What are the diffrence between static and external variables.what are the advantages of making a static variable? why do we prefer making external variables in multifunction programs?the book says that else it gets ititialized very late . I will try to explain:

Problem is that in C static can have different meanings. I will try to give an overview of the different situations in the following paragraphs.

If a variable is defined outside a function, it can be used by all functions in the file. Sometimes also called 'global variable'. This means that there is only one instance of this variable for the whole file. Also the name of the variable is stored in the resulting .OBJ file. This latter is important, since if another file also defines a variable with the same name outside a function, the linker assumes that it's the same variable in both cases, and merges them. To make this extra clear, it's best to add the keyword "extern" to one of the variables. In this case, we say that we declare the variable, instead of defining it. It is an extra signal for the compiler/linker to indicate that we actually want to refer to a global variable defined somewhere else.

If you want to define a global variable, but don't want to make it available to other files, add the keyword static before. The keyword static tells the compiler that the variable name must not be stored in the .OBJ file. This means that two .C files with the following line:

```
static long MyGlobalVariable;
```

will each have their own variable. Both variables will be called MyGlobalVariable.

If you define a variable inside a function, it becomes a local variable. It comes into existence if the function is called, and disappears again after the function is finished. In some situations you want to keep the value of the variable in between function calls. You could do this by using a global variable (instead of a local variable) but then the variable becomes available for all functions in the file, which you don't necessarily want. In that case you can put the keyword static before the variable, like this:

```
void MyFunction()
{
static long MyLocalVariable = 0;
++MyLocalVariable;
}
```

The first time the function is called, MyLocalVariable will be 'created' and initialized with the value 0. At the end of the function, the variable is not destroyed, but kept. So the next time you call this function, the value of the variable will be 1, not zero.

In C it really doesn't matter whether you put the variable outside the function (as global variable) or define it as static inside the function. The only difference is where the variable can be accessed.

In C++, things are quite different. If you write this (outside a function):

```
MyClass MyGlobalVariable;
```

MyGlobalVariable will be constructed (this is: the constructor will be executed) at the start of the application, before even main is called. However, you don't have real control over the order in which all global variables are constructed. So if another file contains this:

```
MyOtherClass MySecondGlobalVariable;
```

You can't know for sure whether MyGlobalVariable or MySecondGlobalVariable is constructed first. This can give problems if the constructor of one of them relies on the presence (construction) of the other one.

On the other hand, if you define the variable as static inside a function:

```
void MyFunction()
{
static MyClass MyStaticVariable;
}
```

Then MyStaticVariable will be constructed the first time the function is called. With this construction, you can write something like this:

```
MyClass &getMyClass()
{
static MyClass MySelf;
return MySelf;
}
```

And we have implemented a singleton where we have control on when it is constructed. The first time we need it, it's constructed.

To be honest, this approach is a rather simplistic one, because it may lead to problems in multi-threaded applications. In that case, there are other tricks.