1 Calling standard C functions

Consider the following program:

```

        extern printf   ; C
        extern exit     ; C
        section .data
                msg1 DB "Hello world!",0
                msg2 DB "This is a test!",0
                fmt DB "Output is: %s %s",10,0
        section .text
                global main

                main:
                        PUSH msg2
                        PUSH msg
                        PUSH fmt
                        CALL printf

                        PUSH 10
                        CALL exit
```

Here, we are calling some C standard functions. **extern** means that which is external. In this case, **printf** and **exit** are external functions. These are the standard functions of C. In x86 assembly, The C parameters are passed to the called routine by pushing them onto the stack. They are pushed onto the stack from right to left(see: StackOverflow).

The **PUSH** instruction basically pushes the arguments and parameters into the stack(**LIFO**). Now, consider the snippet:

```
PUSH msg2
PUSh msg
PUSH fmt
CALL printf
```

We know that the **printf()** function in C is of used as: **printf(".....%s...%d...%f....%....", string1, var1, fvar1, ...);**. Keeping that in mind when **printf()** will be called/invoked, then it should be called in this exact syntax. To ensure that we need to push into the stack in such an order that while popping from the stack the order is matched. So, here, we first push **msg2**, then **msg** and finally **fmt**. So **fmt** was pushed last. In stack, the last element to be inserted is the first element to be eliminated. So **fmt** will be popped first, then **msg** will be popped and then finally **msg2** will be popped(it was the first element to be inserted) and hence the order of arguments for **printf()** will remain intact. First the format specifiers, then the values corresponding to those format specifiers.

Now, to produce the executable for this program, we will need to use **nasm** and **gcc** as follows:

```
        $ nasm -f elf -o <filename>.o <filename>.asm
```

Next, we will use the **gcc** command:

```
        $ gcc -no-pie -m32 <filename>.o -o <executableName>
```

The option **-no-pie** disables **P**osition **I**ndependent **E**xecutable generation. The **-m32** specifies that the compilation should be done for a 32-bit target.

Also, while running this command, I encountered some issues:

```
$ gcc -no-pie -m32 asm15.o -o asm15

/usr/bin/ld: cannot find crt1.o: No such file or
    directory
/usr/bin/ld: cannot find crti.o: No such file or
    directory
/usr/bin/ld: skipping incompatible /usr/lib/gcc/
    x86_64-linux-gnu/11/libgcc.a when searching
    for -lgcc
/usr/bin/ld: cannot find -lgcc: No such file or
    directory
/usr/bin/ld: skipping incompatible /usr/lib/
    x86_64-linux-gnu/libgcc_s.so.1 when searching
    for libgcc_s.so.1
/usr/bin/ld: cannot find libgcc_s.so.1: No such
    file or directory
/usr/bin/ld: skipping incompatible /usr/lib/
    x86_64-linux-gnu/libgcc_s.so.1 when searching
    for libgcc_s.so.1
/usr/bin/ld: skipping incompatible /usr/lib/gcc/
    x86_64-linux-gnu/11/libgcc.a when searching
    for -lgcc
/usr/bin/ld: cannot find -lgcc: No such file or
    directory
collect2: error: ld returned 1 exit status
```

So I installed the 32-bit C library development files using this command:

```
$ sudo apt-get install libc6-dev:i386
```

After this, I again used **gcc**, and encountered these issues:

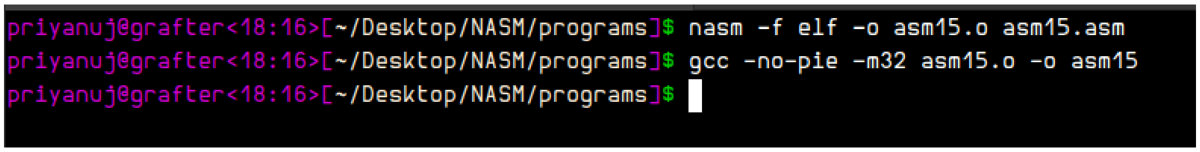
```
$ gcc -no-pie -m32 asm15.o -o asm15

/usr/bin/ld: skipping incompatible /usr/lib/gcc/
    x86_64-linux-gnu/11/libgcc.a when searching
    for -lgcc
/usr/bin/ld: cannot find -lgcc: No such file or
    directory
/usr/bin/ld: skipping incompatible /usr/lib/gcc/
    x86_64-linux-gnu/11/libgcc.a when searching
    for -lgcc
/usr/bin/ld: cannot find -lgcc: No such file or
    directory
collect2: error: ld returned 1 exit status
```

To resolve these issues, I installed **gcc-multilib**(Source: Click here):

```
$ sudo apt-get install gcc-multilib
```

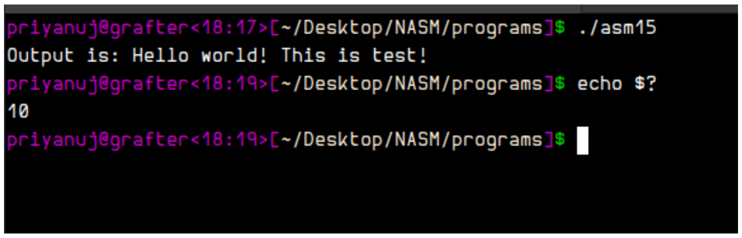
Finally, it worked!!



```
priyanuj@grafter<18:16>[~/Desktop/NASM/programs]$ nasm -f elf -o asm15.o asm15.asm
priyanuj@grafter<18:16>[~/Desktop/NASM/programs]$ gcc -no-pie -m32 asm15.o -o asm15
priyanuj@grafter<18:16>[~/Desktop/NASM/programs]$
```

Figure 1: Success!

We can see the output of this program. The value 10 that we had pushed gets stored in **$?** and we can see its value as shown below:



```
priyanuj@grafter<18:17>[~/Desktop/NASM/programs]$ ./asm15
Output is: Hello world! This is test!
priyanuj@grafter<18:19>[~/Desktop/NASM/programs]$ echo $?
10
priyanuj@grafter<18:19>[~/Desktop/NASM/programs]$
```

Figure 2: Output