Statistics 243: *class notes*

William J. De Meo

9/22/97

Topics

1. Functions
    - 1.1 Communication Between Functions
    - 1.2 External (Global) Variables

1 Functions

A good function for summing:

```
double sum(double *x, long n)
{
double xs;
xs=0;
while(n--) xs += *(x++);
return(xs);
}
```

We pass the values stored at the address `x`, and use these values to compute the sum. The function never changes the values stored at `x`.

1.1 Communication Between Functions

The *scope* of a variable describes how widely the value of that variable is known to your program. Generally speaking, one should only pass to a function the information that the function really needs. This is a concept known as information hiding, and is based on the idea that, the less a function has access to, the less it can damage. A function should rarely need access to anything not passed to it in its argument list. There are exceptional cases, however:

- Suppose we have a large graphics subsystem. We have variables like scale, origin, line type, axis information, etc. that all functions need to know. We could put all these variables in a structure and pass it to every function.
- Suppose we have a function which accepts another function as an argument, but has a limited argument list which we don't want to change. Then we need a new way to communicate between functions.

Keep in mind that these are special circumstances. In general you should be wary of giving functions access to too much.

1.2 External (Global) Variables

In one and only one file, we declare an external or global variable outside of the main function.

```
double *x;

main()
{ ...
```

This variable is automatically accessible by anything inside this file. To access that variable from another file, declare it with the `extern` qualifyer, which says, don't create new memory for this variable, go out and look for the existing variable. E.g., in the file funct1.c, we have:

```
extern double *x;

funct1()
{ ...
```

In a third file, we might only want access to the global variable from within the `funct3()` function:

```
funct3()
{
extern double *x;
...
```

In the first case scenario, that of a large graphics subsystem, it is a stupid idea to declare all the variables as extern. Instead, for communication within a single file, declare the variable outside any curly braces, using the `static` qualifier. The variable is now available to any function in that file.

Don't use these techniques just because defining the same kind of variable is getting tiresome. If something goes wrong with the value of a variable that has a large scope, you will have to find out what's going on in each function that has access to the variable – a big pain.

Another effect of the static qualifier is that the memory allocated for a static variable is untouched throughout the program, so the value store there can only be changed by explicitly changing the static variable. i.e. the compiler will not reallocate the memory. Ex:

```
double func(long int n)
{
static double *x;
static long start=1;
if(start==1)
x = dmalloc(n);
start = 0;

...
```