18. File I/O, Live Code

CPSC 120: Introduction to Programming
Pratishtha Soni ~ CSU Fullerton

Agenda

0. Announce
    a. Sign-in sheet
    b. Midterm 2: Wed Nov 1 (week 11)
1. Technical Q&A
2. File I/O
3. Live Code

1. Technical Q&A

Technical Q&A

Let’s hear your noted questions about...

- This week’s Lab
- Linux
- Any other technical issues

Reminder: write these questions in your notebook during lab

2. File I/O

Review: Filesystem

- Unix organizes storage into a **filesystem**
- A **file** holds data and has a **filename** (e.g. README.txt)
- A **directory** holds files or other directories
  - *Family tree* analogy: the "**parent"** directory holds "**child"** files/directories
- The **root** directory, written / (forward-slash), is the parent of everything else
- A **path** is the location of a file
- **Absolute path**: directions starting from /, with / separating each directory/file name
  - Ex: /usr/share/dict/words
  - The initial / means "start from the root"

File I/O

- **I/O**: Input/Output
- So far: standard I/O
    - `cin`, `cout`
- **File I/O**:
    - `ifstream`: input from a file
    - `ofstream`: output to a file
- Similar to standard I/O
    - `<<`, `>>`
- Output is simpler
    - Less can go wrong
    - Will discuss output first

Uses of File I/O

- INPUT other than command-line arguments, standard input
- Development tools: clang++, make, git
- **Data science**: read dataset with business logic data
- Save/open
  - Program saves information to file
  - Loads file next time it runs

ofstream

- ofstream: **O**utput **F**ile **Stream**
- put data **into** file
- in header `<fstream>`
    - `#include <fstream>`
- ofstream::ofstream (constructor): open file named by string
- ofstream::operator<<: write to file
- Converts to bool
    - `true` == no errors
    - `false` == errors

Example: File Output

```
// save game
int x_coord{1}, y_coord{2}, score{1000};
std::cout << "You are at (" << x_coord << ", " << y_coord
          << "), score=" << score << "\n";
std::ofstream file{"game.dat"};
file << x_coord << " " << y_coord << " " << score << "\n";
if (!file) {
    std::cout << "I/O error writing game.dat\n";
    return 1;
}
```

Standard output:

```
You are at (1, 2), score=1000
```

Contents of game.dat:

```
1 2 1000
```

I/O Errors

- **I/O error**: an I/O operation failed
    - `open`, `<<`, `>>`
- We have seen
    - `cin::>>` fails on invalid input
- Additional reasons for I/O errors with files
    - file not found (wrong name)
    - disk full
    - hardware failure (broken)
- Best practice: **file I/O code must handle I/O errors**
    - if statement to decide whether file object is `true`

ifstream

- ifstream: **I**nput **F**ile **Stream**
- pull data **out of** file
- in header `<fstream>`
    - `#include <fstream>`
- ifstream::ifstream (constructor): open file named by string
- ifstream::operator>>: read from file
- Converts to bool
    - `true` == no errors
    - `false` == errors

Example: File Input

```cpp
// load game
int x_coord{0}, y_coord{0}, score{0};
std::ifstream file{"game.dat"};
file >> x_coord >> y_coord >> score;
if (!file) {
    std::cout << "I/O error reading game.dat\n";
    return 1;
}
std::cout << "You are at (" << x_coord << ", " << y_coord
          << "), score=" << score << "\n";
```

Output when game.dat does not exist:

```
I/O error reading game.dat
```

Contents of game.dat:

```
1 2 1000
```

Output when game.dat exists:

```
You are at (1, 2), score=1000
```

Review: Current Directory

- **current directory** = location where a program "is"
  - a.k.a. **working directory**
- *State:* current configuration, subject to change
- Keep current directory in mind
  - Unlike search-based apps
- pwd command: **p**rint **w**orking **d**irectory

Program Working Directory

- **program's working directory** = working directory of shell command that started program
    - Rule varies by operating system
    - This is the rule for Unix/Ubuntu
- Working directory is not necessarily the same as where the program is stored
- Example: `git` is in /usr/bin/git, but we run it from other directories
- Could be same, ex. `$ ./a.out`
- Could be different, ex. `$ part-1/a.out`

Pitfall: Wrong Directory

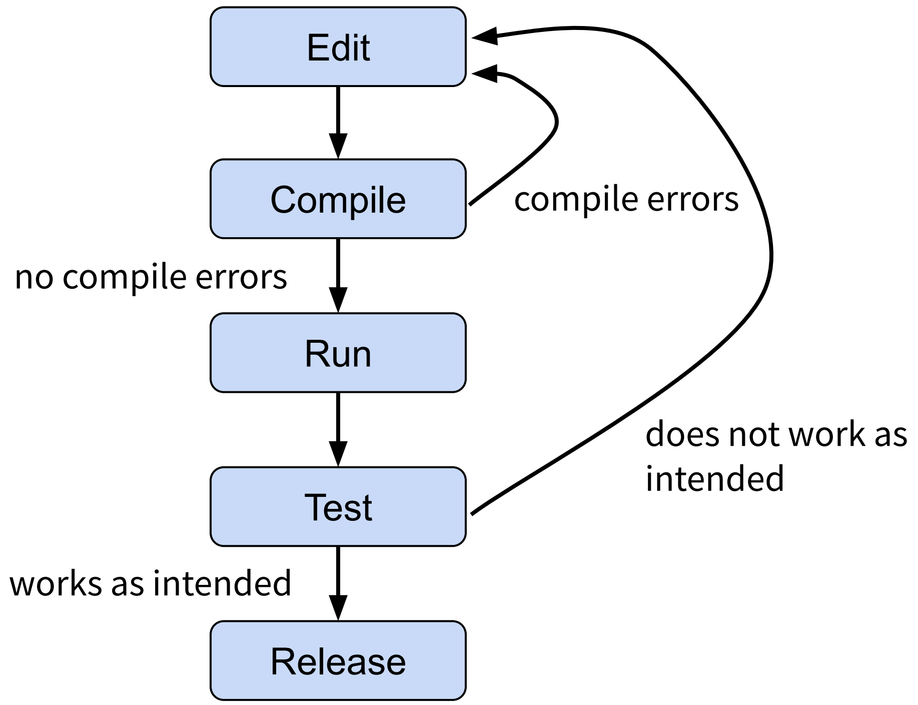
- Runtime error:
  - Input file exists, but program fails to open it
  - Program writes output file, but it doesn’t exist
- Cause: program’s working directory is different than you think
- Review: **program’s working directory** = working directory of shell command that started program
- To debug: make sure you are running program from .
  - (current directory)

3. Live Code

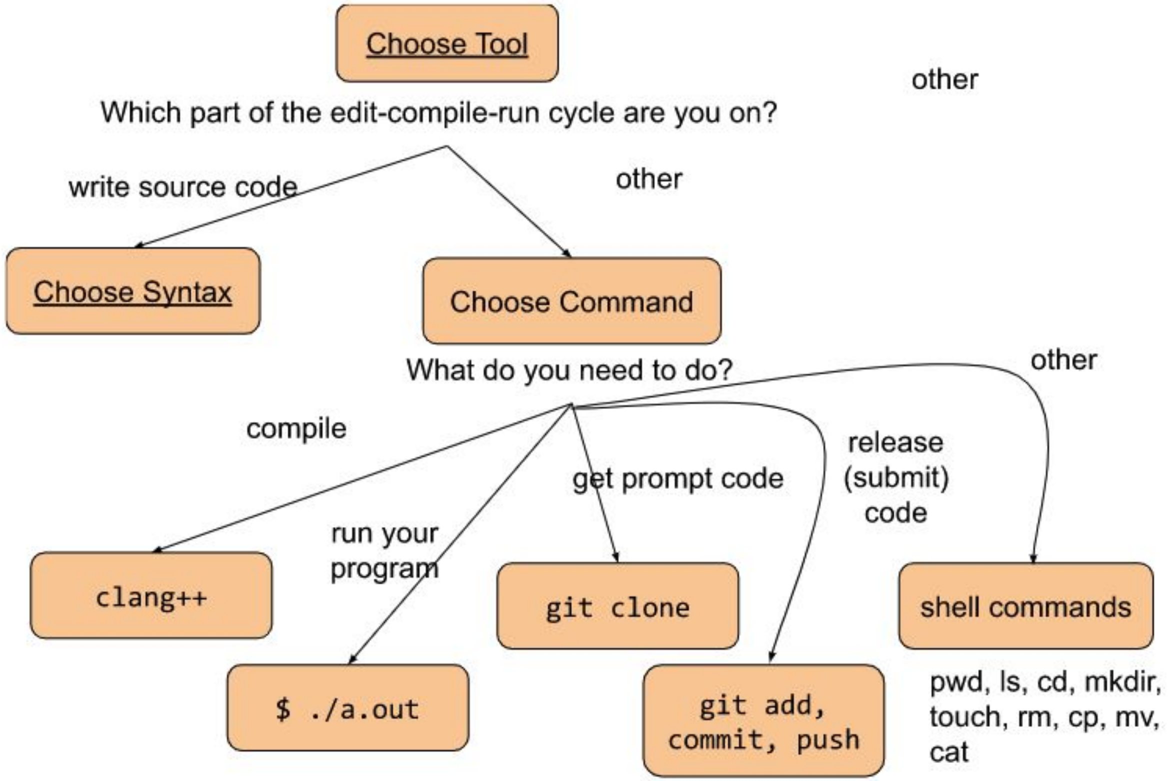
Live Code

- Interactive
- Instructor: **driver**
- Students: **navigators**

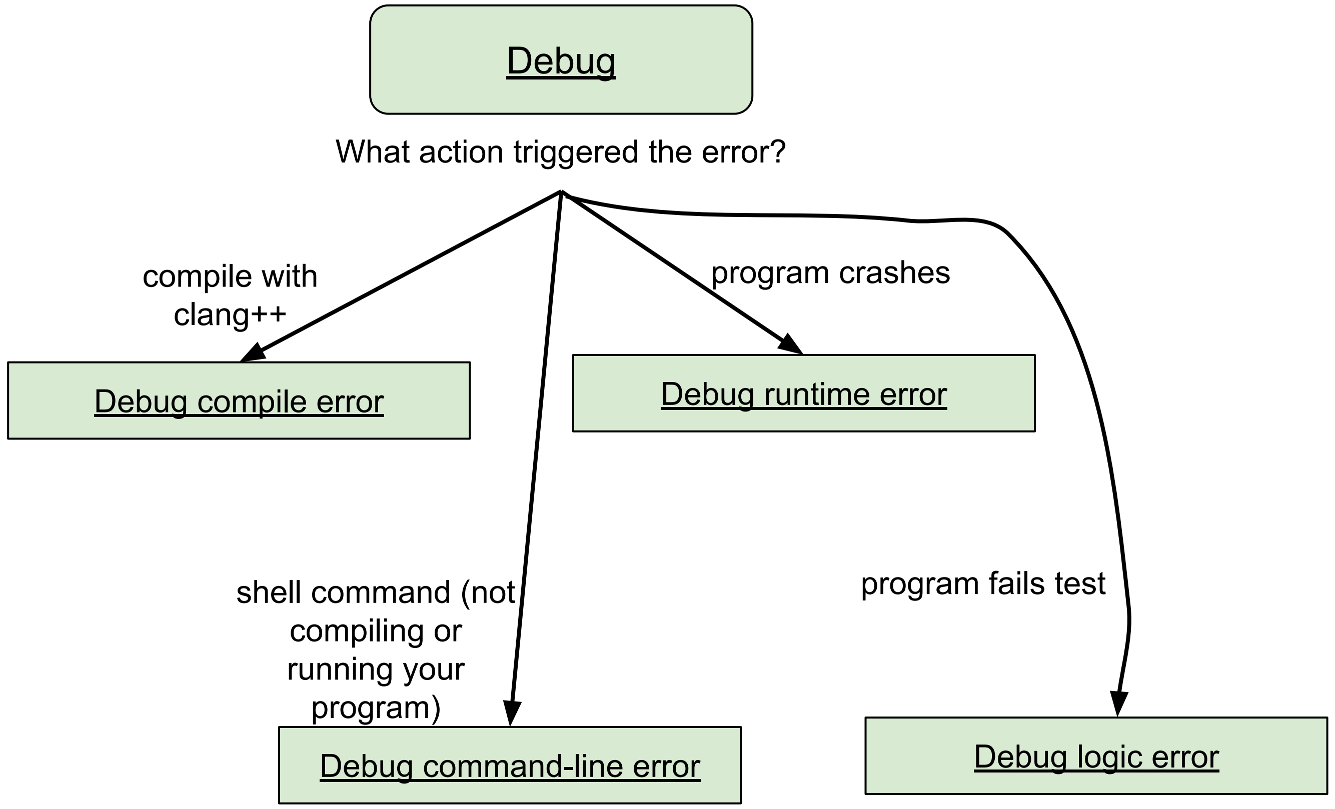
Review: The Edit-Compile-Run Cycle



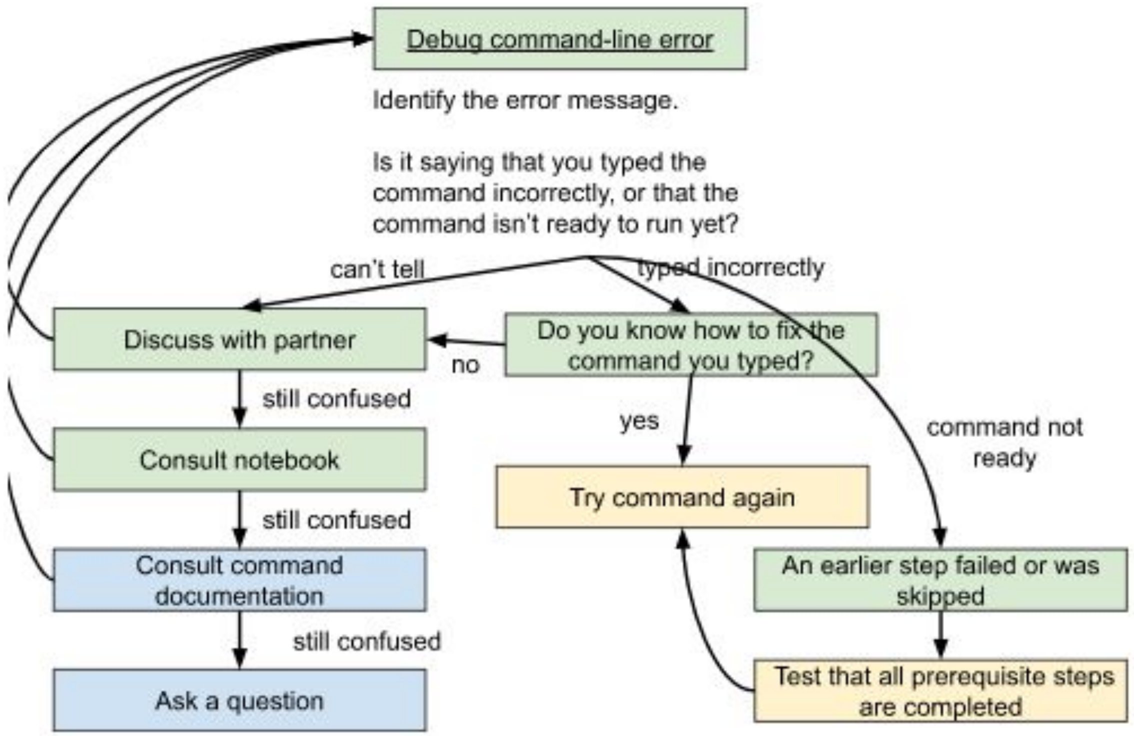
Review: Choose Tool Flowchart



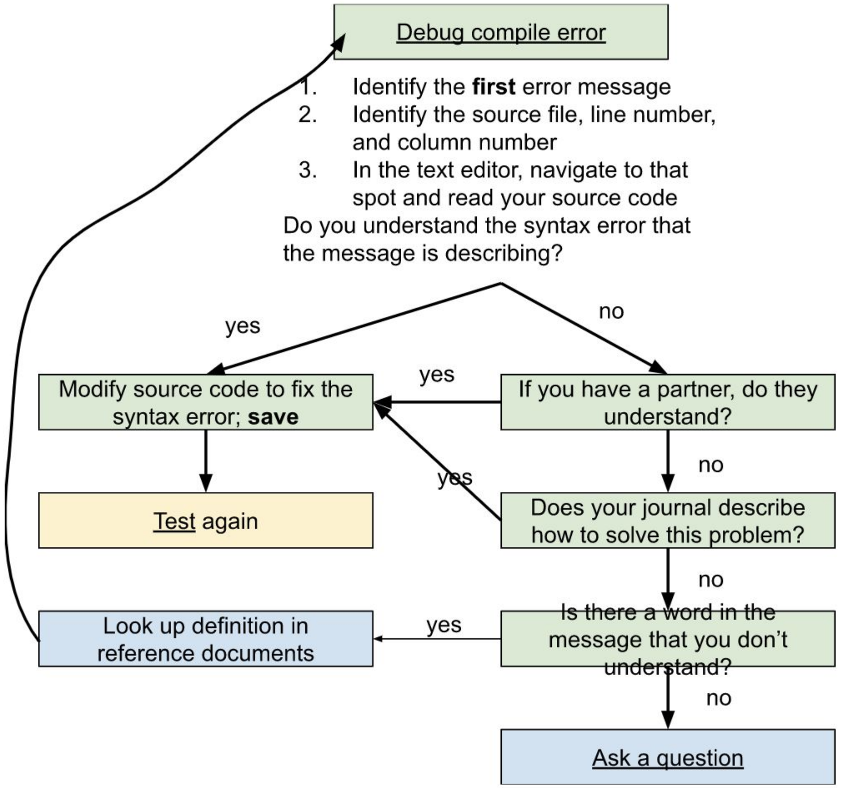
Review: Debug Flowchart



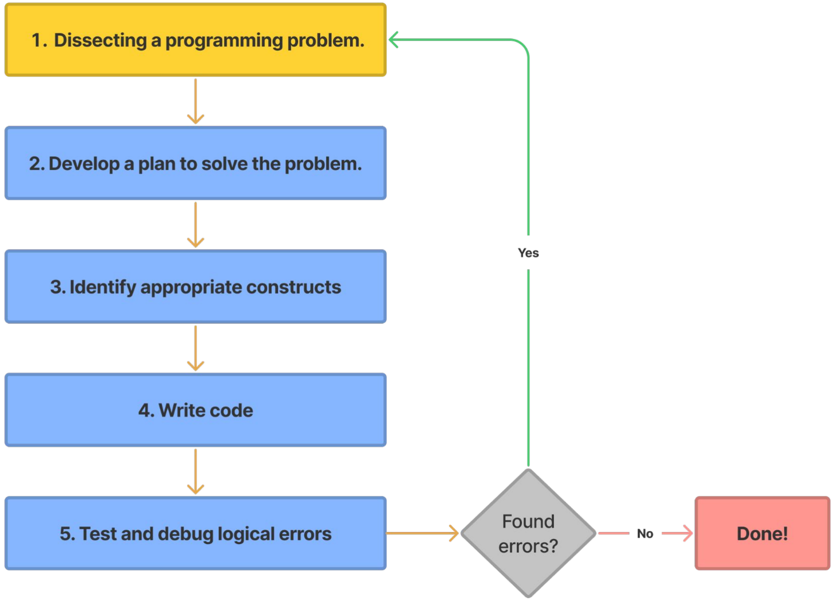
Review: Debug command error flowchart


Debug command-line error
Identify the error message.
Is it saying that you typed the command incorrectly, or that the command isn't ready to run yet?
can't tell
typed incorrectly
Discuss with partner
no
Do you know how to fix the command you typed?
still confused
yes
command not ready
Consult notebook
Try command again
still confused
Consult command documentation
An earlier step failed or was skipped
still confused
Ask a question
Test that all prerequisite steps are completed

Review: Debug compile error flowchart


Debug compile error
1. Identify the **first** error message
2. Identify the source file, line number, and column number
3. In the text editor, navigate to that spot and read your source code
Do you understand the syntax error that the message is describing?
yes
no
Modify source code to fix the syntax error; **save**
yes
If you have a partner, do they understand?
no
yes
Test again
Does your journal describe how to solve this problem?
no
Look up definition in reference documents
yes
Is there a word in the message that you don't understand?
no
Ask a question

Steps for Solving a Programming Problem


1. Dissecting a programming problem.
2. Develop a plan to solve the problem.
3. Identify appropriate constructs
4. Write code
5. Test and debug logical errors
Found errors?
Yes
No
Done!

Demo: File Output

- Run program that writes to file
- Observe
  - Regular file in directory
  - Can open file in VS Code
  - Contains data that was written

Demo: File Input and Working Directories

- Opening...
  - succeeds from same directory as file
  - fails from other directory

Prompt

- Password file
- `password` program
  - prompt for password
  - write into `password.dat`
- `login` program
  - prompt for password
  - compare to `password.dat`
  - print out success/failure