

## Lab 2.2 Running a Hello World Program in C using GCC

### Overview

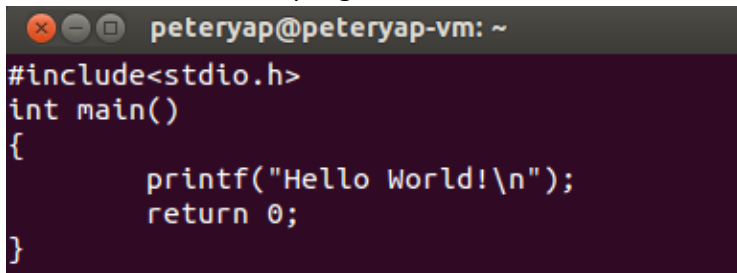
The lab helps familiarize you with writing a simple Hello World program using C, the GCC compiler [link](#), and Pico(a text editor, [link](#)). It uses Ubuntu VM created in Lab 2.1. Here is lab objective:

1. Learn to run a program in gcc.
2. Learn to debug a program in gdb.

### Steps :

1. Writing a simple Hello World program using C language. Create a .c file using text “vi hello.c”. `peteryap@peteryap-vm:~$ vi hello.c`

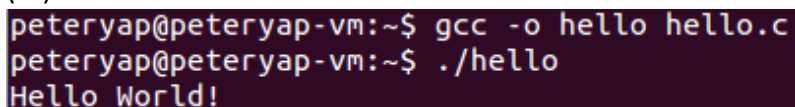
Write a Hello World c program.



```
peteryap@peteryap-vm: ~  
#include<stdio.h>  
int main()  
{  
    printf("Hello World!\n");  
    return 0;  
}
```

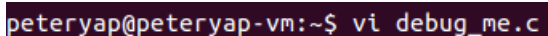
2. Save and exit the text editor. Insert the command with gcc -o hello hello.c to compile the C program, then insert ./hello to execute the c program. “Hello World!” is printed out.

This command will invoke the GNU C compiler to compile the file hello.c and output (-o) the result to an executable called hello.



```
peteryap@peteryap-vm:~$ gcc -o hello hello.c  
peteryap@peteryap-vm:~$ ./hello  
Hello World!
```

3. Create debug\_me.c file. “vi debug\_me.c”.



```
peteryap@peteryap-vm:~$ vi debug_me.c
```

- Copy the code of lab2.2 website and add two more include header file which is `#include<stdio.h>` and `#include<stdlib.h>`

```
peteryap@peteryap-vm: ~
#include<stdio.h>
#include<stdlib.h>

print_string(int num, char* string)
{
    printf("String '%d' - '%s'\n",num,string);
}

int main(int argc, char* argv[])
{
    int i;
    if(argc < 2){
        printf("Usage: %s [...] \n",argv[0]);
        exit(1);
    }

    for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
        print_string(i, argv[0]);
    }
    printf("Total number of strings: %d\n", i);
    return 0;
}
```

- Insert two command “gcc -g debug\_me.c -o debug\_me” and “gdb debug\_me”

```
peteryap@peteryap-vm:~$ gcc -g debug_me.c -o debug_me
peteryap@peteryap-vm:~$ gdb debug_me
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/peteryap/debug_me...done.
(gdb)
```

- In the gdb command insert run “hello,world” “goodbye,world”.

```
(gdb) run "hello,world" "goodbye,world"
Starting program: /home/peteryap/debug_me "hello,world" "goodbye,world"
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x7ffff7ffa000
String '1' - 'hello,world'
String '2' - 'goodbye,world'
Total number of strings: 3
[Inferior 1 (process 3187) exited normally]
```

- Setting breakpoint by using “break main” or “break debug\_me.c:19” at a specific line.

```
(gdb) break main
Breakpoint 1 at 0x40057f: file debug_me.c, line 13.
(gdb) break debug_me.c:19
Breakpoint 2 at 0x4005bd: file debug_me.c, line 19.
```

- After setting breakpoints, run the program step by step with two methods.(next or step).

The difference between "next" and "step" is that "step" stops inside a called function, while "next" executes called functions at (nearly) full speed, stopping only at the next line in the current function.

9. Use the two methods as below :

```
(gdb) run
Starting program: /home/peteryap/debug_me "hello,wolrd" "goodbye,world"
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x7ffff7ffa000

Breakpoint 1, main (argc=3, argv=0x7ffffffffffe298) at debug_me.c:13
13         if(argc < 2){
(gdb) step
18         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) next

Breakpoint 2, main (argc=2, argv=0x7ffffffffffe2a0) at debug_me.c:19
19         print_string(i, argv[0]);
(gdb) next
String '1' - 'hello,wolrd'
18         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) next

Breakpoint 2, main (argc=1, argv=0x7ffffffffffe2a8) at debug_me.c:19
19         print_string(i, argv[0]);
(gdb) next
String '2' - 'goodbye,world'
18         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
```

When running in the second loop, we can use the print method to check the value of variable i, insert command "print i".

```
(gdb) next

Breakpoint 2, main (argc=1, argv=0x7ffffffffffe2a8) at debug_me.c:19
19         print_string(i, argv[0]);
(gdb) next
String '2' - 'goodbye,world'
18         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) print i
$1 = 2
(gdb) print i
$2 = 2
(gdb) next
21         printf("Total number of strings: %d\n", i);
(gdb) next
Total number of strings: 3
22         return 0;
(gdb) print i
$3 = 3
```

When use the next method and goes out of the for loop, the integer i++, and print i again, we can see that the variable become 3.

10. We can also use the command “where” to check the function call stack.

```

Breakpoint 1, main (argc=3, argv=0x7fffffff298) at debug_me.c:15
15         if(argc < 2){ /* 2-1 for program name (argv[0]) and one for a pa
ram.*/
(gdb) step
21         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) step

Breakpoint 2, main (argc=2, argv=0x7fffffff2a0) at debug_me.c:22
22         print_string(i, argv[0]); /* function call */
(gdb) step
print_string (num=1, string=0x7fffffff55d "hello,world") at debug_me.c:7
7         printf("String '%d' - '%s'\n",num,string);
(gdb) where
#0  print_string (num=1, string=0x7fffffff55d "hello,world") at debug_me.c:7
#1  0x000000004005d1 in main (argc=2, argv=0x7fffffff2a0) at debug_me.c:22
(gdb) step
String '1' - 'hello,world'
8     }
(gdb) step
main (argc=2, argv=0x7fffffff2a0) at debug_me.c:21
21         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) print i
$6 = 1
(gdb) print i
$6 = 1
(gdb) frame 0
#0  main (argc=2, argv=0x7fffffff2a0) at debug_me.c:21
21         for(argc--, argv++,i=1 ; argc>0 ; argc--,argv++,i++){
(gdb) step

Breakpoint 2, main (argc=1, argv=0x7fffffff2a8) at debug_me.c:22
22         print_string(i, argv[0]); /* function call */
(gdb) step
print_string (num=2, string=0x7fffffff569 "goodbye,world") at debug_me.c:7
7         printf("String '%d' - '%s'\n",num,string);
(gdb) step
String '2' - 'goodbye,world'
8     }
(gdb) print i
No symbol "i" in current context.
(gdb) frame 1
#1  0x000000004005d1 in main (argc=1, argv=0x7fffffff2a8) at debug_me.c:22
22         print_string(i, argv[0]); /* function call */

```

Using the “frame” command to switch, different frame print out different variable i, because of the variable i is not defined in the function, however the variable i in main function is i = 1.

11. The Lab is completed.