

ENGG1003 - Lab 2

Brenton Schulz

1 Introduction

This lab leaves you to perform a series of programming exercises on your own. This can be daunting at first but confidence and perseverance will be crucial skills.

Do not expect programs to work (or even *compile*) first go. It is totally normal, even for experienced programmers, for several lines of hand typed code to generate several errors the first time it is compiled. You will gain experience interpreting the error messages, typically it will be a syntax error due to a missing parenthesis, semicolon, or double quote. Don't be afraid to ask for help and work with other students in the lab.

2 Simple Flow Control Exercises

This first task presents you with all the required building blocks to implement a simple algorithm. All this program will do is read in 2 integers, perform a division, then print the result. A check is performed to make sure a division by zero does not occur.

Task 1

Write a program which implements the following pseudocode:

```
BEGIN
    Integer a
    Integer b
    Integer c
    Read an integer from standard input, store in a
    Read an integer from standard input, store in b
    IF b is zero
        PRINT "Division by zero error"
        RETURN // stop executing
    ENDIF
    c = a/b
    PRINT "a divided by b is" <the value of c>
END
```

Notes:

- Reading each integer can be done with: `scanf("%d", &a);` (with an appropriate substitution for the variable name). You may prompt the user by placing a `printf();` *without* newline character prior to `scanf();`, eg:

```
1 printf("Enter an integer: ");
2 scanf("%d", &a);
```

- Note that <the value of c> in the pseudocode is implying that a number should be printed there, ie:

```
1 printf("a divided by b is %d\n", c);
```

- The RETURN pseudocode can be implemented with:

```
1 return 0;
```

- The condition "b is zero" needs to be implemented with the `==` (*two* equals symbols) operator.
- You may start with the default code listing provided by OnlineGDB

3 C Summary

This section will be included in all future lab documents and lists a summary of C language features taught prior to the lab session. It will grow each week.

Not everything listed in this section is required to complete a particular lab.

3.1 Basic Structure

```
1 #include <stdio.h>
2 int main() {
3     // Your program goes here
4     return 0;
5 }
```

3.2 Comments

```
1 // This is a comment to end of line
2
3 /* this is a block comment
4    which could span
5    multiple
6    lines
7    */
```

3.3 Operators

Operation	C Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Increment	++
Decrement	--
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Equal to	==
Not equal to	!=

Table 1: Arithmetic operators in C

3.4 Operator Shorthand

Many arithmetic operators support the following shorthand syntax. The left and right columns present equivalent statements.

$x = x + y;$	$x += y;$
$x = x - y;$	$x -= y;$
$x = x * y;$	$x *= y;$
$x = x / y;$	$x /= y;$

3.5 Data Types

Type	Bytes	Value Range
char	1	-128, +127
unsigned char	1	0, 255
short	2	-32768, 32767
unsigned short	2	0, 65535
int	4	$\approx \pm 2.1 \times 10^9$
unsigned int	4	0, 4294967296
long	8	$\approx \pm 9.2 \times 10^{18}$
unsigned long	8	0, 1.8×10^{19}
float	4	1.2×10^{-38} to 3.4×10^{38}
double	8	2.3×10^{-308} to 1.7×10^{308}

3.6 Standard i/o

Read a single variable from stdin with `scanf()`;
`scanf("format specifier", &variable);`

Write a single variable to stdout with `printf()`;
`printf("format specifier", variable);`

You can use `printf()` without a newline (`\n`) to create an input prompt:

```
1 printf("Enter a number: ");
2 scanf("%d", &variable);
```

This prints:

Enter a number: _

where _ indicates the terminal prompt (ie: where typed characters will appear).

3.7 Format Specifiers

The following table is woefully incomplete. The compiler *may* generate warnings if `%d` is given something

other than `int` and `%f` is given something other than `float`. An attempt will be made to ensure these are sufficient.

Data Type	Format Specifier
Integers	<code>%d</code>
Floating point	<code>%f</code>
Float with <i>n</i> decimal places	<code>%.nf</code>

Table 2: Basic format specifiers

3.8 Type Casting - Advanced

Placing the syntax `(type)` before a variable name performs a type cast (ie: data type conversion).

eg: convert `float` to an `int` prior to using its value. This forces a rounding-down to the nearest integer.

```
1 float a;
2 // ...
3 y = (int)a * z;
```

NB: This does **not** modify the original variable.

Data type “upgrades” are done automatically by the compiler but sometimes it is desired to downgrade or force esoteric behaviour. Adding it unnecessarily doesn’t have any negative impact. Applications in ENGG1003 will be limited but it comes up regularly in embedded systems and nobody else explicitly teaches type casting. I have used it extensively in the low-level art of *bit banging*: manual manipulation of binary data. This is, unfortunately, beyond ENGG1003.

3.9 Flow control

Flow control allows selected blocks of code to execute multiple times or only under a specified condition.

3.9.1 `if()`

The `if()` statement executes a block of code only if the *condition* is true. The condition is an arithmetic statement which evaluates to either zero (false) or non-zero (true).

Syntax:

```
if(condition) { /* other code */ }
```

Full example:

```
1 if(x > 10) {
2     // Do stuff
3 }
```

Condition Examples:

- `if(x)` // `if(x is not zero)`
- `if(x+y)` // `if((x+y) is not zero)`
- `if(y >= 5)`
- `if(1)` // Always executes
- `if(0)` // Never executes
 - Can be used for debugging. Might be easier than a block comment `/* */`

NB: *NEVER* place a semicolon after an `if()`, that stops it from having any effect. The block after it will always execute. This bug can take days to find.

3.9.2 `while()`

The `while()` flow control statement executes a block of code so long as a condition is true. The condition is checked before the block is executed and before every repeated execution.

The condition rules and examples are the same as for those listed under the `if()` statement.

Syntax:

```
while(condition) { /* other code */ }
```

Example:

Evaluate the infinite sum:

$$\sum_{n=0}^{\infty} \frac{1}{n^2} \quad (1)$$

to a precision of 1×10^{-6}

```
1 float sum = 0.0;
2 int x = 0;
3 while(1/(x*x) > 1e-6) {
4     sum = sum + 1/(x*x);
5     x++;
6 }
```

3.10 Library Functions

3.10.1 `rand()`

To generate a random number between 0 and MAX:

```
1 #include <stdlib.h> // For rand()
2 // ...
3 x = rand() % MAX;
```

For all work in this course you may assume that the above method works well enough.

For more crucial work (eg: cryptography, serious mathematics) this method is considered problematic. Very advanced discussion [Here](#).