CSE 374: Programming Concepts and Tools

Eric Mullen
Spring 2017
Lecture 16: make

Administrivia

- HW5 out later today (or tomorrow)
- If you haven't got your midterm back, email me or come to my office hours (Tues 11am)
- Partners for HW6 due in one week (Wed May 10, at midnight)
    - Will be simple web survey, just fill in required info for both partners, will be up shortly

Where we are

- You know C fundamentals now (all you need is practice)
- For the rest of the course, we're moving on to tools, a little about malloc, C++, and a touch of undefined behavior (super exciting!)

Compilation

- Remember the midterm: “compile a file only if the source is newer than the executable”
  - `gcc -Wall -std=c11 -g prog.c -o prog` every time can get tedious
- Turns out something similar is extremely useful in practice
- Why don’t we just write a small shell script for each project?

Dependencies

- It turns out that dependency management is hard
- Even once we know all dependencies, calculating what to compile is not trivial
- Thus we have a tool to help us out, called `make`

make

- Simple tool for:
  1. running commands
  2. using explicit dependencies to only run necessary commands

Makefile

- `make` requires directions: commands and dependencies

- we express these as rules in a file named `Makefile`

```
<target>: <dep1> <dep2>
    TAB <command to run>
```

```
prog.o: prog.c prog.h
    TAB gcc -c prog.c -o prog.o -Wall -std=c11 -g
```

special targets

- `all`: define a target called all, give it dependencies of all finished products (i.e. executables). Put it first in the file. When `make` invoked, first target will be built.

- `clean`: define a target called clean, when `make clean` invoked removes all generated files (e.g. executables and .o files)

How to run

- must be in same directory as Makefile
  - alternatively use any file as the Makefile with -f option
- run `make <targetname>` to make a particular target
- run `make` to build first target in file

Demo

- Remember last lecture?

- Let’s make a Makefile for the duo project

Data Structures in C

```
typedef struct charlist {
   char data;
   struct charlist* next;
 } charlist;
```

```
typedef struct floattree {
   float data;
   struct floattree* children[2];
} floattree;
```

Data Structures in C

- Usually put data structures on the heap (use malloc to create)

- Wrap data structure in accessor functions, declare them in a header, implement in separate file

- Sometimes even have public and private header, only expose data representation in private header

Data Structures in C

- Memory management is key: usually provide a "release" method to free all memory used by structure

- If data contained is a pointer to something else, think carefully about who owns the data when

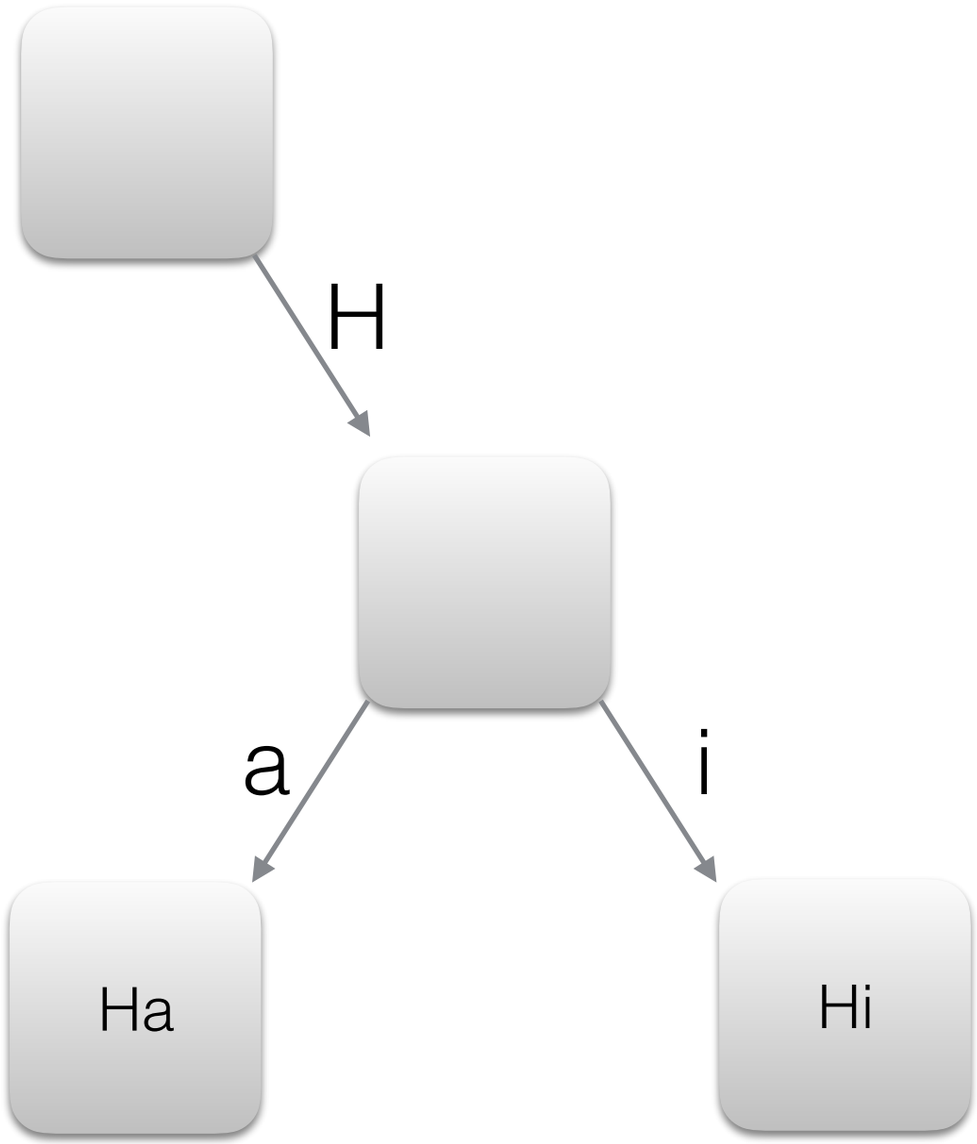
Trie

- A trie is similar to a binary tree, but with more children

- Each node represents a single character in a string, and each node has a number of children possible equal to the number of letters in the alphabet

- Tries are extremely good for looking up strings

Trie



Contains “Hi” and “Ha”

Does not contain “” or “H”

Homework Demo

- I'll show you how your HW5 program should work
- writeup out later today or tomorrow