MC140: lecture #12

today's topic:

constants

arrays

strings, again!

nested loops

constants.

- #define
  - pre-processor directive
  - assigns a value that CANNOT ever be changed
  - in fact, pre-processor substitutes the value for the name
- examples:

```
#define MAX 1024
#define message "hello"
```


constants: example.

```
#include <stdio.h>
#define MAX 1024
int main( void ) {
   char str[MAX];
   printf( "enter your name: " );
   scanf( "%s",str );
}
```

your code

after pre-processor

```
// stdio definitions...
int main( void ) {
   char str[1024];
   printf( "enter your name: " );
   scanf( "%s",str );
}
```


strings.

- storing multiple characters in a single variable
- data type is still char
- BUT it has a *length*
- last character is *terminator* : '\0'
- string constants are surrounded by *double* quotes
- example:

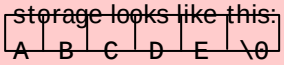
```
char s[6] = "ABCDE";
```


strings, 2.

- example:

```
#define MAX 6
char s[MAX] = "ABCDE";
```

- storage looks like this:



strings: printing.

- format sequence: %s
- example:

```
#include <stdio.h>
#define MAX 6
int main( void ) {
  char str[MAX] = "ABCDE";
  printf( "str = %s\n", str );
} /* end of main() */
```

- output:

ABCDE

string functions.

- string handling library
  `#include <string.h>`
- functions include:
  `int strlen( char *s );`
- many more functions, which we'll cover another day...

arrays.

- a string is an *array* of characters
- according to WWWebster:

  array, *noun.*

  1a. a regular or imposing grouping or arrangement. ORDER. *lined up in a soldierly array.*
- in C, an array has a length associated with it.

defining arrays.

- arrays need:
  (1) data type
  (2) name
  (3) length
- length can be determined
  - *statically* (at compile time) or
  - *dynamically* (at run time)

  (we'll mainly talk about static. we may get to dynamic at the end of the semester)

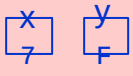
allocating memory.

- defining a variable is called "allocating memory" to store that variable
  - assigning a label to a byte (for a char) or set of bytes (for int, float, double) in memory (i.e., drawing a box)
- defining an array means allocating memory for a group of bytes
  - assigning a label to the FIRST byte(s) in the group (i.e., drawing a bunch of boxes)

boxes.

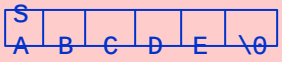
- single variables:

```
int  x = 7;
char y = 'F';
```



- arrays:

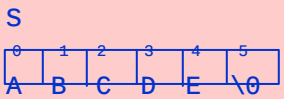
```
#define MAX 6
char s[MAX] = "ABCDE";
```



array elements.

- individual elements of arrays are *indexed*, starting with 0 and ending with length-1
- indeces follow array name, enclosed in [ ]
- example: `strlen(s) = 5`

```
s[0] = 'A'
s[1] = 'B'
s[2] = 'C'
s[3] = 'D'
s[4] = 'E'
s[5] = '\0'
```



printing elements of an array.

```
#include <stdio.h>
#define MAX 6
int main( void ) {
  char str[MAX] = "ABCDE";
  int i;
  for ( i=0; i<MAX-1; i++ ) {
     printf( "%c",str[i] );
  } /* end of for */
  printf( "\n" );
} /* end of main() */
```


another printing example.

```
#include <stdio.h>
#define SIZE 5
int main( void ) {
  int i, j;
  for ( i=0; i<SIZE; i++ ) {
     for ( j=0; j<SIZE; j++ ) {
          printf( "*" );
     } /* end for j */
     printf( "\n" );
  } /* end for i */
} /* end of main */
```


nested loops.

- like nested if-else
- one inside another
  - body can be another loop!
- example: *(blue nested inside red)*

```
for ( i=0; i<5; i++ ) {
   for ( j=0; j<5; j++ ) {
      printf( "i=%d j=%d\n", i, j );
   } /* end of for j */
} /* end of for i */
```


another printing example.

```
#include <stdio.h>
#define SIZE 5
int main( void ) {
  int i, j;
  for ( i=0; i<SIZE; i++ ) {
     for ( j=0; j<SIZE; j++ ) {
          printf( "*" );
     } /* end for j */
     printf( "\n" );
  } /* end for i */
} /* end of main */
```


reading.

- material covered today:
  - DD: 8.1-8.2, 8.10 (strlen)
- *assignment #4 is due Monday 26 Feb*