

1 Usefulness of 'printf()'

`printf()` is one of the most useful functions for beginners in the C programming language. It is a very helpful tool when it comes to debugging, checking statuses of dynamic variables, or simply printing to console. What many users are unaware of is that the function may be used for other very useful tricks.

The formal declaration of the function is:

```
int printf(const char *format, ...)
```

where the returning type `int` is in fact the total number of characters written. A negative number is returned on failure. This property can be used in interesting ways, such as described in the snippet below:

```
#include <stdio.h>

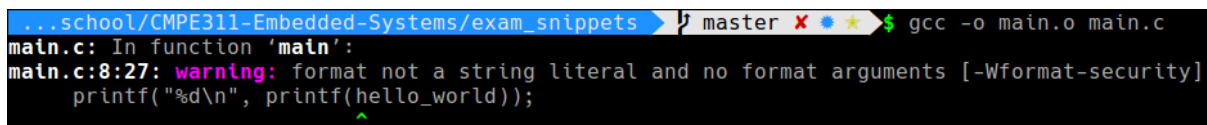
int main() {

    char hello_world[] = "hello world";

    printf("%d\n", printf(hello_world));
    return 0;

}
```

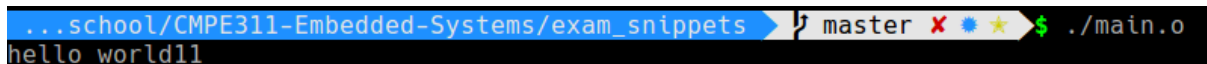
The snippet described will build with a warning:



```
...school/CMPE311-Embedded-Systems/exam_snippets master X ● ★ $ gcc -o main.o main.c
main.c: In function 'main':
main.c:8:27: warning: format not a string literal and no format arguments [-Wformat-security]
printf("%d\\n", printf(hello_world));
               ^
```

Figure 1: Building Snippet 1

And when ran, will output interesting results:



```
...school/CMPE311-Embedded-Systems/exam_snippets master X ● ★ $ ./main.o
hello world11
```

Figure 2: Running Snippet 1

The output on the console consisted of the inner `printf()` function printing out the string first, then returning its number of characters to be dumped by the outer `printf()`.

The function can also be used for a very simple debugging purpose - finding the data type of a variable. If possible, `printf()` will attempt to output any valid arguments passed to it with

its corresponding formatting string. However, passing in an incompatible data type variable with the "%d" string formatter will result in most compilers warning the user during the build. The warning will include the actual data type of the variable, such as the following snippet:

```
#include <stdio.h>

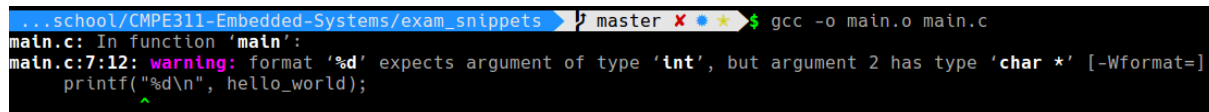
int main() {

    char hello_world[] = "hello world";

    printf("%d\n", hello_world);
    return 0;

}
```

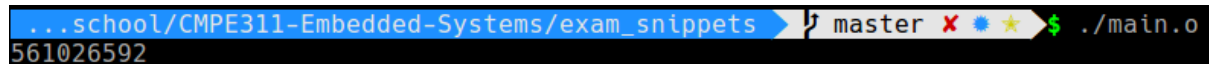
The snippet will invoke the following warning when built:

A terminal window showing the compilation of a C program. The prompt is "...school/CMPE311-Embedded-Systems/exam_snippets" with a "master" branch indicator. The command "gcc -o main.o main.c" is entered. The output shows a warning: "main.c:7:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'char *' [-Wformat=]". The warning points to the line "printf("%d\n", hello_world);".

```
...school/CMPE311-Embedded-Systems/exam_snippets master X ● ★ $ gcc -o main.o main.c
main.c: In function 'main':
main.c:7:12: warning: format '%d' expects argument of type 'int', but argument 2 has type 'char *' [-Wformat=]
    printf("%d\n", hello_world);
           ^
```

Figure 3: Building Snippet 2

thus, notifying the user of their variable's data type. Of course, printing out the variable with the wrong string formatter would not be very useful!

A terminal window showing the execution of the compiled program. The prompt is "...school/CMPE311-Embedded-Systems/exam_snippets" with a "master" branch indicator. The command "./main.o" is entered. The output is "561026592".

```
...school/CMPE311-Embedded-Systems/exam_snippets master X ● ★ $ ./main.o
561026592
```

Figure 4: Running Snippet 2