Handout - Static Storage Class (5/4)

This handout is an extension of what we discussed about the static storage class in lecture today.

What does the keyword *static* placed in front of a function or variable definition mean?

[1] - A static variable inside a function can keep its value between invocations.

Why? Because static variables are stored in the data or bss segment and not in the function's stack frame.

Example: static.c code which we discussed in class (posted in the handouts section.)

[2] - A static *global* variable or function are *visible* only in the file they are defined in.

main.c

```
static int static_var = 0;
int global_var = 1;

main ( ) {
   ...
}
```

func.c

```
// global_var is visible here.
// static_var is not visible here.

func() {

// global_var is visible here
// static_var is not visible here.

}
```

A non-static global variable on the other hand can be accessed from an other file using *extern*. Same restrictions apply for functions.

Surprise Bonus Fact!
Did you know that if you don't specify an executable file name (using the -o option) when creating your executable object file, gcc assigns a default name of **a.out**?

Example

`gcc myfile.c -m32 -o myfile` *Creates an executable object file named **myfile***

`gcc myfile.c -m32` *Creates an executable object file named **a.out***