

## ELEC3730 Embedded Systems Lecture 5: Memory Management

- Motivation
- Objects
- Scope
- Static versus Automatic Objects




---

---

---

---

---

---

---

---

### Memory Management: Motivation/Strategies

- Multi-tasking:
  - Multiple threads/processes sharing same RAM
- Portability/Cost concerns:
  - Limited RAM
- Implications:
  - Potential for data of one task to be accidentally modified by another;
  - Potential for memory requirements to exceed available RAM
- Strategies:
  - Minimize amount of global data;
  - Limit scope of variables;
  - Dynamic memory allocation.

---

---

---

---

---

---

---

---

### C Object: A region of memory with following attributes

Attribute	Description
Type	char, int, unsigned int, etc. <i>(also implies size, range, and resolution)</i>
Name	The identifier used to access the object.
Value	The data held within the object.
Address	The location in memory where the object resides.
Scope	That part of the source code where the object's name is recognized.
Lifetime	A notion of when the object is created and destroyed, and thus when it is available for use.

---

---

---

---

---

---

---

---

### Global Objects - Scope

- Must be declared outside of all functions.
- Scope is from point of declaration to end of file.

Function names are inherently global since no function is ever declared inside another.

---

---

---

---

---

---

---

---

### Scope: Global vs Local Objects

```
int a ;  
  
void f ( )  
{  
  int b ;  
  ...  
  b = ... .  
  ...  
}
```

Containing block for 'b'.

A global that can be used by any function.

A temporary local only used in function "f".

---

---

---

---

---

---

---

---

### Limiting Scope: Two Different Objects - Same Identifier

```
#include <stdio.h>  
void main ()  
{  
  int datum = 1 ;  
  printf("datum=%d\n", datum)  
  if ( ... )  
  {  
    int datum = 2 ;  
    printf("datum=%d\n", datum) ;  
  }  
  printf("datum=%d\n", datum) ;  
}
```

Scope of the 1st object begins at the point of declaration and ends at the closing brace of the function.

Scope of the 2nd object. Impossible for statements outside this block to damage this object. Hides the 1st object declared with the same name in outer block.

---

---

---

---

---

---

---

---

### Accessing External Variables (Defined in Other Files)

```
/* Define and initialize in only one file */
int a = 123 ;
void f1()
{
  .....
}

/* Separate file requiring access ... */
extern int a ; /* no initialization here! */
void f2()
{
  a = ..... /* external reference */
}
```

---

---

---

---

---

---

---

---

### Static keyword: restricted access to global objects

- A variable declared with the `static` keyword is local to the file:

```
file1.c      file2.c
#include ...  #include ...
int x ;      static int y ;
static int foo()
{
  .....
}

int bar()
{
  .....
}
```

Accessible from within other files!

Accessible only within this file.

---

---

---

---

---

---

---

---

### Accessing External Functions (Defined in Other Files)

```
double foo(char a, int x)
{
  .....
}

extern double foo(char, int);

void bar()
{
  y = foo(c, 32);
}
```

"extern" is not required (default).

Function *prototype* tells compiler how to use foo.

Moving extern declaration of function inside function bar does NOT reduce its scope!

---

---

---

---

---

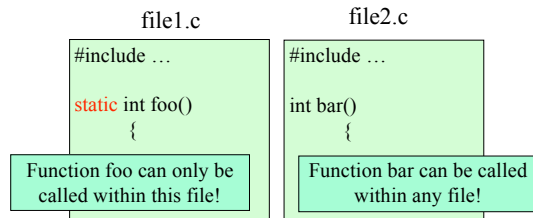
---

---

---

## Restricting Access to Functions

- A function declared with the `static` keyword is local to the file:



---

---

---

---

---

---

---

---

---

---

## Misuse of Globals

### Dangerous

```
unsigned total = 0 ; /* These objects are */  
unsigned count = 0 ; /* global to the file! */  
  
void EnterScore(unsigned score)  
{  
    total += score ;  
    count++ ;  
    printf("average score = %u\n",  
           total / count) ;  
}
```

### Protected

```
void EnterScore(unsigned score)  
{  
    static unsigned total = 0 ; /* These are local */  
    static unsigned count = 0 ; /* to this function */  
  
    total += score ;  
    count++ ;  
    printf("average score = %u\n",  
           total / count) ;  
}
```

If a persistent object is needed by a single function, declare it local to that function.

---

---

---

---

---

---

---

---

---

---

## Memory Allocation in C - Lifetime and Ownership

Allocation	Lifetime	Location	Memory reclaimed by
Static	Entire program execution.		Operating system
Automatic	From entry to exit of containing function.	Stack	Program (implicit)
Dynamic	From malloc() to free().	Heap	Programmer (explicit)

Choice of allocation method determines *how* an object is created and destroyed, and thus who is responsible for memory management.

---

---

---

---

---

---

---

---

---

---

## Memory Allocation in C - Creation, Initialisation and Lifetime

Allocation	Created	Initialized	Destroyed
Static	Once: When the program is first loaded into memory.	Once: Just before the program starts to run.	Once: When the program stops.
Automatic	Each time the program enters the function in which it is declared.	Each time the program enters its block (if specified in declaration).	Each time the program exits the block.
Dynamic	By calling the library function malloc.	By writing statements that modify its content.	By calling the library function free.

---

---

---

---

---

---

---

---

---

---

## Static Objects

- Static objects are allocated **once** when program is first loaded into memory.
- Location in memory never changes during execution of the program.
- Static objects not destroyed until the program terminates.
- **Advantage:** Persistence of values.
  - Unlike objects that use automatic allocation, a value stored in a static object will remain from one execution of a function to another.
- **Disadvantage:** Inability to reclaim memory.
  - Extensive use of static objects can result in programs that require lots of memory.
- **Every global object is a static object!**
  - Avoiding scope restrictions by declaring **all** objects global ultimately produces very "fat" programs.
- **Every static object is not necessarily global!**
  - Using global to get persistence isn't necessary - just add the keyword `static` to the local declaration.

---

---

---

---

---

---

---

---

---

---

## The static Keyword

- **Inside a Function**
  - Changes the allocation method from auto (default) to static.
- **Outside of Functions**
  - Has **no** effect on allocation (remains static);
  - Makes the identifier inaccessible from other files.

---

---

---

---

---

---

---

---

---

---

## Automatic Objects

- Automatic objects are allocated on every entry to their containing function.
- Automatic objects are initialized on every entry to their containing block.
- Physical memory location is fixed during execution of the function.
- Physical memory released when the function is exited.
  - Functions may return the value of an automatic variable, but not its address.
- **Advantage:**
  - Memory conservation through reuse.
  - The program manages the sharing of this memory resource automatically.
- **Disadvantage:**
  - Lack of persistence.

---

---

---

---

---

---

---

---

---

---

## Object Creation (Auto vs. Static)

```
#include <stdio.h>

void f1()
{
    auto int a ;
    static int s ;

    printf(" &auto = %08X\n", &a) ;
    printf("&static = %08X\n", &s) ;
}

void f2() { auto int x; f1() ; }

int main() { f1(); f2(); f1(); return 0 ; }
```

### Program Output:

```
&auto = 0008E8BC
&static = 0000BA28

&auto = 0008E89C
&static = 0000BA28

&auto = 0008E8BC
&static = 0000BA28
```

---

---

---

---

---

---

---

---

---

---

## Object Initialization (Auto vs. Static)

```
#include <stdio.h>

int f1() { auto int a ; a = 0; return ++a ; }
int f2() { static int s ; s = 0; return ++s ; }

int f3() { auto int a = 0 ; return ++a ; }
int f4() { static int s = 0 ; return ++s ; }

int main()
{
    printf("f1(): %d %d %d\n", f1(), f1(), f1() ;
    printf("f2(): %d %d %d\n", f2(), f2(), f2() ;
    printf("f3(): %d %d %d\n", f3(), f3(), f3() ;
    printf("f4(): %d %d %d\n", f4(), f4(), f4() ;
    return 0 ;
}
```

### Program Output:

```
f1(): 1 1 1
f2(): 1 1 1
f3(): 1 1 1
f4(): 3 2 1
```

---

---

---

---

---

---

---

---

---

---

**Object Destruction**  
**(Auto vs. Static)**

**Program Output:**

```
Address=0008E88C, Contents=12345  
Address=0008E88C, Contents=12345  
Address=0000AECC, Contents=12345  
Address=0000AECC, Contents=583820
```

```
int *f1(){static int s = 12345 ; return &s ;}  
int *f2(){ auto int a = 12345 ; return &a ;}  
  
void Demo(int *(*f)()) {  
    int *p = (*f)() ;  
    printf("Address=%08X, Contents=%d\n", p, *p) ;  
    printf("Address=%08X, Contents=%d\n", p, *p) ;  
    printf("\n") ;  
}  
  
int main() { Demo(f1) ; Demo(f2) ; return 0 ;}
```

---

---

---

---

---

---

---

---