FIT2100 Tutorial #1
Introduction to C Programming Basics
Week 1 Semester 2 2017

Dr Jojo Wong

Lecturer, Faculty of IT.

Email: `Jojo.Wong@monash.edu`

July 20, 2017

Revision Status

```
$Id: FIT2100-Tutorial-01.tex, Version 1.0 2017/07/20 12:00 Jojo $
```

Acknowledgement

The majority of the content presented in this tutorial was adapted from the courseware of FIT3042 prepared by Robyn McNamara.

Contents

1 Background

The programming knowledge of C is essential for implementing the various concepts of operating system that we are going learn in this unit. In this first tutorial, you will be introduced with the fundamental elements of the C programming language, as well as how to compile and execute C programs.

You should complete the following reading (Section 2) before attending the tutorial. You should also prepare the solutions for the practice tasks given in Section 3.

2 Pre-tutorial Reading

2.1 A Simple C Program

This C program simply displays a statement "`Welcome to C Programming!`" to the standard output device (i.e. the console).

```c
/* simple.c: a simple C program */

#include <stdio.h>

int main(void)
{
  printf("Welcome to C Programming!\n");
  return 0;
}
```

Each C program is defined with the **main** function since it is always the first function to be executed.

The `main` function often returns an integer value (`int`) indicating the status of its execution. Typically, a return value of `0` implies normal program termination (i.e. no errors); non-zero values indicate unusual or errorneous execution.

The `main` function often accepts two *arguments* or *parameters* (which we will see later in Section 2.8); however, no arguments are accepted in the `main` function given in the program above as indicated by the special type `void`.

Note: Each statement in C programs is terminated with a semicolon (`;`). Blocks of statements are enclosed in a pair of braces (`{}`).

How to create a C program? You can use any text editors of your choice to type in the source code. The source file should be saved with the `.c` extension, for example `simple.c`.

How to compile a C program? Before you are able to execute a C program, the program needs to be first compiled using the C compiler, such as `gcc`. To compile with `gcc`, one of the following commands can be used:

- `gcc simple.c`
- `gcc -o simple simple.c`

Note: The first command produces the *executable* file named `a.out` from the source file. However, the executable file can be re-named using the `-o` option; the second command sets the name of the executable file to `simple`.

How to run a C program? You can now run the C program with one of the following commands based on the name of the executable file:

- `./a.out`
- `./simple`

What is #include <stdio.h>? C provides a standard I/O library of functions for input and output. The library header `<stdio.h>` needs to be included and brought into the source file that intends to perform any input and output by using the directive in C — **#include**.

As for the `simple` program seen earlier, the `#include` statement is important as the compiler needs to know that the `printf` function takes a string as an argument.

2.2 Directives in C

Before the compilation, C programs are first edited by a preprocessor. Directives (always begin with the # symbol) are the commands intended for the preprocessor.

In additional to the `#include` statement that we have seen, *constants* in C programs can be defined with a specific value using the directive `#define`.

```
#define MAX_LENGTH 9
```

Any occurrence of the constant `MAX_LENGTH` within a program will be replaced by the corresponding value specified (such as 9 in the given example).

2.3 Variables in C

In C, variables must be *declared* before they are used. The common practice is to perform the variable declaration at the beginning of each function before any executable statements. (This is however not a rule that you must follow).

```c
int i=0, j=1, k=2;
float pi = 3.1415926553f;
double e = 1.0e-32;
char c = 'a';
char newline = '\n';
```

All the variables have a specific data type which is set at the compile time, and it cannot be changed — because C is statically typed. However, implicit and explicit type conversions can be performed on C variables.

```c
int i = 5;
float f = i;
int j;
j = (int) f;
```

Note: Variable names may contain letters, digits and underscores; but must not begin with a digit. Also note that variable names are case sensitive, thus `x` and `X` refer to two different variables.

2.4 Data Types in C

Data types in C are not guaranteed to be of a fixed size. The following are the primitive data types in C presented in the order of size.

- Integer: `char, short, int, long, long long`
- Floating point: `float, double, long double`

To determine the size of a particular type or variable, you can use the `sizeof` operator provided in C.

C does not have the built-in string type unlike other programming languages (such as Java and Python). A string in C is in fact an *array* of `char` terminated with the `null` character '\0'.

2.5 Arrays in C

C supports both single-dimensional and multi-dimensional arrays. Array subscripts in C always start at 0.

```
#define MAX_LENGTH 9

int a[10];     /* one-dimensional array of size 10 */
a[2] = 3;      /* initialise the third element */

char unit_code[MAX_LENGTH];    /* string with 9 chars */
char unit_name[] = "OS";       /* string of size 3 including '\0' */
```

```
#define ROW 10
#define COL 10

int a[ROW][COL];      /* two-dimensional array of 10x10 */
a[9][9] = 100;        /* initialise the last element*/
```

Note: Multi-dimensional arrays can also be achieved by using arrays of **pointers** to other arrays (which we will cover in the next tutorial).

```
char *argv[];         /* arrays of pointers*/
```

2.6 Enumerated Type in C

C supports another type of constant through *enumeration*. It is a convenient way to associate a list of constant values with names. The first name in an enumerated list is associated with the value of 0, the second name with 1, and so on; unless the values are explicitly specified.

```
enum boolean {NO, YES};    /* NO is 0, YES is 1 */

enum months {JAN = 1, FEB, MAR, APR, MAY, JUN,
                    JUL, AUG, SEP, OCT, NOV, DEC};    /* FEB is 2, MAR is 3 */
```

Note: The names must always be distinct, but the associated values need not be.

2.7 Operators in C

Operator	Description
=	assignment
+ - * /	add, subtract, multiply, divide
-	unary minus (negation) — e.g. x = -y
%	modulus (remainder)
++ --	increment, decrement — e.g. x++; --y;
== !=	equals, not equals
< > <= >=	less than, greater than, less than or equal, greater than or equal
&& || !	logical AND, OR, NOT
& | ^ ~	bitwise logical AND, OR, XOR, NOT
<< >>	bitwise left shift, right shift

2.8 Functions in C

Typically, a C program may consist of multiple functions which can possibly be declared in multiple source files.

Like variables, functions must be declared before they are used through function *prototypes*. The prototype of a function gives only the function name, its arguments, as well as the return type; without the function body (i.e. the statements of the function).

The function prototype must agree with the function definition in terms of the number of arguments and their types as well as the return type. However, the argument names need not agree as they are in fact optional in the prototype.

```c
/* foo.h: the header file */

int foo(int a, int b);     /* function prototype */
```

```c
/* foo.c: the source file */
#include "foo.h"

int foo(int a, int b)    /* function definition */
{
   /* ...function code here... */
   return 0;
}
```

As a common practice, function prototypes are often declared in the header file (the `.h` file), while the function definitions are placed in the source file (the `.c` file). If the prototypes are declared in the source file, they should be placed before all the functions (including the `main` function) — at the beginning of the source file.

Note: The header file needs to be included (e.g. `#include "foo.h"`) in the source file where the body of a function is defined. Likewise, whenever a function needs to be used (invoked) in another source file, the corresponding `.h` file must be included to provide the compiler with the necessary information about the function.

The main function Recall that the execution of a compiled C program always begins with the function named `main`. The main function often accepts two parameters which allow you to pass on a number of command-line arguments to a C program when it begins executing.

- `argc`: a count of the number of command-line arguments the program is invoked with;
- `argv`: an array of strings representing the arguments, with the program name as the first argument.

Running the following C program (`sample.c`) with the command: `./sample hello world` prints "`hello world`" as the output.

```c
/* sample.c: display command-line arguments */

#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("%s %s\n", argv[1], argv[2]);   /* argv[0] is the program name */
   return 0;
}
```

Note: Arguments in C functions are passed "by value" — the called function is given the copies of the original arguments.

2.9 Variable Scopes

Variables in C have different scopes (i.e. where they can be referenced) within a program, which are determined by *where* and *how* they are declared.

Global scope Variables declared outside all the functions have the global scope and are visible throughout the entire program; thus they are known as global variables.

Block scope Local variables are those declared within a function (i.e. the braces surrounding a function body). They are accessible from the point at which they are declared to the end of the enclosing function body; hence they are not visible to other functions.

Local variables that are defined with the `static` modifier still have the block scope within the function in which they were declared; however, their values *persist* between function invocations.

```c
#include <stdio.h>

void func(void);

int main(void)
{
        func();    /* x = 1, y = 1 */
        func();    /* x = 1, y = 2 */
        func();    /* x = 1, y = 3 */
}

void func(void)
{
        int x = 0;
        static int y = 0;

        x++;
        y++;

        printf("%d %d\n", x, y);
}
```

Note: Global variables are often defined once in one source file. To make global variables visible in other source files, the `extern` modifier is used during the declaration in the header file.

2.10 Basic I/O in C

As we have seen in the Section 2.1, the standard I/O library (`stdio.h`) provides a set of functions that enable high-level input and output.

Three stardard I/O streams are defined in `stdio.h`: `stdin` (the standard input), `stdout` (the standard output), and `stderr` (the standard error output).

For most of the stdio functions, a reference (pointer) to an open I/O stream must be passed as an argument; however, there are a number of "shorthand" functions that assume one of these streams instead of requiring them to be specified.

The printf function The printf function is an example that we have seen, which is used to write to the stdout stream. It is a formatted output function that converts, formats, and prints arguments to the standard output.

```
#include <stdio.h>

int main(void)
{
  int number = 1;
  char unit_code[] = "FIT2100";

  printf("This is Tutorial %d for %s\n", number, unit_code);
  return 0;
}
```

The first argument in printf is the format string which may contain two types of characters:

- literal characters — any ASCII characters;
- formatting characters (conversion specifiers), with each of which converts the next matching argument (variable) into the desired output format specified using the % directive.

Format Specifier	Description
%c	single characters
%d or %i	signed decimal integers
%e or %E	floating-point numbers in scientific format
%f	floating-point numbers in decimal notation
%d	double precision floating-point numbers in decimal notation
%g or %G	either %f or %e (%E) is used, whichever is shorter
%o	unsigned octal integers
%p	pointers
%s	character strings
%u	unsigned decimal integers
%x or %X	unsigned hexadecimal integers using a-f (A-F)
%%	the percent sign

The scanf function To read characters from the standard input, the scanf function is used. Each of the conversion specifiers (%) must have a matching reference (pointer) specified in the argument list — variables prefixed with &, which the `scanf` function is used to store the input values it interprets.

```c
#include <stdio.h>

int main(void)
{
  int day, month, year;

  printf("Enter the date: ");
  scanf("%d %d %d", &day, &month, &year);

  printf("%d-%d-%d\n", day, month, year);
  return 0;
}
```

Note: Variables where input values are assigned to usually require the & to appear before them. However, you will notice that this is not always the case, in particular when reading strings which are arrays. (This is related to the concept of **pointers** which we will see the next tutorial.)

2.11 The General Structure of C Programs

```c
#include headers
#define constants

/* function prototypes */
void func_A (int a, int b);
. . .

/* declaration of global variables */

int main(int argc, char *argv[])
{
  /* declaration of local variables */
  /* statements of main*/
}

/*function definitions */
void func_A (int a, int b)
{
  /* declaration of local variables */
  /* statements of func_A*/
}

. . .
```

3 Practice Tasks

3.1 Task 1

Write a C program that asks users to enter their first name and last name. The program then displays the *initial* of the users based on the first letter of the first and last names. You may assume the maximum number of characters for each string is 20.

3.2 Task 2

Write a C program that reads the radius of a circle. The program then computes and prints the *circumference* and the *area* of the circle, with two decimal places of precision. You should define the value of π as a constant.

3.3 Task 3

Write a C program that reads two integer values and performs the four arithmetic operations on them: *addition*, *subtraction*, *multiplication*, and *division*. Each of these operations should be implemented as a separate function.

3.4 Task 4

Write a C program that reads a number with three digits. The program then prints the number with its digits reversed. For instance, if the input is `123`, the program should print `321`. You should define an array to hold the digits of the number in the reversed order.

3.5 Task 5

Write a C program that generates a random sequence of three numbers by using the `rand` function from the *stdlib* library. Your program should generate a different sequence each time the program is run.