CMSC 16200: Honors Introduction to Computer Science II

Lab Session 1 **Original Author: Yongshan Ding**

1 Introduction

Welcome to the first lab session of CMSC 16200! Today, we are going to help you set up the tools needed for this course, get started with our first C program, and review some of the material covered in the lecture.

2 Announcements

- The first assignment (Lab 0 - Can You C It) has been released.

3 Logistics

In general, lab session will be a place for a quick review of previous lectures, some supplementary material (if needed), some practice problems, and discussions of lab assignments. Collaboration is encouraged during lab sessions. To succeed in this course, you may want to take advantage of the following resources.

- Lectures. Coming to lectures is not enough, you must attend and engage with the course. Interacting with the course staff is a great way to learn.
- Practice. This includes attending lab sessions and working on the lab assignments.
- Get help. Please don't hesitate to ask questions. Note that Piazza and office hours are all great places to raise your questions. The course staff will try their best to explain and clarify things.
- Check updates. We will use the course website and Piazza to make announcements, please check them regularly.

4 Checklist

This is a checklist for getting your system set up for this course.

◯ Course website. `https://sites.google.com/view/cs162-winter-2020/home`

◯ Piazza signup link: `https://piazza.com/uchicago/winter2020/cs162`

◯ File editor. Choose your favorite editor, e.g. vim, emacs, or Sublime Text, etc.

◯ C compiler. `gcc` compiler included in most Linux distributions.

◯ LaTeX. Excellent reference: `https://en.wikibooks.org/wiki/LaTeX`

5 Obtaining handout code

After downloading the handout source files from the Piazza, extract the zipped file by running:

```
> > tar -xvf week1-handout.tgz
```

from a terminal window.

6 Hello, world!

Let's start by looking at a very simple program:

```
1  /*
2   * hello.c: First C Program for CMSC 16200
3   */
4
5  #include <stdio.h>
6
7  int main(void) {
8      printf("Hello, world!\n");
9      return 0;
10 }
```

Despite very simple, the `hello.c` file contains most of the important features of a C program:

- Line 1-3: Multi-line comment, bounded by `/*` and `*/`. A single-line comment starts with `//`.
- Line 5-6: The `#include` macro, importing the header files that contains libraries and other declarations of constants and functions.
- Line 8: Declaration of main entry point of C program, including the function argument and return types.
- Line 9: Print out string to console output (i.e. stdout).
- Line 10: Return and exit program. 0 for "success".

7 Compiling program

In your terminal console, invoke `gcc` with the following flags:

```
> > gcc -Wall -Wextra -Werror -std=c99 hello.c -o hello
```

to compile your program. `-Wall` and `-Wextra` enables all compiler warnings; `-Werror` will treat all warnings as errors; `-std=c99` applies the C99 standard; `-o hello` means the output executable is named `hello`.

Alternatively, you can also use a compilation tool called `make`. Check out the `Makefile` in the handout folder:

```
1 helloworld: hello.c
2         gcc -Wall -Wextra -Werror -std=c99 hello.c -o hello
```

where first line contains the target (`helloworld`) and dependencies (`hello.c`), and the second line, which starts with a TAB, is the compilation line. Now to compile, simply type

```
> > make helloworld
```

8 Running program

Now type

```
> > ./hello
```

to execute your program.

9 Basic C Syntax

- **Semicolons:** Most line statements are ended with semicolons. Exceptions are function definitions, use statements, conditional statements, and loop statements.
- **Data types:** Must declare variables before use and the data types cannot be changed once declared.

  Task 9.1 (0 pts). Determine the size (in bytes) and range of the following data types on your machine, by writing a program with `#include <limits.h>` and `#include <float.h>` and the `sizeof()` operator. Note that 8 bits make up 1 byte.

  - `char`:
  - `short`:
  - `int`:
  - `unsigned int`:
  - `long`:
  - `float`:
  - `double`:

- **Conditional statements:** if-statement example code:

```
1 if (/* condition1 */) {
2       /* code if condition1 == true */
3 } else if (/* condition2 */) {
4       /* code if condition1 == false && condition2 == true*/
5 } else {
6       /* code if condition1 == false && condition2 == false*/
7 }
```

Logical AND operations (i.e. conjunction) are denoted as `x && y`, while logical OR operations (i.e. disjunction) are denoted as `x || y`. They are called *infix* operators. Some other arithmetic operations that give a boolean are `==`, `!=`, `<`, `>`, `<=`, `>=`, etc.

- **Loop statements:** while-loop example code:

```
1 int i = 0;
2 while (i < count) {
3     /* code */
4     i++;
5 }
```

for-loop example code:

```
1 for (int i = 0; i < count; i++) {
2     /* code */
3 }
```

Some useful keywords inside loop statements: `break` exits inner most loop. `continue` skips rest of the inner most loop body and jumps to next iteration.

main.c	magic.c	magic.h

`#include <stdio.h>`
`#include <stdlib.h>`
`#include "magic.h"`
 	
`int main(void) {`
`  int a = 123;`
`  int b = 0;`
`  printf("Before: %d\n", a);`
`  b = magic(a);`
`  printf("After : %d\n", b);`
`  return 0;`
`}`
	`#include "magic.h"`
	
	`/* My magic function */`
	`int magic(int a) {`
	`  int x = a;`
	`  a = 0;`
	`  while (x > 10) {`
	`  a += x % 10;`
	`  x /= 10;`
	`  }`
	`  a += x;`
	`  return a;`
	`}`
		`/* Header guard:`
		` * ensure declared only once.`
		` */`
		`#ifndef _MAGIC_H_`
		`#define _MAGIC_H_`
		
		`/* My magic function */`
		`void magic(int a);`
		
		`#endif`

Table 1: What does this code example do?

- **Preprocessor macros:** Lines that begin with `#`.
  - More on `#include` macro: Separates interface from implementation. Interfaces are contained in the `.h` header files, and implementations are contained in the `.c` source files. Table 1 contains a code example showing how to link the header files with the source files. For instance you can compile the example by typing:

    ```
    gcc -Wall -Wextra -Werror -std=c99 main.c magic.c -o magic
    ```

    Notice that all sources need to be listed.

- `#define` macro: can be used for inline function definitions. Note: use parentheses around expression to avoid problems after substitution.
- Conditional macros: `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif` will conditionally choose to compile a body of code.