MC140: lecture #19

today's topic:

function arguments

passing arrays to functions

function arguments.

- function *arguments* are the variables between the parentheses in a function header
- the *value* of the argument is *passed* to the function, for use inside the function
- example:

```
#include <stdio.h>

void a ( int i );

int main( void ) {
  int x = 25;
  printf( "in main, x = %d\n", x );
  a ( x );
  printf( "in main, x = %d\n", x );
} /* end of main() */

void a ( int i ) {
  printf( "in a, i = %d\n", i );
} /* end of a() */
```

output:

```
in main, x = 25
in a, i = 25
in main, x = 25
```

function arguments, 2.

- when the argument is used inside the function, sometimes its value changes
- this change will NOT be retained when the function exits, because the argument is considered *local* to the function
- example:

```
#include <stdio.h>
void a ( int i );

int main( void ) {
  int x = 25;
  printf( "in main, x = %d\n", x );
  a ( x );
  printf( "in main, x = %d\n", x );
} /* end of main() */

void a ( int i ) {
  i++;
  printf( "in a, i = %d\n", i );
} /* end of a() */
```

output:

```
in main, x = 25
in a, i = 26
in main, x = 25
```

function arguments, 3.

- but sometimes, you WANT the changes to be retained when the function exits
- in this case, instead of passing the *value* of the variable into the function, you pass the *location* of the variable, *using a pointer!*
- example:

```
#include <stdio.h>
void a ( int *i );

int main( void ) {
  int x = 25;
  printf( "in main, x = %d\n", x );
  a ( &x );
  printf( "in main, x = %d\n", x );
} /* end of main() */

void a ( int *i ) {
  *i++;
  printf( "in a, i = %d\n", *i );
} /* end of a() */
```

output:

```
in main, x = 25
in a, i = 26
in main, x = 26
```

function arguments, 4.

- when you pass a pointer as a function argument, its scope is NOT local to the function
- its scope is the same as that of the function that called it
- so if the function is called from main() and the variable passed is declared in main(), then the variable is local to main()
- this information is helpful when drawing and labeling your boxes...

function arguments, 5.

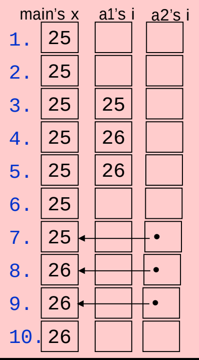
- the last two examples combined:

```
#include <stdio.h>
void a1 ( int i );
void a2 ( int *i );

    int main( void ) {
1.    int x = 25;
2.    printf( "in main, x = %d\n", x );
3.    a1 ( x );
6.    printf( "in main, x = %d\n", x );
7.    a2 ( &x );
10.   printf( "in main, x = %d\n", x );
    } /* end of main() */

    void a1 ( int i ) {
4.    i++;
5.    printf( "in a1, i = %d\n", i );
    } /* end of a1() */

    void a2 ( int *i ) {
8.    *i++;
9.    printf( "in a2, i = %d\n", *i );
    } /* end of a2() */
```

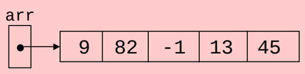


numbers are order in which lines are executed

passing arrays to functions.

- arrays are always passed to functions as pointers
- this is because arrays *are* pointers
- the name of the array is the same as &array[0] -- a pointer to the first element in the array
- arrays are really stored like this:

```
int arr[5] = {9,82,-1,13,45};
```



passing arrays to functions: example.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

/* define constant */
#define NUM_DICE 10

/* function prototypes */
int roll_die();
void roll_dice( int dice[],int size );
void print_dice( int dice[],int size );
void sort_dice( int *dice,int size );
void swap( int *a,int *b );
```

```
int main( void ) {

   int i, j;
   int dice[NUM_DICE];

   /* initialize random seed */
   srand( time ( NULL ));

   /* fill the dice array with
      random numbers */
   roll_dice( dice,NUM_DICE );

   /* print the dice */
   printf( "messy dice: " );
   print_dice( dice,NUM_DICE );

   /* sort the dice and print
      them again */
   sort_dice( dice,NUM_DICE );
   printf( "nice dice: " );
   print_dice( dice,NUM_DICE );

   return( 0 );

} /* end of main() */
```

passing arrays to functions: example, cont.

```
/* this function returns a random number between 1 and 6,
   symbolizing the roll of one die */
int roll_die() {
  return(( rand() % 6 ) + 1 );
} /* end of roll_die() */


/* this function fills the dice array, symbolically rolling
   NUM_DICE dice (the number of entries in the dice array) */
void roll_dice( int dice[], int size ) {
  int i;
  for ( i=0; i<size; i++ ) {
    dice[i] = roll_die();
  } /* end for i */
} /* end of roll_dice() */
```

passing arrays to functions: example, cont.

```
/* this function prints out the contents of the dice array.
   the dice are printed one at a time, separated by spaces,
   followed by a newline character. */
void print_dice( int dice[], int size ) {
  int i;
  for ( i=0; i<size; i++ ) {
    printf( "%d ",dice[i] );
  } /* end for i */
  printf( "\n" );
} /* end of print_dice() */
```

passing arrays to functions: example, cont.

```
/* this function swaps the values stored in the two argument variables */
void swap( int *a, int *b ) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
} /* end of swap() */

/* this function implements "bubble sort", sorting the entries in the
   dice array in ascending order. */
void sort_dice( int *dice, int size ) {
  int pass, i;
  for ( pass=1; pass<=size-1; pass++ ) {
    for ( i=0; i<=size-2; i++ ) {
      if ( dice[i] > dice[i+1] ) {
        swap( &dice[i],&dice[i+1] );
      } /* end if */
    } /* end for i */
  } /* end for pass */
} /* end of sort_dice() */
```

reading.

- material covered today:
  - DD: 6.5
- EXAM #2 will be on MON 19 MARCH
- EXAM #3 will be on WED 11 APRIL
- OFFICE HOURS:
  - additional hours on Friday 16th: 3-5pm