CSE 1325: Object-Oriented Programming
Lecture 01 – Chapters 01 and 02

Introduction

Mr. George F. Rice
george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Today's Topics

- Overview of CSE 1325
  - What you'll need
  - What you'll learn
  - Class policies
- C++ introduction
  - Brief history and context
  - How to compile and run a program
- Version control
  - How **not** to become a case study in my future lectures

About Me

- 40+ years of computer experience with NASA, Waterways Experiment Station, and General Dynamics / Lockheed Martin
  - Retired at the end of 2015 and began teaching part-time at UTA
- Worked with mainframes, minicomputers, and especially microcomputers
  - Designed early computer architectures, OS, and compilers
  - Programmed early CP/M, Atari, DOS 2.0+, and other 8-bit machines
  - Macintosh OS 1.0+, Windows 1.0+, OS/2 2.0+, and Unix System 7
  - Windows NT 1.0+, Linux 2.0+ including Red Hat 6.0+, Mandrake 7.0+, Ubuntu 4.10+, and SUSE Linux Enterprise Real-Time (SLERT) 10.2+
- Designed and programmed hard real-time embedded chips and motherboards
  - Aircraft test equipment and trainers
  - Data acquisition and processing
- IT architecture and applications for world's largest contract
  -

CSE 1325: Expanding Your Programming Toolkit

- From procedural to object-oriented paradigm
  - Polymorphism + Inheritance + Encapsulation
- Adding a new language, C++
  - And a graphic object-oriented notation, the Unified Modeling Language (UML)
  - And an OO software engineering toolkit, Patterns
- Broadening your knowledge to areas relevant to Science, Technology, Engineering, and Math
  - Respecting “intellectual property”, simplified Scrum process management, debuggers, version control...

Class Schedule (Tentative)

Class Date	Lecture	Chapters	Topic
Thu, Aug 24	1	1, 2	Syllabus, Intro to Object-Oriented Programming and C++; Version Control
Tue, Aug 29	2	3	Classes, Types, and Values; Top-Level Diagram; Intellectual Property
Thu, Aug 31	3	4	Computation; UML Class Diagram; Singleton Pattern
Tue, Sep 5	4	5, 26	Errors, Exceptions, Testing, and Debugging; Extending UML Diagrams
Thu, Sep 7	5	8	Eclectics: Declarations, Arguments, Scope, Namespaces, and Makefiles
Tue, Sep 12	6	(6)	Writing an Object-Oriented C++ Program (Part 1); UML Use Case Diagram; MVC Pattern
Thu, Sep 14	7	(7)	Writing an Object-Oriented C++ Program (Part 2); UML Sequence Diagram; Review
Tue, Sep 19			Exam #1
Thu, Sep 21	8		Return Exam; Intro to Scrum
Tue, Sep 26	9	12	Intro to Graphical User Interfaces (GUI) and Drawing; Façade Pattern
Thu, Sep 28	10	16	GUI Widgets and Dialogs
Tue, Oct 3	11	13, 14	Main Windows, Custom Dialogs, and Callbacks; Observer and Factory Patterns
Thu, Oct 5	12	15	Drawing, Plotting Data, and Designing a Class Library
Tue, Oct 10	13	(16)	Writing a Full C++ GUI Application (Part 1)
Thu, Oct 12	14	(16)	Writing a Full C++ GUI Application (Part 2); Review
Tue, Oct 17			Exam #2 (Last day to drop is Nov 1)
Thu, Oct 19	15		Return Exam; Intro to the Class Project
Tue, Oct 24	16	9	Operator Overloading, Multiple Inheritance; Strategy Pattern
Thu, Oct 26	17	10	Files and I/O
Tue, Oct 31	18	11	Custom I/O; UML Activity Diagram
Thu, Nov 2	19	25	Embedded Programming; UML Statechart Diagram, State Design Pattern
Tue, Nov 7	20	17, 18	Free Store: Pointers, Destructors, and Memory Maps
Thu, Nov 9	21	19	Free Store: Templates, Iterators, and Miscellany;
Tue, Nov 14	22	20, 21	Concurrency and Hyperthreading; UML Deployment Diagram; Adapter Pattern
Thu, Nov 16	23	23, 24	Text Manipulation, Numerics; Anti-Patterns
Tue, Nov 21			Thanksgiving Week
Thu, Nov 23			Thanksgiving
Tue, Nov 28	24	22	Ideals and History, or Guest Lecture Day, or Project Demos
Thu, Nov 30	25		TA Lecture, or Guest Lecture, or Project Demos
Tue, Dec 5	26		Review, or Project Demos (Dec 6 is last day of classes)
Tue, Dec 12			Final Exam: Section 001 (8 am) at 8-10:30 am Section 003 (2 pm) at 2-4:30 pm
Thu, Dec 14			Final Exam: Section 002 (9:30 am) at 8-10:30 am

Recommended Resources

Texts

- Programming: Principles and Practice Using C++, 2nd Edition, Bjarne Stroustrup
  http://stroustrup.com/programming.html
- Unified Modeling Language Reference Manual, 2nd Edition, James Rumbaugh et. al.
  http://dl.acm.org/citation.cfm?id=993859
- A Good Web Browser :-)
  - http://cplusplus.com
  - https://developer.gnome.org/gtkmm/stable/
  - http://stackoverflow.com/
  - And many more

Software

- Ubuntu Linux 16.04[1]
- Umbrello 2.18.3[2] (UML designer)
- Gnu Compiler Collection (gcc) C++ compiler 5.4[3]
  - The Gnu Data Display Debugger (ddd) 3.3.12[2]
  - GIMP Tool Kit-- (gtkmm) 3.0[2]
- git Software Configuration Management (git) version 2.7.4[2]
- Your choice of editor

[1] Any version 14.04 through 16.10, flavors such as LDXE, and derivatives such as Mint or Kiwi will *probably* work
[2] Any later version will *probably* work
[3] Any version 4.8.1 or later will *probably* work

Options for Running Ubuntu

- Use a Virtual Machine (recommended)
  - Use my VirtualBox VM, with *most* tools pre-installed*
  - Install and configure Ubuntu as you like using any VM software, and install the tools yourself
- Run Ubuntu natively
  - Install by itself on a spare machine
  - Dual-boot with Windows or MacOS
- Purchase a machine with Ubuntu pre-installed

Note: Microsoft's Windows Subsystem for Linux is NOT an acceptable long-term solution because it cannot run the GUI application projects you'll develop

Running a Prebuilt VM
(which includes all CSE1325 tools pre-installed)

- Install and launch Oracle VirtualBox (https://virtualbox.org/)
- Select File→Import Appliance…
- In the “Appliance to Import” dialog, select the CSE1325_Lubuntu_1.1.ova file (**2 GB** – yikes!) from
  https://drive.google.com/file/d/0B2kXlVDGGdp1ckpLVV9fbjU5Mk0/view?usp=sharing
- In the “Appliance Settings” dialog
  - Allocate 2048 MB of RAM (if you have at least 4 GB), otherwise, allocated half of your RAM
  - Ensure “Reinitialize the MAC address” is enabled
- Click “Import” and then “Accept”
- Once imported, click CSE1325_Lubuntu_1.1, then Start
- Complete “Sharing a Folder with VirtualBox” on a later slide

Default user ID and password are ‘student’
Change your password immediately!

Installing Your Own VM
(1 of 2)

- Download Ubuntu (https://www.ubuntu.com/download/desktop)
- Install and launch Oracle VirtualBox (https://virtualbox.org/)
- Select Machine → New...
  - In the Create Virtual Machine dialog, set Type to Linux and Version to Ubuntu (64-bit), and name the machine to your liking
  - In the Memory Size dialog, select 2 GB or half your physical RAM, whichever is smaller
  - Select “Create a virtual hard disk now” of *at least* 10 GB (40 is better)
- Select Machine → Settings → Storage
  - Click the icon to the far right of “Optical Drive”, and select “Choose Virtual Optical Disk File...”
  - Select the Ubuntu ISO file you downloaded above
- Select Machine → Start → Normal Start and follow the prompts

Installing Your Own VM
(2 of 2)

- If you plan on sharing a folder with VirtualBox and your host operating system (described on the next slide), you have a few additional steps
  - Open bash, e.g., Ctrl-Alt-t
  - Install Guest Utils, typing your password when prompted:
    `sudo apt-get install virtualbox-guest-utils`
  - Add yourself to the vboxsf group:
    `sudo usermod -a -G vboxsf student`
  - Shutdown:
    `sudo shutdown now`
- Complete “Sharing a Folder with VirtualBox” on a later slide
- Complete “Installing Tools under Ubuntu” on a later slide

Sharing a Folder with VirtualBox
(if you're running VirtualBox pre-built or custom)

- VirtualBox allows you to share a folder(s) between your host computer (e.g., Windows or Mac) and your VM (e.g., Linux)
- Open VirtualBox, but don't start your VM yet
  - Select Machine → Settings → Shared Folders
  - On the right, select the "Adds a shared folder" icon
  - In the Add Share dialog, for Folder Path select Other...
  - In the Select Folder dialog, select the Windows or Mac folder to share, e.g., Documents, and click Select Folder
  - In the Add Share dialog, enable Auto-mount and click OK
  - In the Shared Folders dialog, click OK
- Now click Start, and your folder will appear on the Linux desktop
  - You may want to develop your projects in this shared folder for easy access from either operating system!

Other Thoughts on VirtualBox

- Go / exit full-screen by pressing RIGHT Ctrl-f
- Change your password in bash via **`passwd`**
- Manage your Vms like data
  - Load as many VMs as you like, sharing a vdisk if desired – they don't burn RAM unless they are running!
  - Take snapshots occasionally, for more info see http://news.filehippo.com/2014/06/use-snapshot-virtualbox/
- NEVER close VirtualBox while a machine is running!
  - This is like pulling the desktop PC cord from the wall!
  - Instead, shut down via the menu or via bash's **`sudo shutdown now`**

Running Ubuntu Natively

- Install natively or via dual boot
  - https://www.ubuntu.com/download/desktop/install-ubuntu-desktop
  - rEFInd (https://sourceforge.net/projects/refind/) seems to be highly regarded for managing boot images on a Mac
- Purchase a machine with Ubuntu pre-installed
  - Try e.g., http://dell.com/developers, http://system76.com, or http://emperorlinux.com/

Installing Tools under Ubuntu

(if you installed your own VM or are running natively)

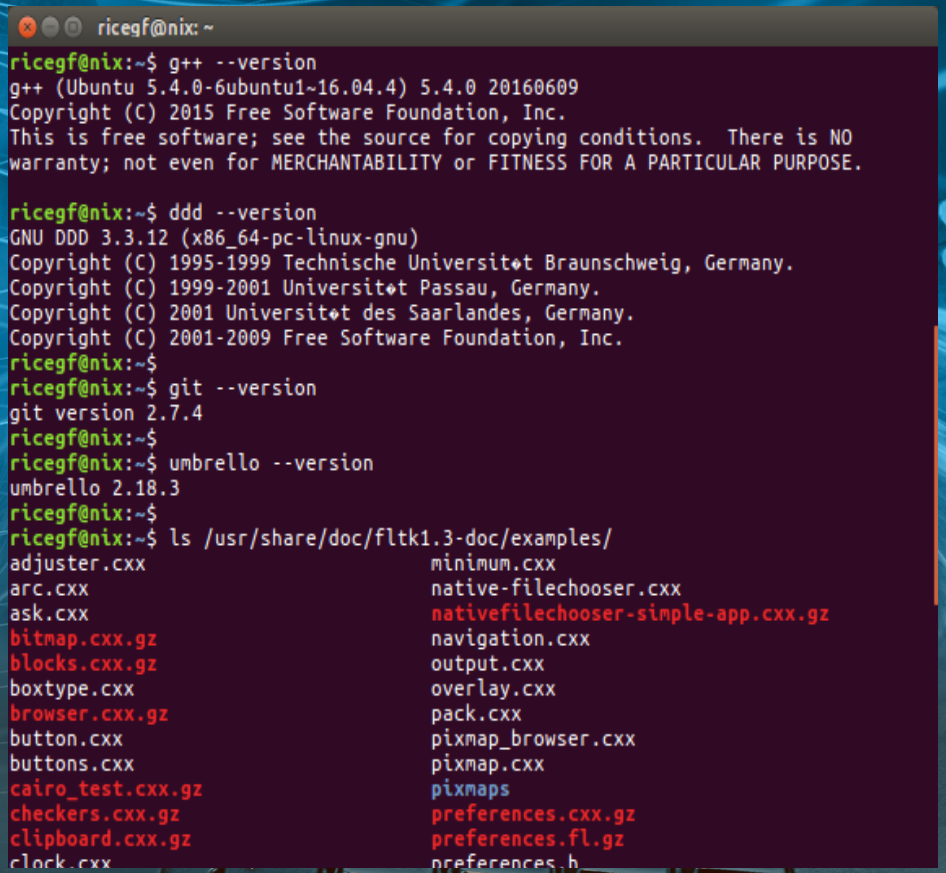
- All CSE 1325 tools are available in the 16.04 repository for installation using standard apt-get
  - Press Ctrl-Alt-t (or double-click LXTerminal on the desktop, or click  and type “terminal”)
  - Install →

    Note: You may install Umbrello natively on your Windows or Mac computer instead

```
$ #Do NOT type the $ - that's a prompt!
$ sudo apt-get update
$ sudo apt-get install build-essential
$ sudo apt-get install ddd
$ sudo apt-get install libgtkmm-3.0-dev
$ sudo apt-get install libgstreamermm-1.0-dev
$ sudo apt-get install libgtkmm-3.0-doc
$ sudo apt-get install libgstreamermm-1.0-doc
$ sudo apt-get install devhelp
$ sudo apt-get install gtk-3-examples
$ sudo apt-get install git-all
$ sudo apt-get install umbrello
$ sudo apt-get install kio
$ sudo apt-get install oxygen-icon-theme
```

bash

- Linux (and MacOS) rely on the bash shell
  - Bash is a Command Line Interface (CLI)
  - Programmers often prefer CLI shells because they are more efficient
  - You will learn (**and be tested on**) bash this year
- See “Bash in 5 Pages” on Blackboard to get started



Final Grade Composition

Pop Quizzes : 10%

Homework: 30%

(Roughly 3% per week of assigned work)

2 Exams + Final: 20% Each

Make-Up Exams

- Once started, the exam will not be made up
- A valid excuse (e.g., doctor's note) for an **unexpected** issue is **required** to avoid a zero on a missed exam
  - The make-up exam, if offered, **may be different** from that given to those attending on exam day
  - At the professor's discretion, an excused missed exam grade may be replaced by the Final Exam grade
  - If excused, the Final Exam will be marked Incomplete and a make-up exam scheduled

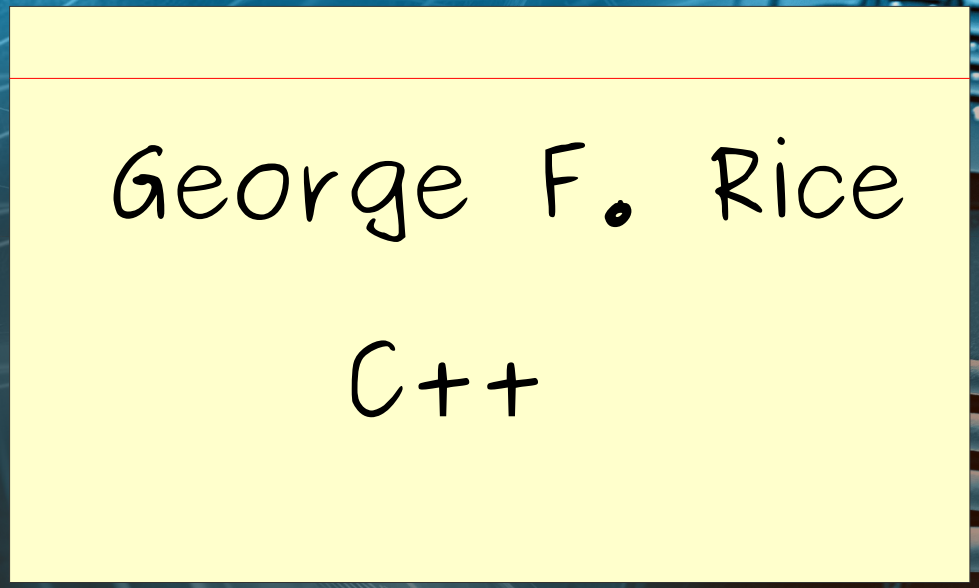
Extra Credit

- Homework will often have “Bonus Levels” for extra credit
  - Completing the Full Credit portion of the homework successfully earns 100%
  - Bonus, Advanced Bonus, and Extreme Bonus levels may offer up to 25% additional credit
  - Homework averages over 100% DO count toward final grade and can compensate for exam mistakes
- No additional extra credit work is available

Start Early and Avoid the Rush

Attendance

- I will not call roll
- Many classes will include a 1 minute pop quiz
  - This is principally to take attendance
  - Always bring a 3x5 index card to class!



A Brief Regression to Philosophy

Engineers Solve Problems

“The ultimate aim of programming
is always to produce useful systems.”
– Bjarne Stroustrop

http://stroustrup.com/

“Talk is cheap.
Show me the code.”
– Linus Torvolds

“Most Popular” Languages

	TIOBE	IEEE	RedMonk
1	Java	C	JavaScript
2	C	Java	Java
3	C++ ←	Python	PHP
4	Python	C++ ←	Python
5	Visual Basic .NET	R	C++ ←
6	C#	C#	C#
7	PHP	PHP	Ruby
8	JavaScript	JavaScript	CSS
9	Assembly	Ruby	C
10	Perl	Go	Objective C
	Dec '16	Jul '16	Jun '16

Tie

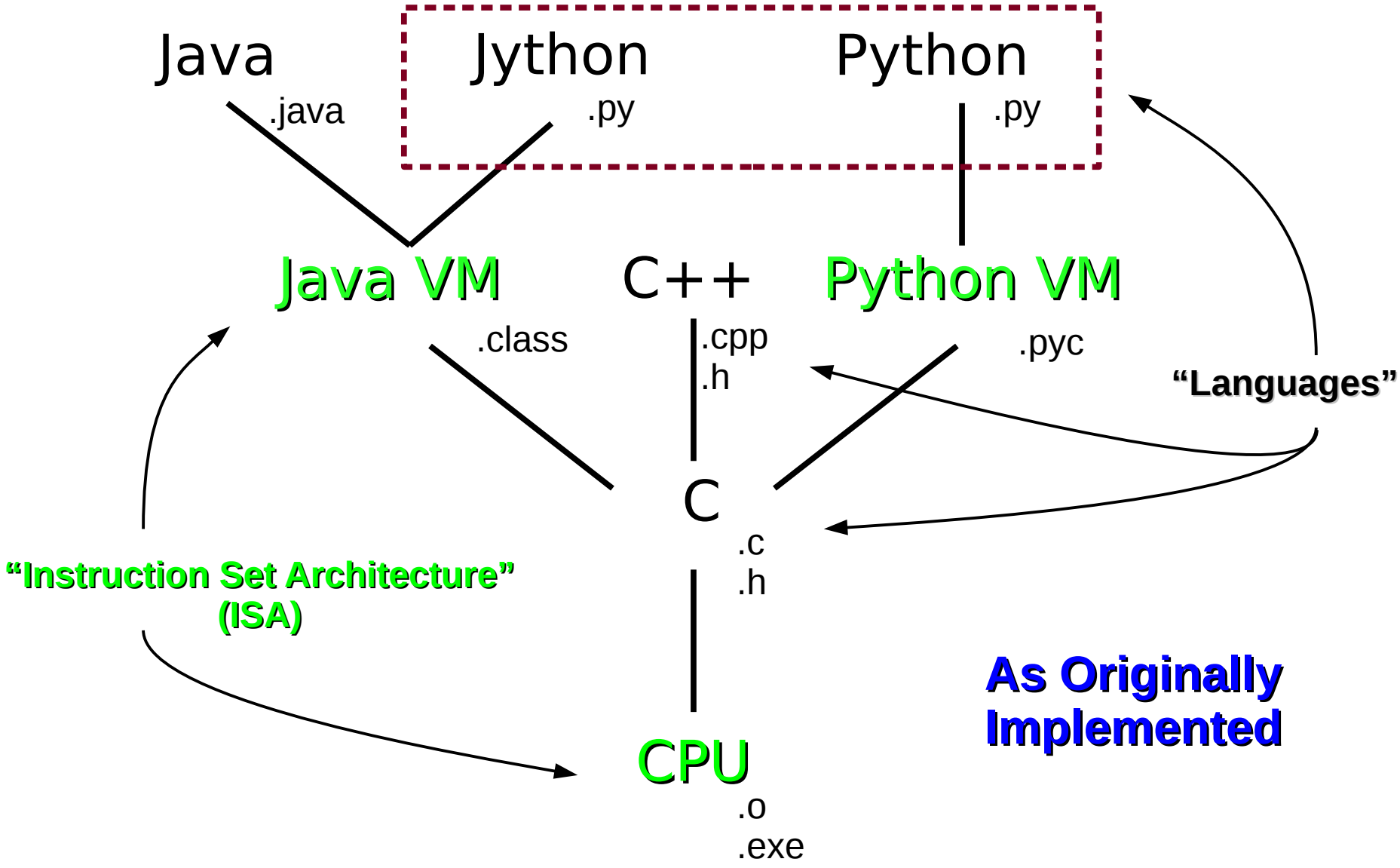
Based on the number of skilled engineers world-wide, courses and third party vendors

Driven by weighting and combining 12 metrics from 10 data sources such as the IEEE Xplore digital library, GitHub, and CareerBuilder. The weighting of these sources can be adjusted in their interactive Web app

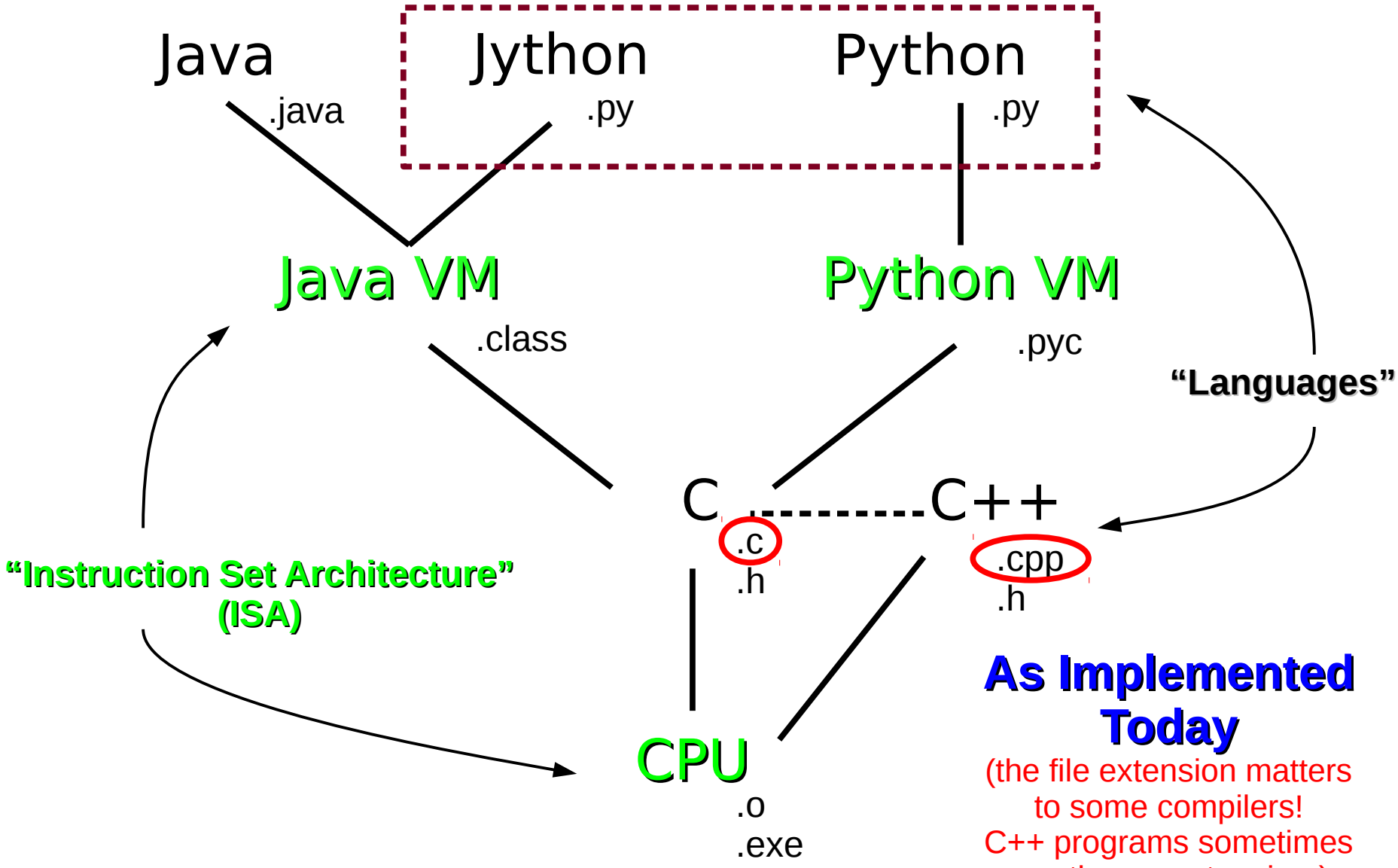
Comparison of programming languages relative to one another on GitHub and Stack Overflow

http://tiobe.com/tiobe_index http://redmonk.com/sogrady/2016/07/20/language-rankings-6-16/
http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

Language Hierarchy



Language Hierarchy



What's In a Binary?

Python

```
>>> dis.dis(hello)
  2           0 LOAD_CONST
               1 ('Hello, world!')
              3 PRINT_ITEM
              4 PRINT_NEWLINE
              5 LOAD_CONST
               0 (None)
              8 RETURN_VALUE
>>>
```

Python Virtual Machine Code

5 lines

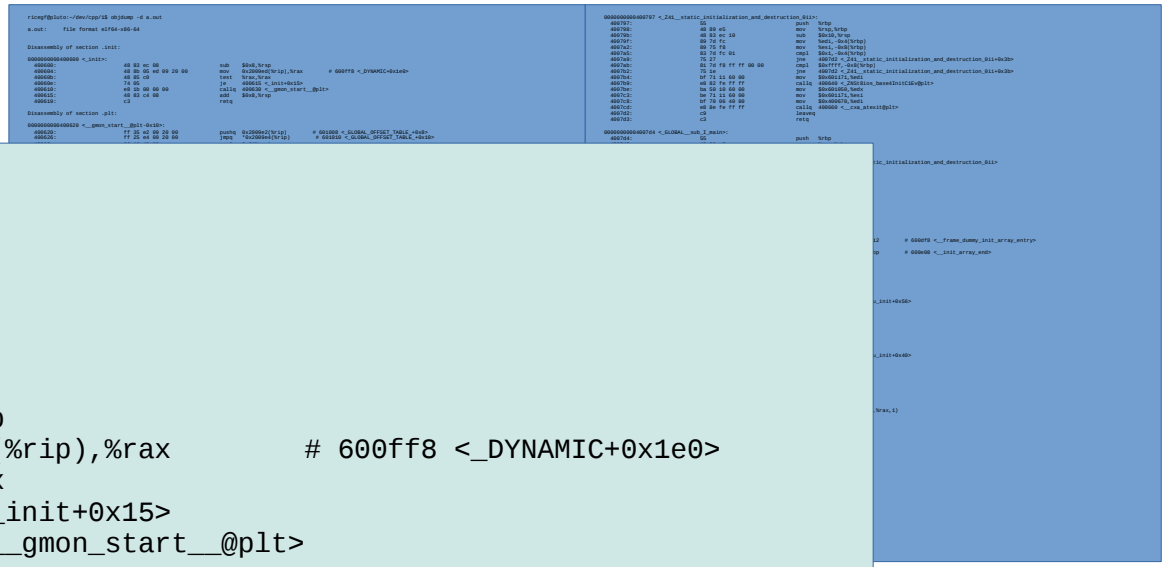
Java

```
ricegf@pluto:~/dev/cpp/1$ javap -c HelloWorld.class
Compiled from "HelloWorld.java"
public class HelloWorld {
  public HelloWorld();
    Code:
       0: aload_0
       1: invokespecial #1
                   // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2
                   // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3
                   // String Hello, World
       5: invokevirtual #4
                   // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
ricegf@pluto:~/dev/cpp/1$
```

Java Virtual Machine Code

15 lines

What's In a Binary?



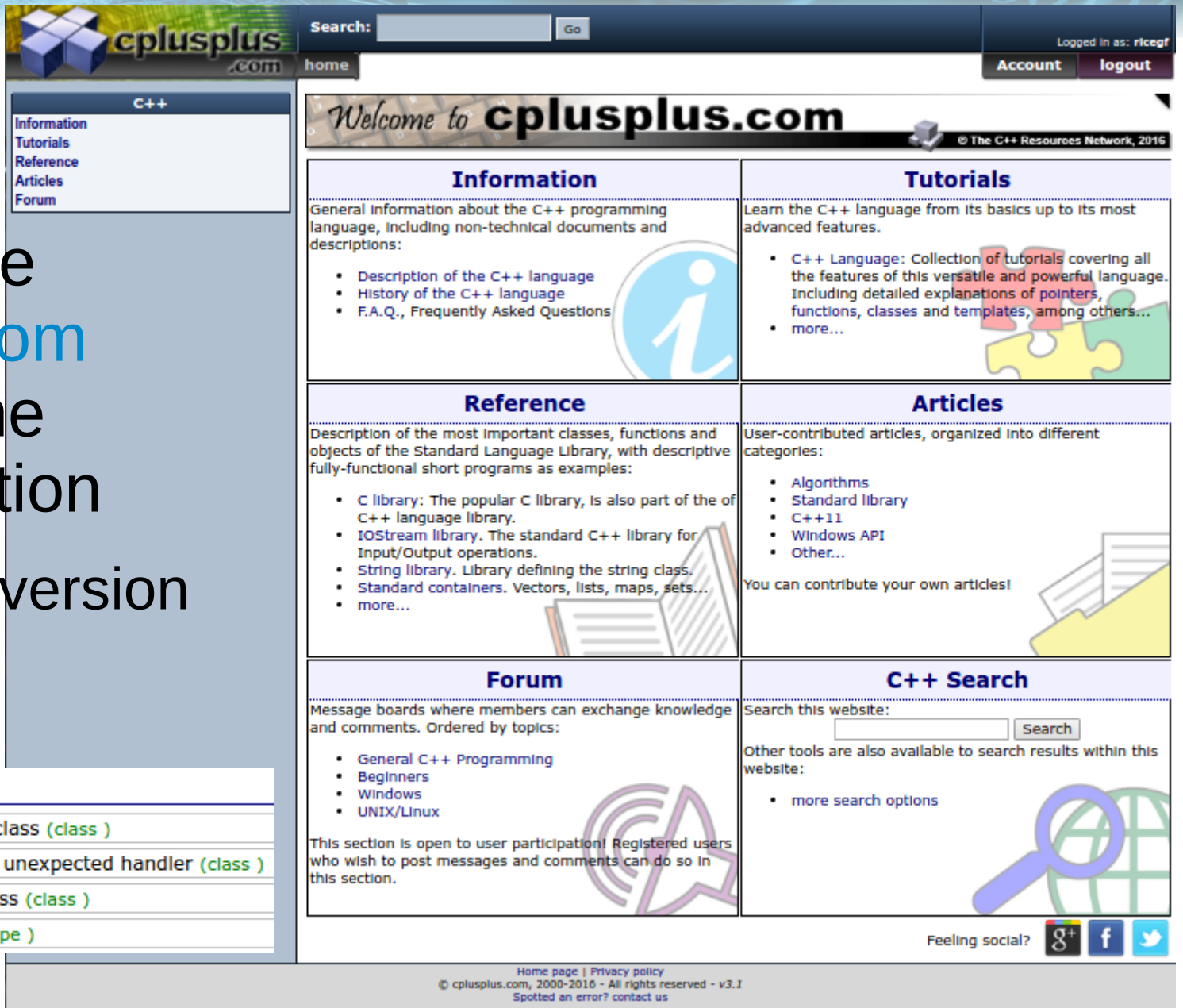
```
ricegf@pluto:~/dev/cpp/1$ objdump -d a.out

a.out:     file format elf64-x86-64


Disassembly of section .init:

0000000000400600 <_init>:
  400600:   48 83 ec 08             sub    $0x8,%rsp
  400604:   48 8b 05 ed 09 20 00    mov    0x2009ed(%rip),%rax        # 600ff8 <_DYNAMIC+0x1e0>
  40060b:   48 85 c0                test   %rax,%rax
  40060e:   74 05                   je     400615 <_init+0x15>
  400610:   e8 1b 00 00 00          callq  400630 <__gmon_start__@plt>
  400615:   48 83 c4 08             add    $0x8,%rsp
  400619:   c3                      retq

Disassembly of section .plt:

0000000000400620 <__gmon_start__@plt-0x10>:
  400620:   ff 35 e2 09 20 00       pushq  0x2009e2(%rip)        # 601008 <_GLOBAL_OFFSET_TABLE_+0x8>
  400626:   ff 25 e4 09 20 00       jmpq   *0x2009e4(%rip)        # 601010 <_GLOBAL_OFFSET_TABLE_+0x10>
  40062c:   0f 1f 40 00             nopl   0x0(%rax)

…and so on
```

222 lines

x64 Native Machine Code

Where's the Standard C++ Docs?

- C++ has none
- But we'll use cplusplus.com as our online documentation
  - Watch for version identifiers



Types:

exception	Standard exception class (class)
bad_exception	Exception thrown by unexpected handler (class)
nested_exception C++11	Nested exception class (class)
exception_ptr C++11	Exception pointer (type)

Become Familiar with Internet Programming Resources

- Here are a few to get you started
  - stackoverflow
  - The animal books from O'Reilly 

  - Cplusplus beginner forum 

- My job is less to teach you *how* to program, and more to teach you how to *learn* to program
  - Technology changes constantly
  - You will be learning for the rest of your ~~career~~ life

Writing the Canonical 1st Program

Python:

Structured
Object-Oriented

```
print("Hello, World")
```

C:

Structured

```
#include <stdio.h>
main() {
   printf("Hello World");
}
```

Java:

Object-Oriented

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

C++:

Structured
Object-Oriented

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello World!" << endl;
}
```

Don't worry about these yet

Running hello.cpp Manually

Via Bash

```
ricegf@pluto: ~/dev/cpp/201708/01
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~$ cd dev/cpp/201708/01
ricegf@pluto:~/dev/cpp/201708/01$ ls
hello.cpp
ricegf@pluto:~/dev/cpp/201708/01$ cat hello.cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello, World!" << endl;
}
ricegf@pluto:~/dev/cpp/201708/01$ g++ hello.cpp
ricegf@pluto:~/dev/cpp/201708/01$ ./a.out
Hello, World!
ricegf@pluto:~/dev/cpp/201708/01$
```

List the files in this directory

List the contents of hello.cpp

Convert hello.cpp into a.out

Run (execute, launch...) a.out

Running hello.cpp via the Makefile System

Via Bash

```
ricegf@pluto: ~/dev/cpp/201708/01
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~/dev/cpp/201708/01$ cat hello.cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello, World!" << endl;
}
ricegf@pluto:~/dev/cpp/201708/01$ cat Makefile
hello: hello.cpp
        $(CXX) -o hello hello.cpp

ricegf@pluto:~/dev/cpp/201708/01$ make hello
g++ -o hello hello.cpp
ricegf@pluto:~/dev/cpp/201708/01$ ./hello
Hello, World!
ricegf@pluto:~/dev/cpp/201708/01$
```

The Makefile describes how to make the executable hello

This make command follows the above rules to make hello.

Writing the Canonical 1st Program Stroustrop Style

C++:

Object-Oriented

```
#include "std_lib_facilities.h"
int main() {
  cout << "Hello World!";
  return 0;                    // Linux/Mac enhancement
  keep_window_open();          // Windows enhancement
}
```

Also search “locally”

```
ricegf@pluto:~/dev/cpp/1$ g++ hello_world_stroustrup.cpp
In file included from /usr/include/c++/4.8/ext/hash_map:60:0,
                 from std_lib_facilities.h:34,
                 from hello_world_stroustrup.cpp:1:
/usr/include/c++/4.8/backward/backward_warning.h:32:2: warning: #warning This fi
le includes at least one deprecated or antiquated header which may be removed wi
thout further notice at a future date. Please use a non-deprecated interface wit
h equivalent functionality instead. For a listing of replacement headers and int
erfaces, consult the file backward_warning.h. To disable this warning use -Wno-d
eprecated. [-Wcpp]
 #warning \
  ^
ricegf@pluto:~/dev/cpp/1$ ./a.out
Hello World!
ricegf@pluto:~/dev/cpp/1$
```

NOT Recommended

Download from http://stroustrup.com/Programming/std_lib_facilities.h

Trouble with the Façade

- Dr. Stroustrop provides a “façade”* - a wrapper to make C++ “easier” to learn
  - The intent is to make C++ “easy enough” to learn as a first language – but this is NOT your first language!**
  - This façade sometimes works
  - But it often fails in unexpected and spectacular ways
- We will **NOT** use std_lib_facilities.h in class, homework, OR exams
  - We will use the C++ 11 (or 14) standard with libraries
  - If you use the textbook, be advised of this difference

* We'll learn much more about the Façade pattern in lecture 9
** If it is, please see me ASAP!

What is “Correct C++”?

- Answer #1: It conforms to the C++ 11 standard
  - https://isocpp.org/std/the-standard
  - Warning: You can't easily learn C++ from the standard!
- Answer #2: It works with the compiler
  - For homework and exams, this is gcc 5.4 on Ubuntu 16.04
- Answer #3: It is “better” than alternate answers
  - “Better” is requirements-dependent, and *may* mean:
    - “faster”
    - “more memory efficient”
    - “more maintainable”
    - “more flexible”

The “stakeholders” decide!

Special Topic: Version Control

- “The ~~dog~~ computer ate my homework!”

You'll see definitions marked as “Expertise” again. Soon. Likely on an exam!

Version Control - The task of keeping a software system consisting of many versions and configurations well organized

A Simple Git Session...

```
ricegf@pluto:~$ mkdir test
ricegf@pluto:~$ cd test
ricegf@pluto:~/test$ vi hello.cpp
ricegf@pluto:~/test$ git init
Initialized empty Git repository in /home/ricegf/test/.git/
ricegf@pluto:~/test$ git add hello.cpp
ricegf@pluto:~/test$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hello.cpp

ricegf@pluto:~/test$ git commit
[master (root-commit) 879d8a0] Initial version
 1 file changed, 7 insertions(+)
 create mode 100644 hello.cpp
ricegf@pluto:~/test$ git log
commit 879d8a0648241f7c521ba851b59f93aa6b7b35e7
Author: ricegf <george.rice@uta.edu>
Date:   Wed Aug 24 19:59:52 2016 -0500

    Initial version
ricegf@pluto:~/test$
```

Create one or more valuable files (you might consider nano instead of vi)

Initialize this folder to use git

Add valuable files to git's list of files to be protected.

They aren't protected *yet*, though – git just knows you care about them.

Commit the file(s) to git's protection. It will open your text editor so you can describe the submission.

Ask git for a list (“log”) of every commit you've made – its “hash” (internal name – just call it 35e7 for short), your name and date, and the comment you typed on submission.

...Can Save the Day!

```
ricegf@pluto:~/test$ ls
hello.cpp
ricegf@pluto:~/test$ cat hello.cpp
 #include "std_lib_facilities.h"
 int main() {
   cout << "Hello, World!" << endl;
   return 0;
   keep_window_open();
 }

ricegf@pluto:~/test$ shred hello.cpp
ricegf@pluto:~/test$ head -1 hello.cpp
eq◆◆8 ◆◆◆ ◆◆d◆◆6
ricegf@pluto:~/test$ #woops
ricegf@pluto:~/test$ git checkout hello.cpp
ricegf@pluto:~/test$ cat hello.cpp
 #include "std_lib_facilities.h"
 int main() {
   cout << "Hello, World!" << endl;
   return 0;
   keep_window_open();
 }

ricegf@pluto:~/test$
```

Our beloved file, blissfully unaware that disaster is only seconds away!

What's that compile command again?

Uh oh – something looks... different. When is this due again?

Every version of every file that you've ever “committed” to git can be recalled (“checked out”) non-destructively. If you want to see how a file looked 113 commits ago, it's less than a second away!

Saved by version control... again!!!

Git Basics
“git init”

- Initialize the current directory as a “repository”
  - Creates a subdirectory named “.git”
  - “.git” contains all of the tracking information
  - “.git” does NOT stand alone –
    you need the current directory, too!
- WARNING: This repo is LOCAL
  - If you delete the directory, you delete the repository!
  - Backups! Backups! **Backups!!!**

```
ricegf@pluto:~$ mkdir test
ricegf@pluto:~$ cd test
ricegf@pluto:~/test$ git init
Initialized empty Git repository in /home/ricegf/test/.git/
ricegf@pluto:~/test$ ls
ricegf@pluto:~/test$ ls -a
.  ..  .git
ricegf@pluto:~/test$ ls -al .git
total 40
drwxrwxr-x 7 ricegf ricegf 4096 May 23 14:52 .
drwxrwxr-x 3 ricegf ricegf 4096 May 23 14:52 ..
drwxrwxr-x 2 ricegf ricegf 4096 May 23 14:52 branches
-rw-rw-r-- 1 ricegf ricegf   92 May 23 14:52 config
-rw-rw-r-- 1 ricegf ricegf   73 May 23 14:52 description
-rw-rw-r-- 1 ricegf ricegf   23 May 23 14:52 HEAD
drwxrwxr-x 2 ricegf ricegf 4096 May 23 14:52 hooks
drwxrwxr-x 2 ricegf ricegf 4096 May 23 14:52 info
drwxrwxr-x 4 ricegf ricegf 4096 May 23 14:52 objects
drwxrwxr-x 4 ricegf ricegf 4096 May 23 14:52 refs
ricegf@pluto:~/test$
```

Git Basics

Introduce Yourself to Git

- The first time you use git, give it your name
  - “git config --global user.name ricegf”
  - “git config --global user.email george.rice@uta.edu”
  - In case it’s not obvious, use your name and email!
- You only need to do this once per account

```
ricegf@pluto:~/test$ git config --global user.name ricegf
ricegf@pluto:~/test$ git config --global user.email george.rice@uta.edu
ricegf@pluto:~/test$ git config --get user.name
ricegf
ricegf@pluto:~/test$ git config --get user.email
george.rice@uta.edu
ricegf@pluto:~/test$
```

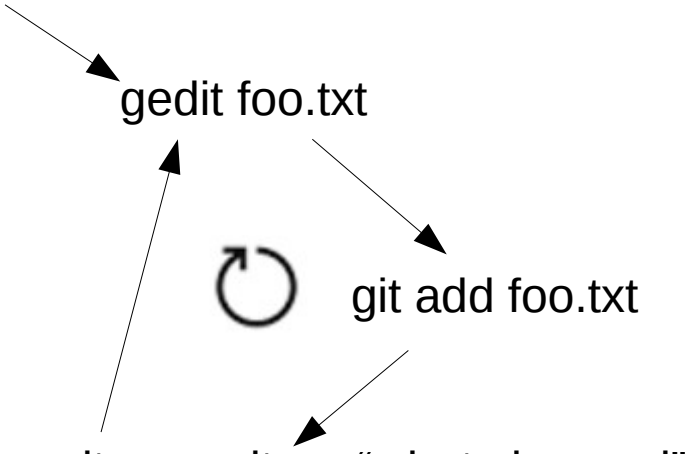
Git Basics
“git status”

- “git status” tells you about files in the current directory
  - “fatal: Not a git repository” means you need to type “git init”
  - “nothing to commit” means the repository is up to date
  - “untracked files” lists files that git is ignoring
  - “changes not staged for commit” lists files that git is watching, but that won’t be added to git next commit. If you want them added, use “get add” (next slide)
  - “changes staged for commit” are the files that git will add to the repository next commit because you typed “get add”

```
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   library_gui.cpp

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working dire

        modified:   Makefile
        deleted:    library_cli.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        gui
```

Git Basics
“git add”

- “git add” tells git to add these files to the repository next commit (covered soon)
  - If git wasn’t watching them before, it is now
  - Regardless, next commit the current file contents will update git’s repository
  - You can “undo” an add with “git reset”




```
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   library_gui.cpp

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working dire

        modified:   Makefile
        deleted:    library_cli.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        gui
```

Git Basics
“git checkout”

- “git checkout <filename>” restores filename to its contents as of the last commit
  - This can be an undelete, to restore a file you accidentally deleted
  - This can be an unmodify, to throw away all changes since the last commit

```
ricegf@pluto:~/dev/cpp/201701/P6/fc$ ls
gui  library_gui.cpp  Makefile
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git checkout library_cli.cpp
ricegf@pluto:~/dev/cpp/201701/P6/fc$ ls
gui  library_cli.cpp  library_gui.cpp  Makefile
ricegf@pluto:~/dev/cpp/201701/P6/fc$
```

Git Basics
“git diff”

- “git diff <filename>” shows the changes you made to a file since the last commit
  - Deletions are in red preceded by “-”
  - Additions are in green preceded by “+”
  - Unmodified lines in white add context

```
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git diff Makefile
diff --git a/Makefile b/Makefile
index 478b707..35b13a3 100644
--- a/Makefile
+++ b/Makefile
@@ -1,17 +1,17 @@
 # Makefile for Library
-CXXFLAGS = -std=c++11
+CXXFLAGS = -w -std=c++11
+LDFLAGS = -L/usr/local/lib -lfltk -lXext -lX11 -lm

 all: gui

 debug: CXXFLAGS += -g
-debug: library
+debug: gui
+
+rebuild: clean gui

-rebuild: clean library
 gui: library_gui.cpp
-	fltk-config --compile library_gui.cpp
-	mv library_gui gui
-cli: library_cli.cpp
```

Git Basics
“git commit -m”

- “git commit -m <message>” puts added files into git
  - Add a message in double quotes after -m
    - Keep it short – this will help you find changes later
- “git log” lists all commits
  - Add “--pretty=oneline” to keep it concise

```
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git commit
Aborting commit due to empty commit message.
ricegf@pluto:~/dev/cpp/201701/P6/fc$ vi Makefile
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git commit -m "Update Makefile for gui"
[master c78e59f] Update Makefile for gui
 1 file changed, 120 insertions(+), 63 deletions(-)
ricegf@pluto:~/dev/cpp/201701/P6/fc$ git log --pretty=oneline
c78e59ff97781839f434091761f837bb450d0c7c Update Makefile for gui
e587c1dff9a20543f552b22d99086fb17ba3fa30 Patron list now gui
4be0e9c810677883134b565b42583ace3850ae7e Adding patron now gui
ffde3560f9ca8ae991433a5aaff134d9454a4e93 Check in now gui
5b2752db4af35560ae4dd63496203e65623c5ef9 Check out now gui
27b4c5af0e6d5c4050f34579d674f18333f20536 Publications list now gui
8ab90a450cb89cba2f6d834a16db636a929702b8 Adding a pub now all gui
4e8d962999e5e9c26df33680222e22db492bfe2f Split CLI and GUI, gui main menu
2be472e11aafb620ac2c64350d983a46ede6b3bf Added data validation
2b7de9c80c3b9b59b6b0f8fef41961379aed38ef works well, no error handling
44857b62cd6600bb644c5dd8fc4327a3eb72f692 runs but core dump on (6)
ricegf@pluto:~/dev/cpp/201701/P6/fc$
```

Get git. Learn git.

Pre-installed in the CSE1325-Lubuntu VM

https://git-scm.com/download/windows

https://git-scm.com/download/mac

sudo apt-get install git-all

sudo yum install git-all

“Git in 5 Pages” from Blackboard

https://git-scm.com/book

Git is Half of the Answer
Make Backups!

- All Technology Eventually Fails™
- Backup options (pick at least two)
  - Duplicate your git repository periodically
    - Add (exactly!!!) this one line to the end of your ~/.bashrc file
      ```
      alias backup='DIR=../$(basename $PWD)-$(date +%Y%m%d-%H%M%S);mkdir -p $DIR;cp -ru . $DIR'
      ```
    - Now type “backup” in a *new* bash shell to duplicate the current directory
  - Copy your git repository to a flash or portable drive
    - These usually mount in the file manager automatically
    - Keep them unmounted and off when not backing up or restoring!
  - Copy your git repository to the cloud
    - Git-compatible “origin” server such as Github, Bitbucket, ...
    - Autosync options like Dropbox, Google Cloud, SpiderOak, ...
    - Manual sync options like ownCloud, sftp, ...

Photograph of the Montparnasse derailment in Paris, France on October 22, 1895 is no longer subject to US Copyright and thus in the Public Domain

Quick Review

- Which tools and environments will be used for homework assignments and projects this semester?
- The task of keeping a software system consisting of many versions and configurations well organized is called _____ _____.
- Specify the git command for each of the following actions:
  - Initialize a local git repository
  - Add / update a file to *later* commit to the local repository
  - Commit added / updated files to the repository
  - Undelete a file by retrieving from the repository
  - Compare the differences between two file versions
- Does adding a file to a local git repository store the file?
- List some options for backing up your class work.
  - Why is this important?

For Next Class

- Install and configure software
  - Set up Linux however you prefer and install Umbrello, gcc, ddd, fltk, and git

  OR

  - Install VirtualBox on your laptop and load the predefined VM
- Read and practice Bash in 5 Pages and Git in 5 Pages
- (Optional) Read Chapters 1 and 2 in Stroustrop
  - **Do the Drills!**
- Skim Chapter 3 for next lecture

You learn to program by programming!

NOW is the time to start. A week before the first homework will be too late!

Homework #1

- **Build and Run "Hello World"** ***using your own name***
  - **Bonus:** Ask the user for a name and use that
  - **Extreme Bonus:** Determine current user's name *without asking* and use that
- Deliver code and screen shots to Blackboard by **Thursday, 31 August at 8 am**
- **Details are provided on Blackboard**

(HINT: These are the easiest points you'll earn all semester. Don't blow it!)

You learn to program by programming!

NOW is the time to start. A week before the first homework will be too late!

Why 8 a.m. on the Due Date?

- This →
- Late night is traditionally "coding time"
- But most important, we can review the Suggested Solution <u>the same day</u> that you submit yours
  - Faster feedback accelerates learning

Next Lecture

- First taste of OOP: Classes, Types, and Values
- Second taste of git: The Git GUI
- Intro to UML and the Top-Level Diagram
- Intellectual Property Basics: Copyright, Trademark, Patents, and the rest

Remember: Bash in 5 Pages and Git in 5 Pages

Why 8 a.m. on Due Date?

- That →
- Late night is traditionally "coding time"
- But most important, we can review the Suggested Solution shortly after you submit yours
  - Faster feedback accelerates learning