CST8234 - C Programming
Winter 2017
Lab 01: Setup Lab

Setup:

By the end of this lab, you should have properly installed in your laptop:

1. A Linux virtual machine or a Linux desktop / server, Ubuntu or Fedora are recommended, 64-bit or 32-bit is OK. If using a virtual machine, be sure your VM has at least 4GB of RAM assigned to it.
2. Create an account as your **username** (from Algonquin)
3. To verify your version of Linux:

```
/* To verify your Kernel version */
# uname -r
 4.4.0-34-generic
/* To verify your hardware platform – in this case it shows a 64 bit arch */
# uname -m
x86_64
```

4. Be sure you have **gcc** properly installed.

```
/* To verify your gcc version */
# gcc -v
```

5. Download and compile and run **hello.c**

```
# gcc -o hello hello.c
```

6. Have a share folder in between your host and guest OS. This facilitates having access to your programs. You can use dropbox or any other mechanisms. If using VMWare, here is more information: http://www.vmware.com/support/ws4/doc/running_sharefold_ws.html

Using gdb

Compile and run **divide.c** using **gdb**

1. Download **divide.c**
2. Compile the program to be able to use **gdb**

```
root@luna:CST8234/Labs/00_Lab# gcc -g -o divide division.c -ansi -pedantic -Wall
```

3. Run **gdb** and load **divide**

```
root@luna:CST8234/Labs/00_Lab# gdb
(gdb) file divide
Reading symbols from CST8234/Labs/00_Lab/divide...done.
```

4. Run **divide:**
   You can use the **run** command or the short version **r**

```
(gdb) r
Starting program: CST8234/Labs/00_Lab/divide divide
10 / 5 = 2

Program received signal SIGFPE, Arithmetic exception.
0x00000000004005bb in divide (a=10, b=0) at division.c:47
47          return a / b;
```

gdb is saying that it encountered an arithmetic exception (**SIGFPE**) when running th program. The error was encountered at line **47** in the program **division.c**
The program was executing the function **divide** with arguments **a=10** and **b=0**
Line **47** executes **return a / b**;

5. Use the command **`list`** or **`l`** for short. This command allows you to see the context of the crash, listing codes near around line **`47`** of **`divide.c`**

```
(gdb) list

45    int divide( int a, int b ) {
46
47            return a / b;
48    }
```

6. Move one level of execution up with the **`up`** command. The crash happened at the **`divide`** function. You want to see what happens in the function that called **`divide`**, in this case **`main`**.

```

(gdb) up
#1  0x0000000000400579 in main () at division.c:34
34            divide( x, y );
(gdb) list
29            y = 5;
30
31            printf("%d / %d = %d\n", x, y, divide( x, y ));
32
33            y = 0;
34            divide( x, y );
35            printf("%d / %d = %d\n", x, y, divide( x, y ));
36
37            return 0;
38    }
```

7. Print the values of the variables **`x`** and **`y`**. Notice that when in the **`divide`** function the arguments are called **`a`** and **`b`**

```
(gdb) print x
$1 = 10
(gdb) print y
$2 = 0
```

Program #1:

This is a programming exercise, you may use **any programming language** you feel comfortable with. Write a small program: **`numbers.c`** – In case your are using C, use appropriate file extension for any other language – that:

1. Prints the numbers from 1 to 100
2. If the number is a multiple of three, it should print instead "**`I'm a multiple of 3!`**"
3. If the number is a multiple of five, it should print instead "**`I'm a multiple of 5!`**"
4. If the number is a multiple of three and five, it should print instead "**`I'm a multiple of 3 && 5!`**"

This program should not take you more than 10 minutes to write.
The following demonstrates the execution of the program:

```
#./numbers
...
8
9 I'm multiple of 3!!!
10 I'm multiple of 5!!!
11
12 I'm multiple of 3!!!
13
14
15 I'm multiple of 3 && 5!!!
```

Excerpt SAMPLE TEST OUTPUT: numbers