Global vs Local Variables

Recently we had a small discussion in one of the threads regarding local and global variables and its advantages or disadvantages.
I would like to show up what I have learnt so far about this topic.

Some vocables beforehand:

VARIABLE refers to a memory location that the program can access and read or alter its value.

SCOPE describes the visibility of a variable. That is the area within the complete c-program from which that variable can be red or written to.

EXTENT describes the existence of a variable. That is the duration when memory for that variable is allocated and may be accessed.

From those definitions it is obvious that both properties for a variable must be fulfilled when it is accessed: It must be within the scope and it has to exist. Usually the compiler will raise errors when trying to use a variable which is out-of-scope but access to a non-existing var will (usually) not be detected and even might crash the program's execution.[1]

Any variable declared outside a function (remember: main() is also a function) is a **global** variable and its scope begins with its declaration and ends with the last line of the source. Its extend is from the start of the program - first line in main() - to the end of the program which is normally in the embedded world the moment the power is switched off.

A variable declared within a function (again: main() is also a function) has its scope within that function only, it is a **local** variable. It cannot be accessed from outside the function and normally its extent is too within the function only. "Normally" means that this behaviour can be overridden by the storage-class "static" which will give that var the same life-long extent as a global var.

How is that managed by the compiler.
C is usually realized on CPUs with good stack-handling, so the PSoC3 with its 8051 core has given the c-designers a lot of headaches, I presume.
Local vars are allocated on the stack at the entry to the function where they

1 To access a non-existing variable is quite easy: just let a function return the pointer to a local variable (preferably a string). Whatever you do with that pointer (except forgetting it) this will be the first steps leading into disaster.

are defined in and they are freed when the function returns. This ensures a pretty good utilization of the SRam the stack normally resides in, unneeded vars simply do not "exist" when not used, giving their memory to other vars at need. Programmers like that!

What (dis)advantages does that facts have for us software-designers?
Well, the first fact is that a global variable can be altered from anywhere within the program and that might not be what is wanted. On the other hand this variable is visible throughout the program and so may save some lengthy parameter-lists in function calls to have them accessed.

Now for the beef: Some CPU-cores have trouble dealing with memory on the stack (PSoC1, PSoC3) and so the compiler has to generate extraneous code to perform what we want. In time-critical missions this would hinder a code-efficient access to local vars. A help out of that is to use the storage-class "static" which will put that local var into normal SRam but retaining the limited scope as we want. Care has to be taken with the initialization of static local vars since those will be initialized only once when the function is called the very first time while normal local vars are initialized each time the function is entered.

I have seen more than once programs in which a for-loop variable was declared as global which is flagged as an error by some compilers. Easy to understand that the next for(i=0;... will clobber any value previously assigned to that var and not even a warning hints the desperate programming guy to the cause of that bug.
I try to avoid global declared vars, so I tend to reduce global vars to its absolute minimum, which usually are only those that are referred from other modules as external vars. Additionally I remove all vars from main() since they shrink the stack unnecessarily. The reduced scope of local variables not only makes the program safer but additionally more readable (one of my favourite arguments).

Happy coding
Bob Marlowe