Announcements

• Homework assignment due Tuesday (1 day extension)
• Read Chapter 4 in the book
• Midterm on Feb 6
Lecture 10: Overview

- Review of Arrays
  - Introduction
  - Indexing
  - Initialization
  - Multi-dimensional arrays
  - Operator associativity and precedence
Arrays

- Array data type in C
  - Composite data type
    - Type is an array of a sub-type (e.g. array of int)
  - Fixed number of elements
    - Array size is fixed at time of definition (e.g. 100 elements)
  - Element access by index (aka. subscript)
    - Element-access operator: array[index]  (e.g. A[42])
- Example:

```c
int A[10]; /* array of ten integers */
A[0] = 42;  /* access to elements */
A[1] = 100;
```
Arrays

- Array Indexing
  - Start counting from 0
  - First element has index 0
  - Last element has index Size-1
- Example:

```c
int A[10];
A[0] = 42;
A[1] = 100;
/* ... */
```
Arrays

- Array Indexing
  - Array indices are *not* checked by the compiler!
  - Accessing an array with an *index out of range* results in unpredictable behavior!

- Example:

```c
int A[10];
int i;

A[-1] = 42; /* INVALID ACCESS! */
for(i=0; i<=10; i++)
  /* INVALID LOOP RANGE! */
  { printf("%d, ", A[i]);
  }
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in {} 
- Example:

```c
int A[10] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99};
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in `{ }`
- Example:

```c
int A[ ] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99};
```

- With given initializer list, array size may be omitted
  - automatically determined
Arrays

• Array Initialization
  • Static initialization at time of array definition
  • Initial elements listed in {}

• Example:
  int A[10] = { 1, 2, 3};

• With given initializer list and array size, unlisted elements are zero-initialized
  • array is filled up with zeros
Arrays

- Multi-dimensional Arrays
  - *Array of an array...*
- Example:

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

int i, j;

for(i=0; i<3; i++)
  for(j=0; j<2; j++)
    printf("%d ", M[i][j]);
printf("\n");
```

```
1 3 5 2 4 6
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Passing Arguments to Functions

• Pass by Value
  • only the *current value* is passed as argument
  • the parameter is a *copy* of the argument
  • changes to the parameter *do not* affect the argument

• Pass by Reference
  • a *reference* to the object is passed as argument
  • the parameter is a *reference* to the argument
  • changes to the parameter *do* affect the argument

• In ANSI C, ...
  • ... basic types are passed by value
  • ... arrays are passed by reference
Passing Arguments to Functions

• Example: Pass by Value

```c
void f(int p) {
    printf("p before modification is \%d\n", p);
    p = 42;
    printf("p after modification is \%d\n", p);
}

int main(void) {
    int a = 0;
    printf("a before function call is \%d\n", a);
    f(a);
    printf("a after function call is \%d\n", a);
}
```

a before function call is 0
p before modification is 0
p after modification is 42
a after function call is 0

Changes to the parameter do not affect the argument!
Passing Arguments to Functions

- **Example: Pass by Reference**

```c
void f(int p[2])
{
    printf("p[1] before modification is  %d\n", p[1]);
    p[1] = 42;
    printf("p[1] after modification is   %d\n", p[1]);
}

int main(void)
{
    int a[2] = {0, 0};
    printf("a[1] before function call is %d\n", a[1]);
    f(a);
    printf("a[1] after function call is  %d\n", a[1]);
}
```

```
a[1] before function call is 0
p[1] before modification is 0
p[1] after modification is 42
a[1] after function call is 42
```

Changes to the parameter do affect the argument!
Character Arrays: Strings

• Text is represented by character arrays (aka. *strings*)
  • Strings are null-terminated arrays of characters
  • String output
    • `printf()` conversion: “%s”
• Example:

```c
char s1[] = {'H','e','l','l','o',0};

printf("s1 is %s.\n", s1);

s1 is Hello.
```

```c
<table>
<thead>
<tr>
<th></th>
<th><code>H</code></th>
<th><code>e</code></th>
<th><code>l</code></th>
<th><code>l</code></th>
<th><code>o</code></th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s1</td>
</tr>
<tr>
<td>1</td>
<td><code>H</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td><code>e</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td><code>l</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td><code>l</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><code>o</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: "%s"
- Example:

```c
char s1[] = {'H','e','l','l','o',0};
char s2[] = "Hello";
printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
```

`s1` is Hello.
`s2` is Hello.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
- Strings are null-terminated arrays of characters
- String output
  - `printf()` conversion: "\%s"
- Example:

```c
char s1[] = {'H','e','l','l','o',0};
char s2[] = "Hello"

printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
s1[1] = 'i';
s1[2] = 0;
printf("Modified s1 is %s.\n", s1);
```

s1 is Hello.
s2 is Hello.
Modified s1 is Hi.
Character Arrays: Strings

- Text is represented by character arrays (aka. \textit{strings})
  - Strings are null-terminated arrays of characters
  - String input
    - \texttt{scanf()} conversion: \texttt{``\%Ns''}, where \texttt{N} specifies maximum field width = array size - 1
    - address argument can be \texttt{&string[0]}

```c
char s1[6];
printf("Enter a string: ");
scanf("\%5s", &s1[0]);
printf("s1 is \%s.\n", s1);
```

Enter a string: Test
s1 is Test.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
- Strings are null-terminated arrays of characters
- String input
  - `scanf()` conversion: “%Ns”, where N specifies maximum field width = array size - 1
  - address argument can be `&string[0]`

```
cchar s1[6];
printf("Enter a string: ");
scanf("%5s", s1);
printf("s1 is %s.\n", s1);
```

```
Enter a string: Test
s1 is Test.
```
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - Characters are represented by numeric values
  - ASCII table defines character values 0-127
- Example:

```c
char s1[] = "ABC12";
int i = 0;
while(s1[i])
    { printf("%c = %d\n",s1[i],s1[i]);
      i++; }
```

<table>
<thead>
<tr>
<th>i</th>
<th>(s_1[i])</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>‘A’</td>
</tr>
<tr>
<td>1</td>
<td>‘B’</td>
</tr>
<tr>
<td>2</td>
<td>‘C’</td>
</tr>
<tr>
<td>3</td>
<td>‘1’</td>
</tr>
<tr>
<td>4</td>
<td>‘2’</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

A = 65
B = 66
C = 67
1 = 49
2 = 50
Problem 1

• Search for a string in another string
• Input: a search string and a string to be searched
• Output: location of search string or -1
Problem 2

- Normalize a n-dimensional vector
- Input: vector
- Output: normalized vector
Problem 3

• Compute median, mean, and mode of a list of integers
• Input: list of integers
• Output: median, mean, and mode
Problem 4

• Capitalize a string
• Input: string
• Output: capitalized string
Problem 5

- Compute determinant of a matrix
- Input: matrix
- Output: value of the determinant