02. Environments, Flowcharts, Notes, Pair Programming

CPSC 120: Introduction to Programming

Pratishtha Soni ~ CSU Fullerton

Agenda

0. Sign-in sheet
1. Notes
2. Environments
3. Flowcharts
4. Pair Programming

1. Notes

Note taking

Notes and notebooks are integrated into our

- lecture
- labs
- exams
- flowcharts (explained soon)

Purpose of note taking

- Focus
- Learning
  - 3 repetitions
  - kinesthetic
  - summarizing
- Record of what was covered
- Quick reference in lab
  - Patterns
  - Problems and solutions
- Exams
  - Study
  - Open-note

Considerations

- Need to summarize
  - Instructors speak ≈125-140 words/minute
  - Write ≈25 words/minute
- Highlight major points
- Today: two methods
  - Sentence/paragraph
  - Outline

Signals

Listen to words, look at slides/whiteboard/gestures

Main ideas	Supporting info.	Conclusion/summary
“Our agenda for today...” “There are five parts....” “First...second...third...” “The key point is...”	“Similarly...” “Also...” “On the other hand...” “For example...”	“In summary...” “Finally...” “Therefore...” “In conclusion...”

Sentence/paragraph method

- Record of complete thoughts
- Write every new thought, fact or topic presented
- Use a “new” line for each new thought/fact
- Number lines as you progress

Review: Digital Computers

- Now, **computer** means a machine with...
- **Central Processing Unit (CPU)**
  - a.k.a. **processor** or **chip**
  - E.g. Intel Core i3 or Qualcomm Snapdragon
  - "brain;" executes **instructions**, e.g. add two numbers
- **Memory** (a.k.a. RAM): fast, volatile, small; stores data in use
- **Storage** (a.k.a. disk): slow, nonvolatile, large; stores data when computer is off
- **Input/Output (I/O)**: screen, keyboard, mouse...
- **Peripherals**: other stuff (wifi, battery...)
- **Bus**: connects these

Example: sentence/paragraph

1. Computer parts
2. CPU - exec. instructions
3. memory - fast, volatile (off), small
4. storage - slow, not vol., larger
5. I/O: screen, keyb., etc
6. bus: connects ↑

abbreviations and symbols to speed up writing

Numbered ideas, facts, thoughts

Outline method

- Left: general, broad information
- Right: more specific, clarification
- Major points on left column
- Indent more specific points
- Distance from left indicates importance

Review: Edit-Compile-Run Cycle

A programmer…

1. **Edits** source code (C++) in a text file (`main.cpp`)
2. Runs compiler (clang++) to **compile** source code into object code (`main.cpp` becomes `./a.out`)
   a. If the compiler understood everything you wrote, this works
   b. Otherwise (it is confused), it gives feedback **messages**; keep editing
3. **Runs** the object code (`./a.out`)
4. **Tests** if it works; if not, keep editing
5. **Release** the object code (`./a.out`) to **users**
   a. they can run it at-will, without the source code (myprog.cpp)

You will do 1-4 in lab, and 5 when you submit your work.

Example: outline method

edit-compile-run cycle = workflow ← main point

- edit - source code - editor ← specific point
- compile
  - clang++
  - object code
  - compile errors ⟶ edit again

  ← even more specific points
- run
- test
  - doesn't work ⟶ back to edit
- release - done

When to take notes

- Reading assignments
- Lecture
- Explicitly told to by...
  - ...flowcharts
  - ...lab instructions
- When you solve a problem
  - Summarize situation
  - And solution

2. Environments

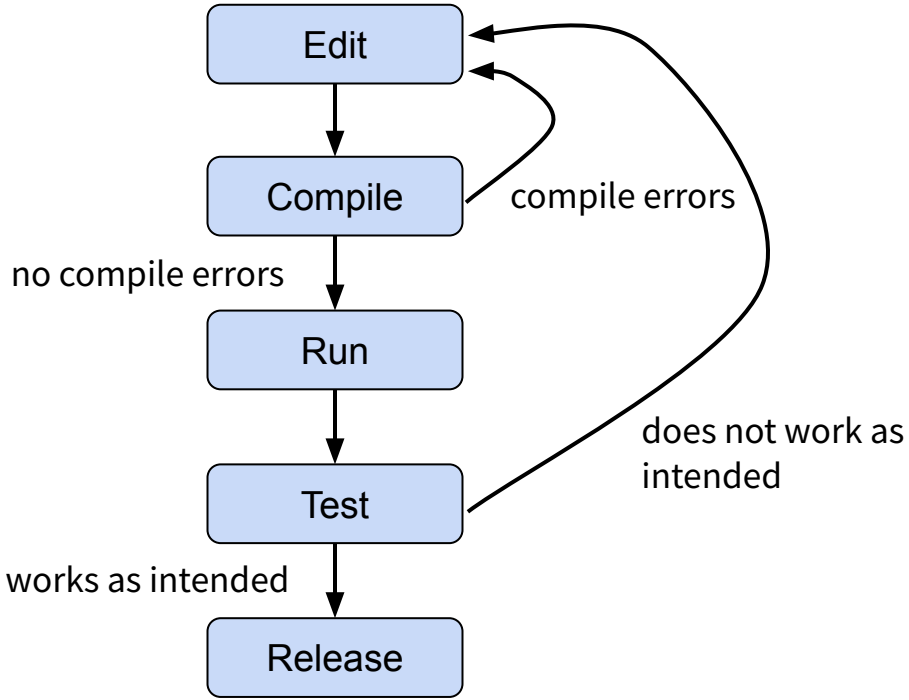
Development Environments

- *Development Environment:* suite of software tools for programming
  - Edit, compile, run, test, debug, release
- *Integrated Development Environment (IDE)*
  - Microsoft Visual Studio, Apple Xcode, Eclipse, …
  - Graphical native app for all tasks
  - Intimidating, confusing for beginners
- *Command-Line Interface (CLI)*
  - Separate shell command for each task
  - Old-school (nothing wrong with that)
  - Learn in pieces
  - Exposes what's happening
  - What we are doing

Consumer versus professional workspaces

Consumer experience	Professional workspace
movie theater	movie set
restaurant dining room	commercial kitchen
music album	recording studio
(polished experience of final product)	(productive, safe, creative workshop)
Windows, macOS, Android, iOS, XBox, ...	Linux, Xcode, Visual Studio
inadequate for creation	supports creation

Review: The Edit-Compile-Run Cycle



Keyboard-First Principle

- Humans can type faster than they can click
- Excessive mouse moving causes Repetitive Stress Injury
  - RSI, Carpal Tunnel Syndrome
- **Keyboard-First Principle**: using keyboard is better than mouse

Mise-en-Place Principle

- Mise-en-Place: putting tools, components in place for ergonomics

Programming Mise-en-Place

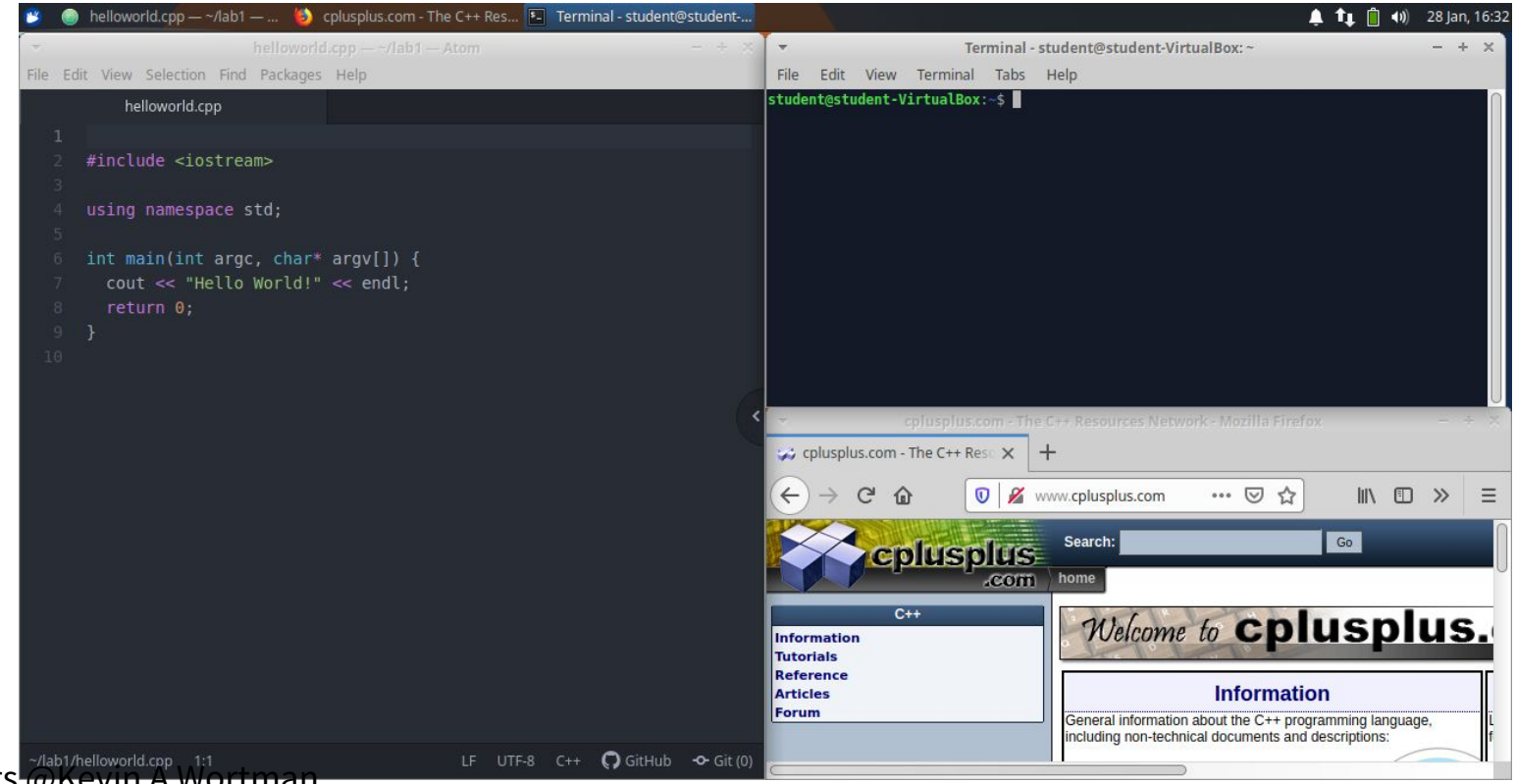
Gotta-haves while programming:

1. Editor (VS Code)
2. Shell
3. Documentation (browser, lab instructions, cppreference.com, etc.)

Best Practice:

- (keyboard-first, mise-en-place)
- Arrange windows so you can see 1, 2, 3 at the same time
- ALT-TAB to switch between the windows (don't click)

Window Mise-en-Place



Unix, Linux, Ubuntu

- **Unix**: widely-used framework for operating systems
  - All modern platforms except Microsoft
  - macOS, iOS, Linux, Android, Chrome OS, PlayStation, cloud servers, ...
  - WSL: Unix inside Windows
- **Linux**: a popular, free, version of Unix
  - Created by Linus Torvalds
  - Rhymes with his Finnish name: “linnukks”
- **Ubuntu**: Linux distribution (version)
  - Popular
  - Good installation support
  - What we are using

Files, Text Files, Editors

- **Text file**: a file that contains human-readable text
- Types of text files
  - .txt: text for human consumption, e.g. LICENSE.txt
  - .cc: C++ source code
  - .md: Markdown, text for human consumption with formatting, e.g. README.md
- **Text editor** (aka “editor”): program for opening, editing, saving text files
  - Core programmer’s tool
- We use an editor called **VS Code**

Shell and Terminal

- **Shell**: a special Unix program that allows a user (you) to run and interact with other programs
- **Terminal**: a thing that lets you see shell input/output
  - Physical terminal: monitor, keyboard, connection to real computer
  - Terminal emulator: program that simulates a physical terminal
- **Prompt**: when the shell is waiting for a command,
  It prints a “prompt” ending in $ (dollar sign)
- You type a **command**, then the Enter key to run the command
- Unix programs are **concise**: if everything worked, there is no stdout output

Running a Unix Program

program name: “git”

Commandline arguments:
“clone” “https://github..”.

stdout

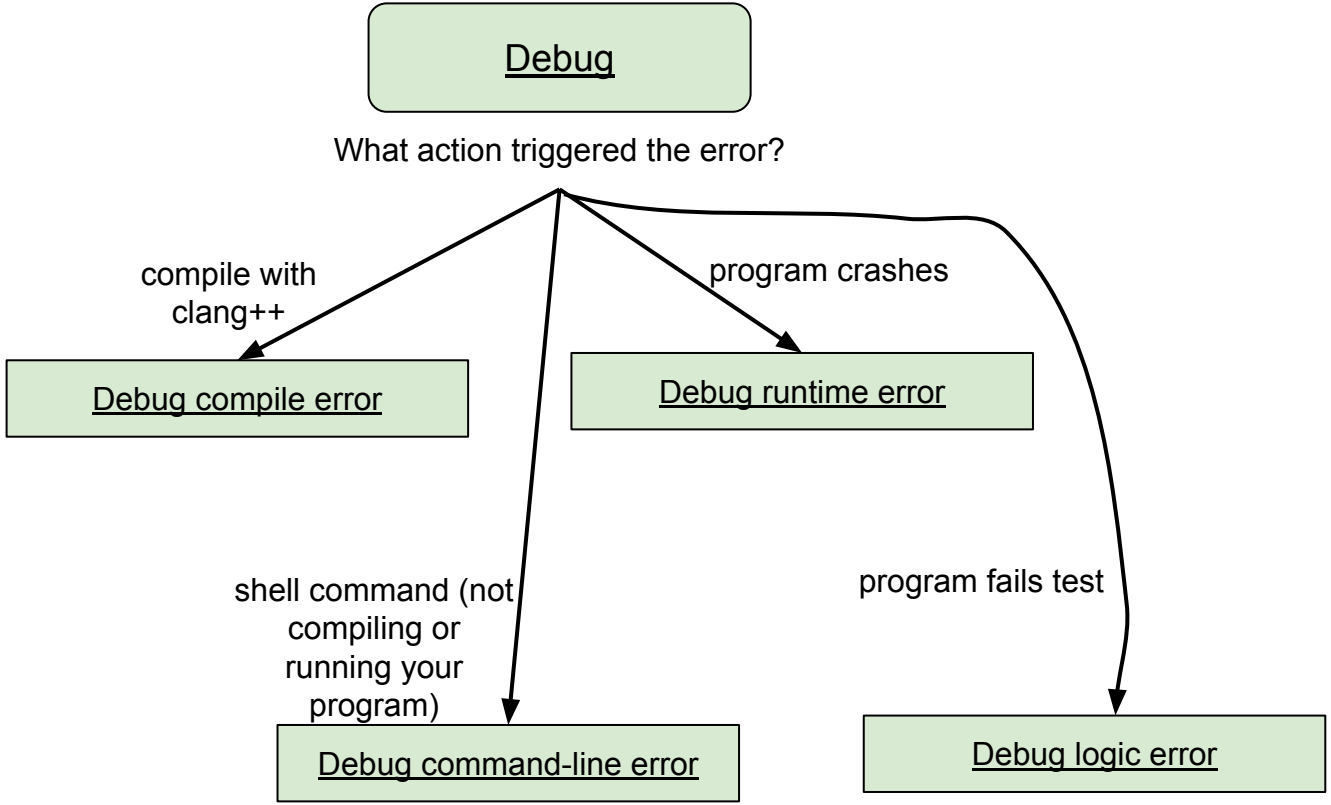
Terminal - student@student-Virtu ~

File Edit View Terminal Tab Help

```
student@student-VirtualBox:~$ git clone https://github.com/githubtraining/hellogitworld.git
Cloning into 'hellogitworld'...
remote: Enumerating objects: 306, done.
remote: Total 306 (delta 0), reused 0 (delta 0), pack-reused 306
Receiving objects: 100% (306/306), 95.57 KiB | 434.00 KiB/s, done.
Resolving deltas: 100% (70/70), done.
student@student-VirtualBox:~$
```

3. Flowcharts

Example: Debug Flowchart



Purpose of flowcharts

- Problem-solving process for intro. CS
- Use when “stuck”
- **Always a way forward**
- **Explicit procedure** in flowcharts
- Goal:
  - internalize processes
  - don’t need visual aid
- For now:
  - follow flowcharts exactly

Using flowcharts

1. Navigator's responsibility
2. Identify which flowchart you need
   a. Choose Tool
   b. Testing
   c. Debugging
   d. Reference
   e. Design
3. Start at top
4. Keep track of where you are
5. Follow arrows according to text along the arrow
6. Underlined word: jump to that flowchart

4. Pair Programming

Why Pair Programming?

According to research, pair programming improves...

- **Quality** of the work
- Amount of **time** taken
- **Enjoyment** of the process
- **Collaboration** and **communication** skills
- **Peer networks**
- **Retention**: number of students who pass course, remain in major

Forming Pairs

- Pairings are created randomly
  - Supports goals on previous slide
  - NCSU study: 93% satisfaction w/ random partners
- For 2 weeks
- If your partner is absent
  - We form new temporary pairs
  - Leftover student: temporary group of 3
- To keep working outside lab class
  - Need to schedule yourself
  - Be flexible and professional
  - ECS Open Lab, room CS-200: http://www.fullerton.edu/ecs/cs/resources/labs.php

Grading

- Make one submission (one GitHub repo) per pair
- Both partners will get the same grade
- Later: confidential survey on your partner’s cooperation
- Participation is a part of your lab grade

Roles

- Pair shares one PC
- *Driver:* controls keyboard and mouse
- *Navigator:* observes, asks questions, suggests solutions, longer-term strategies, tracks flowchart
  - Ex. “remember to save before compiling”
  - Group of 3: two navigators
- Switch every **30 minutes**: TA’s phone timer or verbal announcement

Dealing with Differences

- Expect mismatch of preparation, hard skills, soft skills
- Partner not participating properly:
  - First bring it up directly to them
  - Can ask TA/ILA for help/clarification during lab
  - Can contact TA/instructor outside of class

Conclusion

- (Need sign-in sheet back 😁)
- Reminders:
  - Labs meet this week
  - Note questions to ask next lecture
  - GitHub Login
  - Setup notebook
  - (Other homework in Canvas)
  - New reading assignment and quizzes due next Sunday