NWEN 241
Storage Classes

Winston Seah

School of Engineering and Computer Science
Victoria University of Wellington

Victoria
UNIVERSITY OF WELLINGTON
*Te Whare Wānanga
o te Ūpoko o te Ika a Māui*
CAPITAL CITY UNIVERSITY

Variable Storage Class

C storage classes are:

- **`auto`**
- **`static`**
- **`register`**
- **`extern`**

Storage class of a variable determines its:

- ***Scope*** attribute – where is a variable visible
- ***Lifetime*** attribute – how long does a variable exists

Scope and Lifetime

- Lifetime/storage attributes can be:
  - **`static`** variables are allocated memory when program starts;
  - **`auto`** – automatic variables are allocated memory when execution enters the block that contains it;
  - **`register`** – reside in CPU's high speed memory
- Scope attributes can be:
  - **`local`** – **`v`** is only visible inside the current, innermost scope, independent of storage/lifetime attribute; e.g. there are **`local static`** variables in C
  - **`global`** – **`v`** is visible in the whole compilation unit, from the line of declaration to the end of file
  - **`external`** – **`v`** is visible in all compilation units; **`static`**

`auto` Storage Class

- **`auto`** is the default storage class for a variable defined inside a function body or a statement block
- **`auto`** prefix is optional; i.e. any locally declared variable is automatically **`auto`**, unless specifically defined to be static

Example:

```
{
    auto double x; /* Same as: double x */
    int num;  /* Same as: auto int num; */
    . . .
}
```

`auto` Storage Class

- *Automatic variables* may *only* be declared *within* functions and compound statements (*blocks*)
  - Storage *allocated* when function or block is entered
  - Storage is *released* when function returns or block exits
- Parameters and result are similar to automatic variables
  - Storage is *allocated* and *initialized* by *caller* of function
  - Storage is *released* after function *returns* to caller.
- Variables declared within a function or compound statement are visible *only* from the point of declaration to the end of that function or compound statement.

auto Storage Class example

```
int func (float a, int b) {
  int i;
  double g;

  for (i = 0; i < b; i++) {
    double h = i*g;
    // loop body – may access a, b, i, g, h
  } // end of for(i…) loop
  // func body – may access a, b, i, g
} // end of int func( … )
```

i is visible from this point to end of **func**

g is visible from this point to end of **func**

h is only visible from this point to end of loop!

`auto` Storage Class example

```
int func (float a, int b) {
  int i;
  double g;

  for (i = 0; i < b; i++) {
    double h = i*g;
    // loop body – may access a, b, i, g, h
  } // end of for(i…)loop
  // func body – may access a, b, i, g
} // end of int func( … )
```

Storage for **`i`** created.

Storage for **`g`** created

Storage for **`h`** created.

Storage for **`h`** released.

Storage for **`g`** released.

Storage for **`i`** released.

`auto` Storage Class initialization

- If an `auto` variable is defined but not initialized:
  - Variable has an unknown value when control enters its containing block
- If an `auto` variable is defined and initialized at the same time:
  - Variable is re-initialized each time control enters its containing block
- An `auto` variable's scope is limited to its containing block (i.e., it is `local` to the block)

`static` Storage Class

- Storage for a **`static`** variable:
  - Is allocated when execution begins
  - Exists for as long as the program is running
- A **`static`** variable may be defined either inside or outside a function’s body.
- The **`static`** prefix must be included

  Example:

```
static double seed;
```

`static` Storage Class initialization

- If a **`static`** variable is defined but not initialized:
  - Is set to zero (0) once, when storage is allocated
- If a **`static`** variable is simultaneously defined and initialized:
  - Is initialized once, when storage is allocated
- A **`static`** variable defined inside a function body is visible only in its containing block
- A **`static`** variable defined outside a function body is visible to all blocks which follow it in the current compilation units
- If you wish it to be visible in other compilation units, it must be declared **`extern`**

`static` Storage Class example

```
#include <stdio.h>

void strange( int x )
{ // strange function
    static int y;  /* Persistent */
    if ( x == 0 )
        printf( "%d\n", y );
    else if ( x == 1 )
        y = 100;
    else if ( x == 2 )
        y++;
} //end of strange function

int main (void)
{ // main
    strange(1);  /* Set y in strange to 100 */
    strange(0);  /* Will display 100       */
    strange(2);  /* Increment y in strange  */
    strange(0);  /* Will display 101       */
    return 0;
} // end main
```

Program output

```
winston$ gcc -o strange strange.c
winston$ ./strange
100
101
winston$
```

`register` Storage Class

- The fastest storage resides within the CPU itself in high-speed memory cells called *registers*
- The programmer can request the compiler to use a CPU register for storage

  Example:

  ```
  register int k;
  ```
- The compiler can ignore the request, in which case the storage class defaults to `auto`
- Some machines, e.g. stack architectures, have no user visible register

extern Storage Class (single source file)

- **extern** is the default storage class for a variable defined outside a function’s body
- Storage for an **extern** variable:
  - Is allocated when execution begins
  - Exists for as long as the program is running
- If an **extern** variable is defined but not initialized:
  - Set to zero (0) once, when storage is allocated
- If an **extern** variable is defined and initialized:
  - Initialized once, when storage is allocated
- An **extern** variable is visible in all functions that follow its definition (i.e., it is **global**)

extern Storage Class example

```
#include <stdio.h>

float x = 1.5; /* Definition - extern class - global */

void show (void)
{
   printf("%f\n", x); /* Access global x */
}

int main (void)
{
   printf("%f\n", x); /* Access global x */

   show();

   return 0;
}
```

Storage Classes in Multiple Files

- Functions stored in a single source file can be divided into separate source files.

- Variables defined in one source file can be accessed from other source files via the **`extern`** storage class.

- An **`extern`** variable can be defined in **one file** only. However, it may be declared from other files.

Storage Classes in Multiple Files

- An **`extern`** variable is defined exactly once in a file by placing it outside all blocks.
- If an **`extern`** variable is not initialized at definition time
  → **`extern`** prefix must be omitted
- If an **`extern`** variable is initialized at definition time
  → **`extern`** prefix is optional
- An **`extern`** variable is declared in another file by using the **`extern`** prefix.

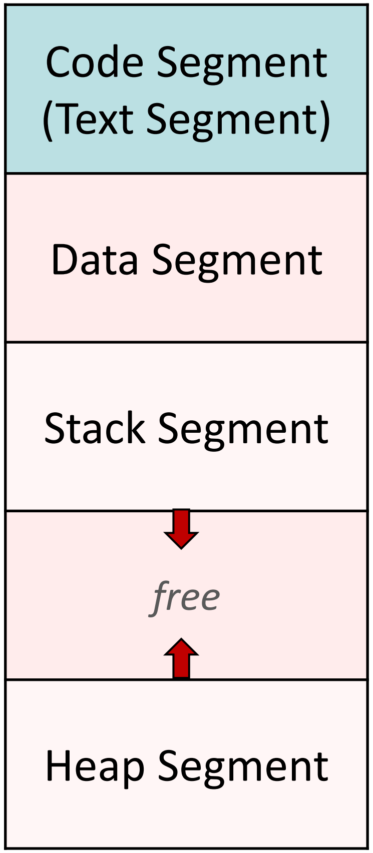
  Example:

  ```
  extern int k;
  ```

Memory Layout of a Program

Memory space for program code includes space for machine language code and data

- **Text / Code Segment**
  - Contains program's machine code
- Data spread over:
  - **Data Segment** – Fixed space for global variables and constants
  - **Stack Segment** – For temporary data, e.g. local variables in a function; expands / shrinks as program runs
  - **Heap Segment** – For dynamically allocated memory; expands / shrinks as program runs



Memory Storage Layout

Where are auto, static, and extern variables stored?

Contains the program's machine code	Code Segment (Text Segment)
Contains static data (e.g., **`static`** class, **`extern`** globals)	Data Segment
Contains temporary data (e.g., **`auto`** class)	Stack Segment
Unallocated memory that the stack and heap can use	⬇ *free* ⬆
Contains dynamically allocated data – later…	Heap Segment