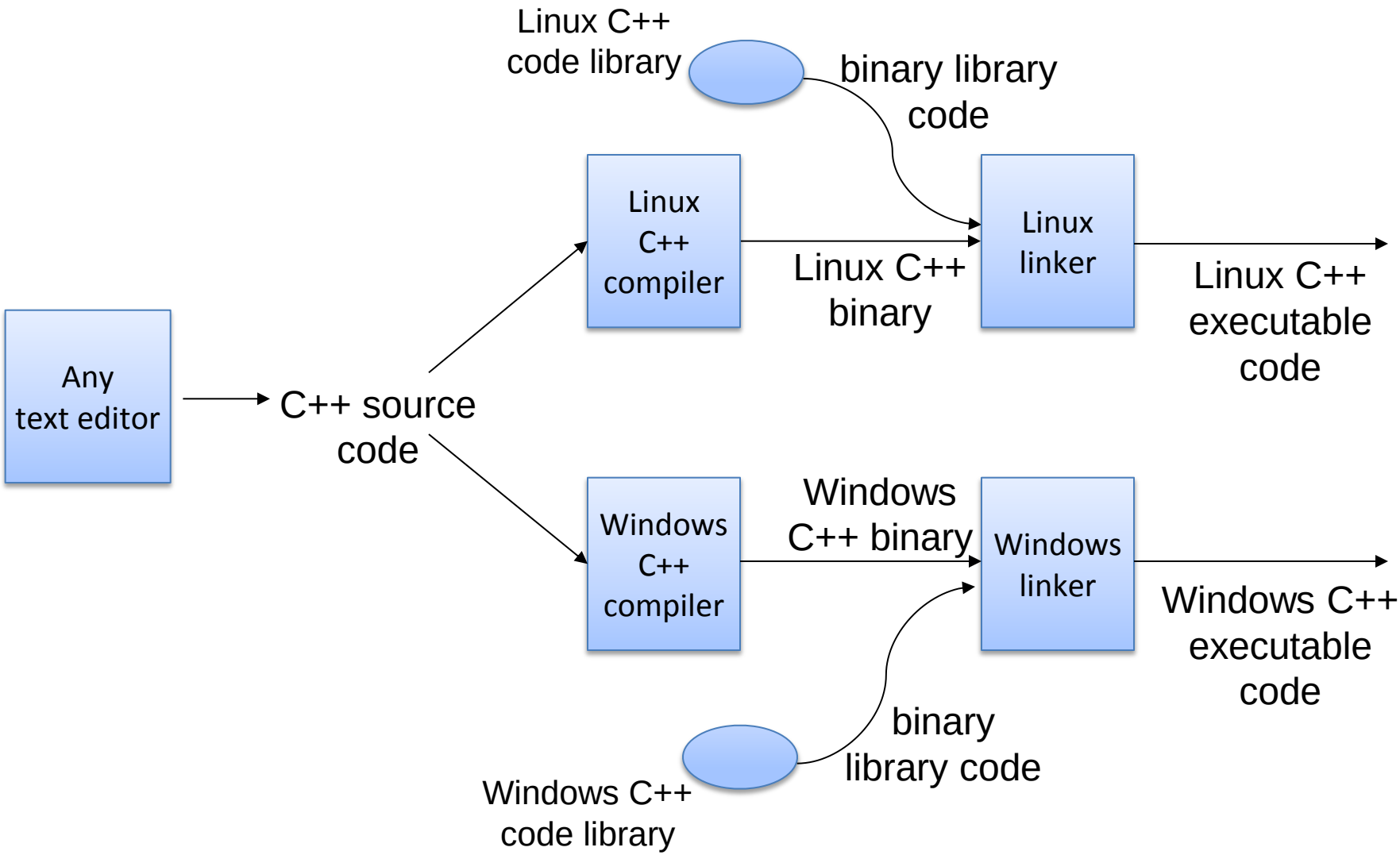
Many other models: e.g., Java (Python is stranger still):



Bytecode is platform independent

JVM is an interpreter that is platform dependent

C++ Compilation and Linkage



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;
}
```

These pieces of code do the same thing!

What's different about these two languages?

Python vs C++ Syntax: Answer

Python

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

- Must have a "**`main()`**" function
- Statements end with "`;`"
- Variables must be declared
- "**`if/else`**" syntax different
- Statement blocks demarcated by "`{...}`"
- But much of it is similar

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;
}
```

C++ Primer

A Sample C++ Program

Display 1.1 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.

C++ Identifiers and Variables

- C++ Identifiers
  - Can't use keywords/reserved words
  - Case-sensitivity and validity of identifiers
  - Meaningful names!
  - Used for variables, class names, and more
- Variables
  - A memory location to store data for a program
  - **Must declare all data before use in program**

Variable Declaration

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

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

Don't forget the semicolon at the end!

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

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

Naming Conventions

- Naming conventions are rules for names of variables to improve readability
  - CMSC 202 has its own standards, described in detail on the course website
    - 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

Display 1.2 **Simple Types**

Important Data 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

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

More Simple Data Types

Important Data Types

`long double`	10 bytes	10^{-4932} to 10^{4932}	
`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

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

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

Assigning Data

- You can initialize data in declaration statement
  - Results will 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`**"

Data Assignment Rules

- Compatibility of Data Assignments
  - Type mismatches
    - Cannot place value of one type into variable of another type
  - **`intVar = 2.99;`** → 2 is assigned to **`intVar`**!
  - Only the integer part "fits", so that's all that goes
  - Called "implicit" or "automatic type conversion"
- Literals
  - **`2`**, **`5.75`**, **`'Z'`**, **`"Hello World\n"`**
  - Also known as "constants": can't change in program

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!

Announcements

- There will be an **important handout** on Tuesday, which will only be available in class
- The Blackboard site will be available soon
- Next Time: Continuation of the C++ Primer, and we'll begin Functions