# MC140: lecture #23

today's topic:

*assignment #7 hints:*

*passing 2-dimensional arrays to functions*

*labelling printed output*

*searching*



# passing 2-dimensional arrays to functions.

- When you pass one-dimensional arrays as function arguments, you don't need to specify the size of the array, either in the function prototype or in the function header itself.
- example:

```
#include <stdio.h>

void printDice( int dice[] );

int main( void ) {
   int dice[5] = { 6, 3, 4, 5, 2 };
   printDice( dice );
   return( 0 );
} /* end of main() */
```

```
void printDice( int dice[] ) {
    int i;
    for ( i=0; i<5; i++ ) {
        printf( "%d ",dice[i] );
    } /* end for i */
    printf( "\n" );
} /* end of printDice() */
```



# passing 2-dimensional arrays to functions, 2.

- However, when passing two-dimensional arrays as function arguments, you must specify the dimensions both in the function prototype and in the function header.
- example:

```
#include <stdio.h>

void printBoard( int board[3][3] );

int main( void ) {
   int board[3][3] =
            { 1,2,3,4,5,6,7,8,9 };
   printBoard( board );
   return( 0 );
} /* end of main() */
```

```
void printBoard( int board[3][3] ) {
    int i, j;
    for ( i=0; i<3; i++ ) {
        for ( j=0; j<3; j++ ) {
            printf( "%d ",board[i][j] );
        } /* end for j */
        printf( "\n" );
    } /* end for i */
    printf( "\n" );
} /* end of printBoard() */
```



# passing 2-dimensional arrays to functions, 3.

- In the book, they leave the first (leftmost) dimension unspecified and specify the remaining dimension(s).
- example:

```
#include <stdio.h>

void printBoard( int board[][3] );

int main( void ) {
   int board[3][3] =
            { 1,2,3,4,5,6,7,8,9 };
   printBoard( board );
   return( 0 );
} /* end of main() */
```

```
void printBoard( int board[][3] ) {
    int i, j;
    for ( i=0; i<3; i++ ) {
        for ( j=0; j<3; j++ ) {
            printf( "%d ",board[i][j] );
        } /* end for j */
        printf( "\n" );
    } /* end for i */
    printf( "\n" );
} /* end of printBoard() */
```

- both methods work.



# labeling rows and columns when printing a grid.

- Suppose you wanted to print out your tic-tac-toe board, labelling its rows and columns as follows:

```
    | a | b | c |
  ---+---+---+---+
   d | 1 | 2 | 3 |
  ---+---+---+---+
   e | 4 | 5 | 6 |
  ---+---+---+---+
   f | 7 | 8 | 9 |
  ---+---+---+---+
```

- The numbers (1..9) are the contents of the board[][] variable. The letters (a..c,d..f) are column and row labels, respectively. All the other characters -- spaces, vertical bars (|), dashes (-) and plusses (+) -- are characters that you print to make the board look nifty.



# labeling rows and columns when printing a grid, 2.

- So to print the first line:

```
printf( "   | a | b | c |\n" );
```

- And to print the second line:

```
printf( "---+---+---+---|\n" );
```

- To print the third line, you'll intersperse the "nifty" characters with the board elements:

```
printf( " d | %d | %d | %d |\n",board[0][0],board[0][1],board[0][2] );
```

- The fourth line is just a repeat of the second line.
- You can continue in this way to print out the whole board, nifty style.
- If you do this, you'll notice some patterns and perhaps be able to figure out how to write the code more concisely...using nested for loops, as in the printBoard(), p3.

## labeling rows and columns when printing a grid, 3.

- You might even take advantage of the ASCII table. Suppose instead of printing

```
printf( "   | a | b | c |\n" );
```

- you had a character variable called `column_label` which you initialized to `'a'` and then incremented it after you'd printed the letter `'a'`:

```
char column_label = 'a';
int j;
printf( "   |" );
for ( j = 0; j<3; j++ ) {
    printf( " %c |",column_label );
    column_label++;
} /* end for j */
```

- You could do the same thing with a row_label variable.



## searching.

- often, when you have data stored in an array, you need to locate an element within that array.
- this is called *searching*.
- typically, you search for a "key" value (simply the value you are looking for) and return its "index" (the location of the value in the array)



## searching, 2.

- as with sorting, there are many searching algorithms.
- we'll study two:
  - linear search
  - binary search



## linear search.

- linear search simply looks through all the elements in the array, one at a time, and stops when it finds the key value.
- this is inefficient, but if the array you are searching is not sorted, then it may be the only practical method.
- a simple example follows on next page



## linear search, 2.

```
#include <stdio.h>

int linearSearch( int dice[], int size, int key );

int main( void ) {
  int dice[5] = { 3,4,6,5,2,1 };
  int k = linearSearch( dice,5,1 );
} /* end of main() */

/* this function returns the index of "key" in the "dice" array.
   it returns -1 if "key" is not found in the "dice" array. */
int linearSearch( int dice[], int size, int key ) {
  int i;
  for ( i=0; i<size; i++ ) {
    if ( dice[i] == key ) {
      return( i );
    } /* end if */
  } /* end for i */
  return( -1 );
} /* end linearSearch() */
```



## reading.

- DD 6.8, 6.9
- *Assignment #7 is due MON 2 APRIL*
- EXAM #3 will be on WED 11 APRIL