MC140: lecture #20

today's topic:

sorting

bubble sort

sorting.

- *sorting* is one of the classic tasks done in computer programs
- the basic idea with sorting is to rearrange the elements in an array so that they are in a specific order -- usually ascending or descending, in numeric or alphabetic order

sorting, 2.

- in this class, we will discuss 4 sorting algorithms:
  - bubble sort
  - blort sort
  - insertion sort
  - selection sort

sorting, 3.

- some sorts require an extra "auxiliary" array during sorting -- the elements are moved from the original array into the auxiliary array, one at a time
- at the end of the sort, the auxiliary array contains all the elements in sorted order
- the final step is to copy the elements from the auxiliary array back into the original array
- insertion, selection and blort sorts are this type

sorting, 4.

- some sorts do not use an auxiliary array during sorting, but just move the elements around within the original array
- these sorts involve the use of a swap() function, to switch the locations of two entries in the array
- bubble sort is this type

bubble sort.

```
#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() */
```

bubble sort, 2.

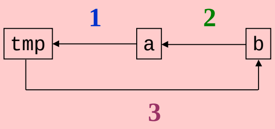
```
int roll_die() {
  return(( rand() % 6 ) + 1 );
} /* end of roll_die() */

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() */

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() */
```

bubble sort, 3:
swap function.

```
/* 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() */
```



bubble sort, 4.

```
/* 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 */
    printf( “after pass #%d: ” );
    print_dice( dice,size );
  } /* end for pass */
} /* end of sort_dice() */
```

one pass of bubble sort.

- *first, we compare 6 and 4. 6 is larger, so we swap the 6 and 4.*

dice = | 6 | 4 | 5 | 2 | 3 |

- *next, we compare 6 and 5. 6 is larger, so we swap the 6 and 5.*

dice = | 4 | 6 | 5 | 2 | 3 |

- *next, we compare 6 and 2. 6 is larger, so we swap the 6 and 2.*

dice = | 4 | 5 | 6 | 2 | 3 |

- *last, we compare 6 and 3. 6 is larger, so we swap the 6 and 3.*

dice = | 4 | 5 | 2 | 6 | 3 |

- *and that's the end of the first pass!*

dice = | 4 | 5 | 2 | 3 | 6 |

bubble sort:
sample run.

```
messy dice: 6 4 5 2 3
end of pass #1:  4 5 2 3 6
end of pass #2:  4 2 3 5 6
end of pass #3:  2 3 4 5 6
end of pass #4:  2 3 4 5 6
nice dice: 2 3 4 5 6
```

reading.

- material covered today:
  - DD: 6.5
- EXAM #3 will be on WED 11 APRIL