

CSC295 2014S, Class 01: Introduction

Overview

- Preliminaries.
 - Admin.
- About the course.
- Some principles and practices.
- Example: C from K and R.
- Detour: A C memory problem.
- Exercises: Some simple tasks.
- Homework.

Preliminaries

Admin

- Welcome to "Thinking in C and *nix" (or whatever the course is called in the catalog).
 - In the future, it will be CSC 282, Thinking in C and *nix, so I think of it that way.
- I'll take attendance, mostly to help me make sure that I know everyone's name. (I have 120 students this semester, so knowing everyone is hard.)
- Feel free to interrupt me at any time with questions and comments.
- I tend to overplan, so we won't do everything in the outline.

About the course

- Your path to competency in C and *nix.
- 14 classes, Sam here for 12.
- Weekly homework.
- Mixture of lab, discussion, recitation, lecture.
- On the Web
- "Textbook" under development.

Some principles and practices

Three primary approaches

- Good programming practices
- Ways of thinking like a C programmer
- Ways of thinking like a *nix programmer

Good programming practices

- Good programmers document. For clients. For themselves.
- Good programmers follow a formatting standard. We'll use GNU C.
- Good programmers debug with a debugger, not with printf statements (although the occasional printf is useful)
- Good programmers use a source code management system (and use it well)
 - Don't pollute the repo.
 - Don't commit non-working code to the main branch.
- Good programmers test their code (usually with unit tests).
- Good programmers refactor rather than cut/paste/change.

Good C programmers

- Know how to write macros, and how to use them in clever ways
 - And other aspects of the C preprocessor
- Know how to use a debugger, probably gdb. \$ gdb program

run CRASH bt

- Know how to compile with libraries
 - And to use compiler flags
- Understand memory and pointers
- Segment large programs into individual files
- Use Make

Thinking like a *nix programmer

- Do things by code rather than by hand
- Write small programs that they can tie together with other programs.
- Use text files for data and communication.
- Share

Example: C from K and R

```
char *  
fun (char *t, char *s)  
{  
    while (*t++ = *s++);  
    return t;  
} // fun
```

This function copies s to to. It's mostly strcpy. It returns a pointer to the one after the end-of-string character in the result.

Detour: A C memory problem

```
x = malloc (...);
foo ();
bar ();
free (x);    // CRASH
```

- Program crashes with the call to free.
- Program does not crash without the call to free (but leaks memory)
- foo and bar use x, but have no calls to free
- Neither foo nor bar reassigns x, although they do include x[i] =
- What might be going wrong?

Exercises: Some simple tasks

Given a DOS-formatted text file (lines end with \r\n rather than just \n), convert it to a standard text file.

If you know awk, you could write an awk script to do it. (less than a minute, if you know awk and use it regularly)

If you know flex, you can write a flex program. (five minutes)

If you program regularly in C (which we'll assume, write a C program). (five to ten minutes).

```
int ch;
while ((ch = getch()) != EOF)
{
    if (ch != '\r')
        putchar (ch);
} // while
```

Two more tools:

- tr - character based translation (fifteen seconds) `tr -d '\r' < dosfile > usablefile`
- sed - stream editor (fifteen seconds)

Given a standard text file, convert all uppercase letters to lowercase.

- Find the Python command to do that (use a scripting language).
- Write a C program. (five minutes)
- Copy into LibreOffice, highlight, find appropriate menu item, copy it back, deal with the automatic translation of " to curly quotes, ... (if you're running LibreOffice and it doesn't do extra translation, less than a minute; if you're not running LibreOffice, +2 minutes)

Given a standard text file, remove all blank spaces at the end of lines.

Make a list of all misspelled words in a text file.

Given a CSV file in which each line has the form

`LastName,FirstName,Assignment,NumericGrade`

find the the five highest grades on HW2

Given an HTML file, find the URLs of all images. In case you don't

know HTML, those will typically look like `` * The `img` can have any capitalization (`img`, `IMG`, `Img`, `iMg`, etc.) * There can be other text between the `img` and the `src`. (That text cannot include a greater than sign.) * You may find it easier to start this problem by assuming that there's only one image tag on a line.

* But, in the end, you should support multiple image tags on the same line.

Detour: Using the Shell

- `!!` repeat the last command
- `!text` repeat the last command that starts with the given text
- `!123` repeats command 123
- `!$` the last thing on the last line

Some important *nix tools

Homework

This will appear online some time in the near future (probably over the weekend).

Repository: <https://github.com/Grinnell-CSC282/hw1>

1. Read chapter 1 of Raymond to get a deeper understanding of the *nix Philosophy.
2. Find good solutions to any of the tasks we did not solve in class. (That is, write a program or command to solve the task.) Put them in a folder with your name in the GitHub repository.
3. Make a list of *nix tools and utilities that you've found useful (or even that you've just heard of). Try to categorize and add a short description. Put them in the `tools.md` file in the repository.
4. Add some notes about what might be wrong with the C code above.

Copyright (c) 2013 Samuel A. Rebelsky.



This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/3.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.