****

Chapter 6 - Storage Classes

****

C uses 4 storage classes: auto, register, extern, static

– Class determines the variable's *duration*, *scope* and *linkage*

 duration - some variables last the life of the program, others shorter

 scope - where is it visible and can therefore be used

 linkage - only the current file or also by other source files

e.g.,

```
static int x;
```

2 ***durations***:

<u>automatic storage duration</u> (auto and register)

 lives on the stack or in a register

  created on entering a block, destroyed when leaving

  normal function variables (local) are like this

  keyword auto is rarely used - it is the default

  "register" suggests to the compiler to place it in a register - may ignore

   today's compilers are pretty good at figuring what to do

e.g.,

```
int countcalls() {
    static int count = 0;
    return ++count;
}
```

<u>static storage duration</u> (extern and static)

  these variables exist for the life of the program

  storage is allocated and initialized once

  (scope of variable can be local or global)

  global variables and function names are extern by default

   global variables declared outside any function

   scope is from the point of declaration to the end of the file

 * local variables can be declared as static - value persists between calls

 * static variables are initialized to zero if you don't initialize

more on extern and static later

local variable x:

```
int x;        // not initialized

static int x;  // initialized to 0
```

Scope: where can we reference and use the variable

four scopes are:

<u>function scope</u>

✗ labels (identifier followed by colon) e.g., start: (e.g., inside switch)

   - only identifiers with function scope

   can't be referenced outside the funciton

<u>file scope</u>

✓ identifier declared outside any function

  visible from that point to the end of the file

   (global variables and function definitions and function prototypes)

<u>block scope</u>

✓ identifiers inside a block (surrounded by {})

   - we've seen this where block is a function

  scope ends at end of block

  variables must be declared at the beginning of the block

  blocks can be nested

  variables can have the same name - inner one hides the outer one

```
while ( ) {
    int x;   <--- different
    :                |
    for ( ) {        v
        int x;
    }
}
```

<u>function-prototype scope</u>

✗ names used in the function prototype

  these are actually ignored as the scope is only within the prototype

```
fct( float principle, float interest, int periods);
fct( float, float, int);
```

Static int x;
extern
volatile   Variable can change at any moment
Const   Variable will not change

```
Const int x = 42;        x can never change
```

```
Const int x;
    x=42;        Error
```

```
#define    x    42
```

define macro

```
#define    square(x)     ((x) * (x))
#define    doubleit(x)   ((x) + (x))
```

```
square (3)
square (2+3)                    ((2+3) * (2+3))
doubleit(4) * doubleit(2)       ((4)+(4)) * ((2)+(2))
```

User-defined type

```
enum (red, green, blue);
      0     1      2

x = red
  ⋮
if (y == blue) do something

enum color (red, green, blue);

enum color mycolor = red
```

$\uparrow$ new type $\uparrow$ variable name

$$\text{enum}\ (\text{jack} = 11,\ \overset{\overset{12}{\swarrow}}{\text{queen}},\ \overset{\overset{13}{\swarrow}}{\text{king}});$$

$$\text{enum}\ (\underset{\underset{12}{\uparrow}}{X} = 12,\ \underset{\underset{13}{\uparrow}}{y},\ \underset{\underset{75}{\uparrow}}{z} = 75);$$