Computer Programming, M 2020
Lab 2

Week 2, 2020

Objective: To be able to write simple C programs

Part A: Practice

1. A first program in C

```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome to C!
```

Commands to compile & run

Copy and paste the above code in a file named ***myfile.c.*** To compile and run your code, use the following commands:

```
$ gcc myfile.c        # to compile
$ ./a.out             # to run
```

Common Programming Error 2.1

*Forgetting to terminate a comment with */.*

Common Programming Error 2.2

Starting a comment with the characters `*/` or ending a comment with the characters `/*`.

Common Programming Error 2.3

Typing the name of the output function `printf` as `print` in a program.

Common Programming Error 2.4

Using a capital letter where a lowercase letter should be used (for example, typing `Main` instead of `main`).

Good Programming Practice 2.1

Every function should be preceded by a comment describing the purpose of the function.

Good Programming Practice 2.2

Add a comment to the line containing the right brace, `}`, that closes every function, including `main`.

Good Programming Practice 2.3

Indent the entire body of each function one level of indentation (we recommend three spaces) within the braces that define the body of the function. This indentation emphasizes the functional structure of programs and helps make programs easier to read.

Good Programming Practice 2.4

Set a convention for the size of indent you prefer and then uniformly apply that convention. The tab key may be used to create indents, but tab stops may vary. We recommend using three spaces per level of indent.

Some common escape sequences:

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

2. Printing on one line with two **printf** statement.

```c
/* Fig. 2.3: fig02_03.c
   Printing on one line with two printf statements */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome " );
   printf( "to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome to C!
```

3. Printing multiple lines with single **printf**.

```c
/* Fig. 2.4: fig02_04.c
   Printing multiple lines with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome\nto\nC!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome
to
C!
```

4. Addition Program

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Error-Prevention Tip 2.1

Avoid starting identifiers with the underscore character (_) to prevent conflicts with compiler-generated identifiers and standard library identifiers.

C is case sensitive—uppercase and lowercase letters are different in C, so `a1` and `A1` are different identifiers.

Good Programming Practice 2.5

Choosing meaningful variable names helps make a program self-documenting, i.e., fewer comments are needed.

Good Programming Practice 2.9

Place a space after each comma (`,`) to make programs more readable.

Good Programming Practice 2.10

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

Common Programming Error 2.6

A calculation in an assignment statement must be on the right side of the = operator. It is a compilation error to place a calculation on the left side of an assignment operator.

Common Programming Error 2.7

Forgetting one or both of the double quotes surrounding the format control string in a `printf` or `scanf`.

Common Programming Error 2.8

Forgetting the `%` in a conversion specification in the format control string of a `printf` or `scanf`.

Common Programming Error 2.9

Placing an escape sequence such as `\n` outside the format control string of a `printf` or `scanf`.

Common Programming Error 2.10

Forgetting to include the expressions whose values are to be printed in a `printf` containing conversion specifiers.

Common Programming Error 2.11

Not providing a conversion specifier when one is needed in a `printf` *format control string to print the value of an expression.*

Common Programming Error 2.12

Placing inside the format control string the comma that is supposed to separate the format control string from the expressions to be printed.

Common Programming Error 2.13

Using the incorrect format conversion specifier when reading data with `scanf`.

Common Programming Error 2.14

Forgetting to precede a variable in a `scanf` *statement with an ampersand when that variable should, in fact, be preceded by an ampersand.*

Common Programming Error 2.15

Preceding a variable included in a `printf` *statement with an ampersand when, in fact, that variable should not be preceded by an ampersand.*

Arithmetic operators:

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Common Programming Error 2.16

An attempt to divide by zero is normally undefined on computer systems and generally results in a fatal error, i.e., an error that causes the program to terminate immediately without having successfully performed its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.

Precedence of arithmetic operators:

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Order in which a second degree polynomial is evaluated:

```
Step 1.  y = 2 * 5 * 5 + 3 * 5 + 7;     (Leftmost multiplication)
         2 * 5 is 10
Step 2.  y = 10 * 5 + 3 * 5 + 7;        (Leftmost multiplication)
         10 * 5 is 50
Step 3.  y = 50 + 3 * 5 + 7;            (Multiplication before addition)
         3 * 5 is 15
Step 4.  y = 50 + 15 + 7;               (Leftmost addition)
         50 + 15 is 65
Step 5.  y = 65 + 7;                    (Last addition)
         65 + 7 is 72
Step 6.  y = 72                         (Last operation—place 72 in y)
```

Good Programming Practice 2.11

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.

Good Programming Practice 2.14

Although it is allowed, there should be no more than one statement per line in a program.

Part B: Write C Programs

1) Write a program to print your details in the format given below:

Name: XYZ
Roll No. xxxxxxxxxx
Branch: Computer Science and Engineering (or ECE)
Group-number: CP-x

You should not take any input from user. To get the output above, simply use 4 printf's.

2) Write a program to perform the following operations on two numbers:

a) addition b) subtraction c) multiplication
d) division e) modulus

Your program should prompt the user to enter the two numbers.

3) Write a program to calculate the hypotenuse of a right angled triangle, given its base and height. Your program should prompt the user to enter the base and the height of the triangle.

4) Write a program to print the sum of first **100** terms of a given arithmetic progression. Your program should prompt the user to input the first two terms of an arithmetic progression. Then in return your program should print the sum of the first 100 terms of the sequence.

[Hint: Let the first two numbers of the sequence are ***a*** and ***b***, the difference between ***a*** and ***b*** is ***d*** and ***n*** is 100. Sum of the first 100 terms, **S = (n(2a + (n-1)d)/2)**]