CSCI 135
Storage Classes

What is a Storage Class?

<storage class> <data type> <name1>,<name2>,. . . ;

The **scope** of a variable is the part of the program where the variable is visible (and can be accessed).

The **lifetime** of a variable is the time duration (with respect to program) in which the variable exists in memory.

Storage classes:

- Used to specify the variable's lifetime (and somewhat, scope)
- Also specifies area of memory where variable/object is stored (*e.g.*, heap)
- Also specifies information about variable that can be used when splitting program across multiple modules/files (*i.e.*, for use by linker)
- Has default values based on position in program, which are normally (but not always!) what we want (we have used defaults so far this semester)

C Storage Classes

- Auto: The variable starts existing when first declared, and is dead when exiting scope in which it was declared. If the scope is re-entered, a new variable with the same name is created (and re-initialized).
- Static: The variable is not killed when exiting scope. If the scope is re-entered, the same variable (with its previous value) is used.
- Register: Like automatic, but hints to compiler to allocate a register. Few modern compilers use this (thus, not recommended to use).
- Extern: Like a definition (not a declaration). Needed by compiler to generate code interacting with external entities (*e.g.*, variables, functions).
- C++ adds additional storage classes

▸ See here for more good information

Scopes

Program	Scope
`int n1;`	file
`void foo(double x);`	file
`int main() {`	
`  int n2;`	function
`  ...`	
`  while (...) {`	
`    int n3;`	block
`    ...`	
`  };`	
`int foo() {`	
`  int n4;`	function
`  ...`	
`};`	

(also, variables declared in other files have global scope)

⚠ Don't confuse scope and lifetime!

Default Storage Classes in C/C++

- Variables in block or function scope: automatic.
- Variables in file or global scope: static.
- Functions: extern. What we called prototypes are actually just function definitions with extern storage class!

Statics and automatics are often [imprecisely] called global and local variables.
We seldom need to use non-defaults, but it is occasionally essential. Most common uses:

- Declaring a persistent variable (static) inside a function, so the function's behavior depends on its state (*i.e.*, prior calls)
- Declaring a persistent variable (static) to store some state variable, where a global variable is not appropriate

Example

```
int foo1() {
  static int n = 0;
  n++;
  return n;
};
int main() {
  int x=0; int y=0;
  while (1) {
    x = foo1();
    y = foo2();
    cout << x << " " << y << " ";
  };
}
```

```
int foo2() {
  auto int n = 0;
  n++;
  return n;
};
```

same as default

infinite loop

? What is the output?

Example

```
int foo1() {
  static int n = 0;
  n++;
  return n;
};
int main() {
  int x=0; int y=0;
  while (1) {
    x = foo1();
    y = foo2();
    cout << x << " " << y << " ";
  };
}
```

```
int foo2() {
  auto int n = 0;
  n++;
  return n;
};
```

same as default

infinite loop

? What is the output?

! 1 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 9 1 10 1 ...

i.e., x is incremented on each iteration, y is always 1

Typical Program Organization

```
#include <iostream>          Setup section used by compiler
#include ...                 to generate code. No assembly code
using namespace;
typedef vector<string> StringVector;   User defined types
static int someGlobal;                 But avoid globals!
extern void initData(ifstream & inf,   Prototypes
                     StringVector & sv);
extern StringVector doStuff(StringVector & sv, int & stats);
extern void dispData(StringVector & sv, int n);
int main() {                           Body of main
  auto StringVector lotsOfStrings, processedStrings;
  ifstream din ("data.txt");
  initData(din, lotsOfStrings);
  processedStrings = doStuff(lotsOfStrings, someGlobal);
  dispData(processedStrings, someGlobal);
  return 0;
}
void initData(ifstream & inf, StringVector sv) { ... };
StringVector doStuff(StringVector & sv, int & stats) {...};
void dispData(ifstream & inf, StringVector sv) { ... };
```

⚠All storage classes above were defaults, and could have been omitted!