Pointers and Storage Classes

COM S 113

February 8, 1999

Announcements

Assignment 2 can be turned in Tuesday; office hours today 2:00–3:30 in Upson 5162

Assignment 3 (short!) available, due Friday

Read Ch. 8 in *C by Dissection* or K&R 5.1–5.9

Pointers and `const`

```
const int a;   /* a is a const int */
const int *b;  /* b is a pointer to a const int */
int * const c; /* c is a const pointer to int */
const int * const d; /* d is a const pointer
                                to a const int */
```

Pointers to `void`

One pointer can be assigned to another only if both have same type or one is pointer to `void`

`void *` is used as a generic pointer type

`malloc()` returns a pointer to `void`, so we can assign the result to any pointer type without a cast

Examples of Pointers to `void`

```
int *p;  float *q;  void *v;
```

```
/* Legal */                       /* Illegal */
p = 0;                            p = 1;
p = (int *) 1;                    v = 1;
p = v = q;                        p = q;
p = (int *) q;
p = malloc(4 * sizeof(int));
```

Example of Call-by-Reference

```
void swap(int *p, int *q) {
  int tmp = *p; *p = *q; *q = tmp;
}

int main() {
  int a=3, b=7;
  swap(&a, &b);
  return 0;
}
```

Storage Classes

Every variable and function has a *type* and a *storage class*

Four storage classes: `auto`, `extern`, `register`, and `static`

Storage Class `auto`

Variables within functions or blocks default to automatic, but storage class can be given explicitly:

```
auto int a, b, c;
```

Memory allocated upon entering block, released at exit, so values aren't kept between invocations

Storage Class `static` (first use)

When applied to variables defined within a block, local variables retain their values between invocations

```
void printletter(void) {
  static int parity; /* initially 0 */
  putchar(parity ? 'A' : 'B');
  parity = (parity + 1) % 2;
}
```

Using `static` Variables for Debugging

```
...
{
  static int cnt = 1;
  printf("On %dth iteration, d has value %d.\n",
         cnt, d);
}
...
```

Storage Class `extern`

Variables declared outside functions, and all functions themselves, have external storage class

`extern` tells the compiler to look for a variable elsewhere, either in the same file or in another file

Example of External Variables

```
#include <stdio.h>
int a = 1, b = 2, c = 3;  /* global variables */
int f(void);              /* function prototype */

int main() {
  int b, c;
  a = b = c = 4;
  printf("%3d\n", f());
  printf("%3d%3d%3d\n", a, b, c); }
```

Useful Example of `extern`

In file `file1.c`:

```
int a = 1, b = 2, c = 3;

int f(void);

int main() { printf("%3d\n%3d%3d%3d\n", f(), a, b, c); }
```

In file `file2.c`:

```
int f(void) { extern int a;
  int b, c;
  a = b = c = 4;  return a + b + c; }
```

Storage Class `register`

Advises (but doesn't require) compiler to store value in CPU register rather than in memory

Defaults to `auto` if compiler decides otherwise

Purpose is to speed program execution by keeping *very frequently* accessed variables (loop counters) immediately available

Storage Class `register` (continued)

Was important when compilers weren't as smart about register allocation; many compilers now ignore `register`

Because `register` variables not necessarily stored in memory, can't take the address of such a variable

```
register int i; /* could be written: register i */
for (i = 0; i < LIMIT; i++)
    ...            /* illegal to refer to &i */
```

Storage Class `static` (second use)

When applied to external declarations (of functions or variables), scope is restricted to current file

Functions in other files can't access external **`static`** variables, even if they attempt to use the **`extern`** storage class keyword

Good way to implement information hiding in C, like private variables and methods in Java, but limited

Example of External `static` Variables

Consider implementation of a stack with operations `push(i)`, `pop()`, `empty()`, and `full()`

We'll implement with integer array `s`, using variable `next` to point to next free element, both declared `static`, in file `stack.c`

Example of External `static` Variables (continued)

```
#include "stack.h"
static int s[MAX_SIZE], next = 0;


void push(int i) { s[next++] = i; }
int pop(void) { return s[--next]; }
int empty(void) { return next == 0; }
int full(void) { return next == MAX_SIZE; }
```

Review of Memory Allocation

```
#include <stdlib.h>
int main() {
  int *a;
  a = malloc(sizeof(int));
  *a = 3;
  printf("a is an int pointer with value %p\n", a);
  printf("a points to an int with value %d\n", *a);
  free(a);                                              }
```

Allocating One-Dimensional Arrays

```
#include <stdlib.h>

int main() {

  int *a, i, n = 100;

  a = malloc(n * sizeof(int));

  for (i = 0; i < n; i++)

    a[i] = i;

  free(a);

}
```

Traversing an Array with Pointers

```
#define N 100
int sumarray(const int a[N]) {
  int *p, sum = 0;

  for (p = a; p < &a[N]; p++)
    sum += *p;
  return sum;
}
```