Announcements

- Program #1B
  - Due on Thursday
- Midterm #1
  - One week from today !!
  - 6:00pm
  - in ARM 0131
  - material covered – through today's lecture
  - Next Monday's lab – is review for the exam
- Reading
  - Chapter 8 and 9 (Today)
    - skip 8.1.4
    - skip 9.3
  - Chapter 6 (Thursday)

Pointer Arithmetic

- Pointers are first class types
  - Arithmetic applies to pointers just like ints
  - This is useful, but can get tricky
- Adding pointer and int
  - Adds n of the object size indicated by the type of p
    - If objects are 100 bytes each
    - p+1 is 100 bytes later in memory than indicated by p
  - Example:

```
int x = 5;
int a[10];
int *p = a;          /* same as    int *p = &a[0] */
*(p + 3) = 42;     /* same as    a[3]  = 42       */
```

- [] is really shorthand for pointer addition
  - a[n] is really the same as *(a + n)

Initializing Arrays

- Syntax: type variableName[size] = { list of values };
- Example:
  - int vector[5] = { 1, 2, 67, 83, 4 };
- Global vs. function variables
  - Global variables are initialized only once
  - Function variables the initialization once for each function invocation
- Matching Number of Elements
  - Too many elements: Error
  - Too few elements: Rest left not initialized
- Automatic Array Sizing (leaving the size out of the declaration)
  - int vector [] = { 1, 2, 67, 83, 4 };
  - Character Arrays (short hand notation)
    - char message[]= {‘T’,’h’,’i’,’s’,’ ‘,…}
    - char message[] = "This is the string";

Multidimensional Arrays

- int matrix[3][4];
  - C uses arrays of arrays to handle multidimensional arrays
  - This is an array of 12 elements
- Storage Order
  - row major order
  - right most dimension has adjacent elements in consecutive memory

- Subscripts
  - matrix[1][3] is the 4th element in the 2nd row
  - matrix[1][5] is 2 beyond that

Pointers To Arrays

- int vector[10];
  - int *vp1=vector;
  - int *vp2 = &vector[0];
  - vp1 and vp2 point to the 0^{th} element in the array of integers
- int matrix[3][10];
  - int *mp = matrix;
  - ERROR: mp points to an integer, but matrix is an array of arrays
- Can also treat multi-dimensional arrays as single
  - int *pi = &matrix[0][0]
  - int *pi = matrix[0]
  - advancing pi (pi++) will navigate entire array of arrays

Passing Multidimensional Arrays

- Passed as base address of first element
  - compiler needs help to know what dimension are
- Example:

```
int matrix[3][10];
func2(matrix);

void func2(int (*mat)[10]);     /* 10 elements in 2nd array */
void func2(int mat[][10]);      /* same as above */
```

- Rule
  - need to define size for all but first array in multidimensional arrays

Initialization

- Two Possible Approaches
- int matrix[2][3] = { 100, 101, 201, 202, 301, 302 };
  - as one long array
- int matrix[2][3] = { { 100, 101, 201},
  { 202 , 301, 302 } };
  - as array of arrays
- Second approach is strongly preferred

Arrays of Pointers

int *api[10];

- An array of 10 pointers to integers;

```
char *keyword[] = {
    "Load",
    "Store",
    "Negate"
  };
```

- An array of three strings
- space allocated as indicated by initializer
- It is an array of three pointers
  - the first points to an array of char which is 5 long
  - the second points to an array of char which is 6 long
  - and the third points to an array of char which is 7 lon

Standard Type Definitions

- Several Types are defined in C's libraries
  - Defined in <stddef.h>
  - size_t - unsigned integer value (often int or long)
  - ptrdiff_t - signed value (used when subtracting pointers)
- Used by several standard routines
  - sizeof(<type>) returns a size_t for example

Strings

- Zero or more characters followed by null char ‘\0’
  - also called NUL
  - not counted as part of string
  - string.h defines prototypes for string routines
- Copying Strings
  - size_t strlen(char const *str);
    - returns count of characters in str
    - up to but not including the null character
  - char *strncpy(char *dst, char const *src, size_t len);
    - copy src to dst
    - copy until ‘\0’ in src or at most len characters
    - pad extra characters will ‘\0’
    - Safety tip: dst[len-1] = '\0'; to force termination of new string
  - char *strncat(char *dst, char const *src, size_t len);
    - append src onto the end of dst
    - always appends NUL to end of dst string

String Functions

- Comparison
  - int strncmp(char const *s1, char const s2, size_t len);
    - returns 0 if string equal up to len
    - returns a negative value if s1 is less than s2
    - returns a positive value if s1 is greater than s2
- Searching
  - char *strchr(char const *str, int ch);
  - char *strrchr(char const *str, int ch);
    - finds the first occurance of ch in str
    - strrchr finds the last occurance
    - returns NULL if not found
  - char *strstr(char const *s1, char const *s2);
    - find the first occurrence of s2 in s1

String Functions Examples

```
char string[] = "this is a test string";
char *ans;
int length;

length = strlen(string);          /* returns 21 */
ans = strchr(string, 'h');        /* returns string + 1 */
ans = strrchr(string, 't');       /* returns string + 16 */
ans = strstr(string, "test");     /* returns string + 10 */
```

Character Functions

- Prototypes in ctype.h
- Classifying characters
  - parameter is int, but it's a character
  - true (non-zero) if it is true or 0 if it is false
  - int isspace(int ch);
    - returns true if ch ' ', '\n', '\t', form feed, or carriage return
  - int isdigit(int ch);
    - returns true if its 0 through 9
  - int islower(int ch); isupper(int ch);
    - return true if it's a-z, A-Z
  - isalpha(int ch);
    - returns true if it's a-z or A-Z
- Transformation
  - int toupper(int ch); int tolower(int ch);
    - converts to upper/lower case

Blocks of Memory

- C supports operations on blocks of memory
  - might be an array, a struct, an array of structs, etc.
- Copying
  - void *memcpy(void *dest, const void *src, size_t len);
    - copies memory pointed to by src to memory pointed to by dest
    - copies len bytes of memory
    - returns the location of src
  - void *memmove(void *dest, const void *src, size_t len);
    - copies memory pointed to by src to memory pointed to by dest
    - copies len bytes of memory
    - returns the location of src
    - handles the case if dest and src overlap

Blocks of Memory

- Testing and Setting
  - void *memset(void *dest, int ch, size_t len);
    - fills dest with len copies of character ch
  - int memcmp(void *a, const void *b, size_t len);
    - compares a to b
      - unsigned character by unsigned character
    - stops after len bytes
    - returns 0 if equal
    - returns a negative if a is less than b (on first different char)
    - returns a positive if b is greater than b (on first different char)
    - **Typo in book, pg. 257 return type is int for memcmp**

Examples of memory functions

```
char buffer[25];
memset(buffer, '\0', sizeof(buffer));
```

```
struct foo {
    int a;
    char b;
} dest, src;

memcpy(&dest, &src, sizeof(dest));
```