Strings
Two-dimensional Arrays

- `int x[3][4];`
- `x[i][j] ≡ * (& x[0][0] + 4*i + j )`

```c
int sum(int x[][4])
{
    int i, j, sum = 0;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            sum += x[i][j];
    return sum;
}
```

<table>
<thead>
<tr>
<th>x[0][0]</th>
<th>x[0][1]</th>
<th>x[0][2]</th>
<th>x[0][3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[1][0]</td>
<td>x[1][1]</td>
<td>x[1][2]</td>
<td>x[1][3]</td>
</tr>
<tr>
<td>x[2][0]</td>
<td>x[2][1]</td>
<td>x[2][2]</td>
<td>x[2][3]</td>
</tr>
</tbody>
</table>
Three-dimensional Arrays

- int x[3][4][5];
- x[i][j][k] ≡ *(& x[0][0][0] + 4*5*i + 5*j + k )

```c
int sum(int x[][4][5])
{
    int i, j, sum = 0;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            for (k = 0; k < 5; k++)
                sum += x[i][j][k];
    return sum;
}
```
Initialization of Multidimensional Arrays

- An array of storage class automatic is not explicitly initialized
- All static and external arrays are initialized to zero by default

```c
int x[2][3] = { 1, 2, 3, 4, 5, 6 };  
int x[2][3] = { {1, 2, 3}, {4, 5, 6} };  
int x[][3] = { {1, 2, 3}, {4, 5, 6} };  
int x[2][2][3] = {{{1,1,0}, {2,0,0}}, {{3,0,0}, {4,4,0}}};  
int x[][2][3] = {{{1,1}, {2}}, {{3}, {4,4}}};  
int x[2][2][3] = { 0 };  
```
Strings

- Strings are one-dimensional arrays of type char
  - A string in C is terminated by the null character ‘\0’ (end-of-string sentinel)
  - The size of a string must include ‘\0’
- String constants are written between double quotes
  - Treated as a pointer
  - The value is the base address of the string

```c
char *p = "abc";
printf("%s %s\n", p, p+1);
"abc"[2]
*("abc" + 2)
char s[] = "abc";
char s[] = { 'a', 'b', 'c', '\0' };
```
Counting the Number of Words in a String

```c
#include <ctype.h>
int wordCount(const char *s)
{
    int cnt = 0;
    while (*s != '\0') {
        while (isspace(*s))
            s++;
        if (*s != '\0') {
            cnt++;
            while ( !isspace(*s) && *s != '\0')
                s++;
        }
    }
    return cnt;
}
```
Some String-Handling Functions in the Standard Library

- `char *strcat(char *s1, const char *s2);`
  - Concatenates two strings `s1` and `s2` and puts the result in `s1`

- `int strcmp(const char *s1, const char *s2);`
  - An integer is returned that is less than, equal to, or greater than zero, depending on whether `s1` is lexicographically less than, equal to, or greater than `s2`
Some String-Handling Functions in the Standard Library

- `char *strcpy(char *s1, const char *s2);`
  - The characters in `s2` are copied into `s1` until `\0` is moved
  - The pointer `s1` is returned

- `size_t strlen(const char *s);`
  - A count of the number of characters before `\0` is returned
strlen()

size_t strlen(const char *s)
{
    size_t n;

    for ( n = 0; *s != '\0'; ++s )
        ++n;

    return n;
}
`strcpy()`

```c
char *strcpy(char *s1,
            register const char *s2)
{
    register char *p = s1;

    while ( *p++ = *s2++ )
        ;

    return s1;
}
```
strcat()

char *strcat(char *s1,
            register const char *s2)
{
    register char *p = s1;

    while ( *p )
        ++p;
    while ( *p++ = *s2++ )
        ;
    return s1;
}
Arguments to main()

- To communicate with the OS
- argc: the number of command line arguments
- argv: an array of strings
  - The strings are the words that make up the command line
  - argv[0] contains the name of the command itself

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  printf("argc = %d\n", argc);
  for (i = 0; i < argc; i++)
    printf("argv[%d] = %s\n", i, argv[i]);
  return 0;
}
```
Ragged Arrays

- An array of pointers whose elements are used to point to arrays of varying sizes
- char x[2][7] = {“abc”, “abcde” };

```
x
  a b c \0
  a b c d e \0
```

- char *p[2] = {“abc”, “abcde” };

```
P
  a b c \0
  a b c d e \0
```
void mergeSort( int a[], int n )
{
    int i, j, m, *w;
    for ( m = 1; m < n; m *= 2 );
    if ( n < m ) {
        printf("%s%s", "error: array size not a power of 2!\n");
        exit( 1 );
    }
    w = calloc( n, sizeof(int) );
    assert( w != NULL );
    for( i = 1; i < n; i*=2 ) {
        for( j = 0; j < n - i; j += 2*i )
            merge(a+j, a+j+i, w+j, i, i);
        for( j = 0; j < n; j++)
            a[j] = w[j];
    printIntArray( a, n );
}
free( w );
}
void merge( int a[], int b[], int c[], int m, int n )
{
    int i = 0, j = 0, k = 0;

    while ( ( i < m ) && ( j < n ) )
        if ( a[i] < b[j] )
            c[k++] = a[i++];
        else
            c[k++] = b[j++];

    while ( i < m )
        c[k++] = a[i++];

    while ( j < n )
        c[k++] = b[j++];
}

Merge Sort (contd.)
Merge Sort (contd.)

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>33</td>
<td>87</td>
<td>-7</td>
<td>-24</td>
<td>50</td>
<td>28</td>
<td>34</td>
<td>-45</td>
<td>82</td>
<td>36</td>
<td>-4</td>
</tr>
<tr>
<td>-3</td>
<td>23</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>33</td>
<td>-7</td>
<td>87</td>
<td>-24</td>
<td>50</td>
<td>28</td>
<td>34</td>
<td>-45</td>
<td>82</td>
<td>-4</td>
<td>36</td>
</tr>
<tr>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>23</td>
<td>-7</td>
<td>11</td>
<td>33</td>
<td>87</td>
<td>-24</td>
<td>28</td>
<td>34</td>
<td>50</td>
<td>-45</td>
<td>-4</td>
<td>36</td>
<td>82</td>
</tr>
<tr>
<td>-7</td>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>23</td>
<td>33</td>
<td>87</td>
<td>-45</td>
<td>-24</td>
<td>-4</td>
<td>28</td>
<td>34</td>
<td>36</td>
<td>50</td>
<td>82</td>
</tr>
<tr>
<td>-45</td>
<td>-24</td>
<td>-7</td>
<td>-4</td>
<td>-3</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>23</td>
<td>28</td>
<td>33</td>
<td>34</td>
<td>36</td>
<td>50</td>
<td>82</td>
<td>87</td>
</tr>
</tbody>
</table>
Exercises

- Write a program that sorts the words in a text file lexicographically using arrays of pointers