CMPUT 201: Practical Programming Methodology

Guohui Lin
guohui@ualberta.ca
Department of Computing Science
University of Alberta
September 2019

Lecture 1: Course Outline

Agenda:

- Course calendar description
- Course objectives
- Student responsibilities
- Course official information
  - introducing the team
  - my office hours
  - weekly plan
  - mark distribution
  - course policies
  - final grade
- Introducing C

Reading:

- eClass: https://eclass.srv.ualberta.ca/course/view.php?id=54569
- Public webpage: https://www.cs.ualberta.ca/~ghlin/cmput201.php
- Textbook: Chapter 1

Major take-home messages:

- Computing science is not programming, but program design
- C language provides a set of tools
- Lectures on good practices of an intermediate programmer
  - "clear and simple" preferred over "clever and complex"
- Students use C to program under the Linux environment
  - reading (textbook, sample codes, sample solutions, etc.)
  - understand the collaboration policies (5%, due November 29, 2019)
    - ✓ Honesty-Trust-Fairness-Respect-Responsibility
    - × plagiarism, cheating, misrepresentation of facts, participation in an offense
  - work on lab exercises (30%, starting Monday September 9, 2019)
  - work on assignments (21%)
  - be prepared for exams (14% + 30%)
- How to succeed:
  - "try it out"
  - search out: `man`, Google, discuss
  - use debugger (`gdb`, `valgrind`)
  - start your work AEAP!

Introducing C:

- History of C
  - a byproduct of UNIX operating system
  - a higher-level language than assembly
  - mostly done in 1960s and 1970s
  - standard C99
- C influences many modern programming languages
  - C++, Java, C#, Perl
- Strengths: efficiency, portability, power, flexibility, standard libraries
  - low-level, access to machine-level concepts
  - small, limited features
  - permissive, you know what you are doing
  - good for "writing a paper"
- Weaknesses: error-prone, difficult to understand, difficult to modify
  - not good for "writing a book"
- Effective use of C — stick to standard !

Regarding Assignment #1:

- All three assignments together as a project
- Assignment #2 is built on Assignment #1
- Continuingly, Assignment #3 is built on Assignments #1 & #2
- Specifications for Assignment #1:
  1. read in an instance, redirect, and validate
  2. print out an instance, redirect
  3. implement a dynamic programming algorithm
- Purchase a solution program?
  - each costs you 2 marks (no matter you use the purchased program or not)

    [e.g., if your program gets x marks, then the actual is $x-2$, which can be negative]
  - cannot be re-distributed (cheating penalty applies)
  - need an email or written paper (to your TA) to confirm you understand all these

Agenda:

- CSC 159: `ugXX.cs.ualberta.ca`
- The Linux environment, shell commands
  - demo remote login
- Editing: `vi`
- Example C programs
  - `celsius.c`
  - `fibonacci.c`
- Compilation
  - we use the standard options: `gcc -Wall -std=c99 celsius.c`
- Running

Reading:

- Textbook: Chapter 2

The UNIX/Linux shell:

- The machines in the CMPUT 201 lab are
  - `ugXX.cs.ualberta.ca`, where `XX` ranges from `00` to `34`
  - e.g., `ug10.cs.ualberta.ca`

```
login as: ghlin
ghlin@ug10.cs.ualberta.ca's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-55-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo

ghlin@ug10:~>
```

- The shell prompts for commands to execute with "`ghlin@ug10`"
  - `ls`, `mkdir`, `cd`, `cat`, `vi`, `gcc`, `cp`, `rm`, `mv`, ...
  - `man` (manual pages are extremely useful)

The UNIX/Linux shell:

- For example,

```
ghlin@ug10:~>man ls
```

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
 Manual page ls(1) line 1 (press h for help or q to quit)
```

The UNIX/Linux shell:

- Use an editor such as `vi` (`vim`, `emacs`) to create a new C source file
  - if file exists, `vi` opens the file for editing
  - for example (Pages 24–25),

```
ghlin@ug10:~>vi celsius.c
```

```
/* Converts a Fahrenheit temperature to Celsius */

#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

int main(void) {

    float fahrenheit, celsius;

    printf("Enter Fahrenheit temperature: ");
    scanf("%f", &fahrenheit);

    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;

    printf("Celsius equivalent: %.1f\n", celsius);

    return 0;
}
```

The UNIX/Linux shell:

- Compile the C source file
  - we use "gcc", with various options
- Run the executable file

```
ghlin@ug10:~>gcc celsius.c -o celsius
ghlin@ug10:~>lc
total 4
-rwx------+ 1 ghlin prof 8613 Jan  3  2017 celsius*
-rw-------+ 1 ghlin prof  373 Jan  3  2017 celsius.c
ghlin@ug10:~>./celsius
Enter Fahrenheit temperature: 240
Celsius equivalent: 115.6
ghlin@ug10:~>./celsius
Enter Fahrenheit temperature: 40
Celsius equivalent: 4.4
```

General form of a C program:

```
/* directives */

int main(void) {

    /* statements */
}
```

- Typical `directives`,

```
/* headers */
#include <stdio.h>

/* macros */
#define FREEZING_PT 32.0f

/* global variables */
int num_pt = 0;

/* function prototypes */
void mst_prim(int n, int **point);

/* main function */
int main(void) {

    /* statements */
}
```

General form of a C program:

```
/* directives */

int main(void) {

    /* statements */
}
```

- Typical `statements`,

```
/* declarations */
float fahrenheit, celsius;

/* assignments */
celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;

/* function calls */
scanf("%f", &fahrenheit);

mst_prim(num_pt, point);

/* function terminates, and returns a value */
return 0;
```

```c
/* Converts a Fahrenheit temperature to Celsius */

/* directives */
/* headers */
#include <stdio.h>

/* macros */
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

/* main function */
int main(void) {

    /* statements */
    /* declarations */
    float fahrenheit, celsius;

    /* function calls */
    printf("Enter Fahrenheit temperature: ");
    scanf("%f", &fahrenheit);

    /* assignments */
    celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;

    printf("Celsius equivalent: %.1f\n", celsius);

    /* function terminates, and returns a value */
    return 0;
}
```

General form of a C program:

- As a whole,
  - use of `/* ... */` to enclose comments
  - explaining *semantics* — what you do!

```
/*******************************************/
/* Converts a Fahrenheit temperature to Celsius */
/* Name: celsius.c                         */
/* Author: K. N. King                      */
/* September 11, 2017 **********************/
```

- Not only for correctness, also easy for eye to separate
  - use of "space" to separate "tokens" (variables, constants, operators, etc)
  - use of "indentation" to construct "blocks"
  - use of "blank lines" to separate logical blocks
  - **stick to standards !**

Fundamentals:

- C's standard I/O (`<stdio.h>`):
  - stdin — keyboard / terminal screen
  - stdout — terminal screen
  - stderr — terminal screen
- `/* ... */` encloses comments
- `main(void)` function starts the program
- `return 0;` the program terminates and returns the value 0 (*status code*)
- Every variable **must** have a type (the kind of data to be held, size, operations, etc.)
- `gcc -o` specifies the executable name, otherwise "`a.out`"
  - we use "`gcc -Wall -std=c99`"

Fundamentals:

- Use "=" for assignments
  - e.g., `celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR`
  - different from "defining names for constants" (macros)
  - e.g., `#define FREEZING_PT 32.0f`
- Reading an input vs. printing an output — use of "`&`" (memory address)

```
float fahrenheit, celsius;

scanf("%f", &fahrenheit);
printf("Celsius equivalent: %.1f\n", celsius);
```

- Identifiers — names for variables, functions, macros, etc.
  - quite flexible in general (name what it is for)
  - must start with a letter or underscore
  - case sensitive
  - some keywords are reserved (such as "`union`", see Table 2.1 in Page 26)
  - some key prefixes are reserved (such as "`_X...`")
- **Don't** abuse/mis-use any punctuation marks !!!

Agenda:

- `printf()`
  - two most frequently used functions
  - need "`#include <stdio.h>`"
  - `printf(format string, expr1, expr2, ...);`
  - conversion specifications
- `scanf()`
  - most powerful in reading numbers
  - pattern-matching ability
  - inappropriate character push back to the input
  - has a return value
- How integers and floating-point numbers are stored

Reading:

- Textbook: Chapter 3