MC140: lecture #18

today's topic:

scope

functions, pointers, arguments

type casting

scope.

- refers to the portion of the program that knows about a variable
- DD defines four types:
  (1) file scope
  (2) function scope
  (3) function prototype scope
  (4) block scope (we won't cover this)

scope:

file scope.

- also called *global variables*
- any variables that are declared outside any function (including main)
- these variables are allocated when the program executes and any functions inside the same file know about them

board is a global variable

```
#include <stdio.h>
int board[3][3];
int main( void ) {
  a();
} /* end of main() */
void a ( int i ) {
   int x = 35;
   return( i + x );
} /* end of a() */
```


scope:

function scope.

- also called *local variables*
- any variables that are declared in the header of a function
- any variables that are declared within the body of the function
- these variables are essentially discarded after the function exits -- they only exist in memory during execution of the function

i and ***x*** are local variables

```
void a ( int i ) {
  int x = 35;
  return( i + x );
} /* end of a() */
```


scope:

static.

- the *static* keyword can be used when declaring a variable inside a function
- the variable is a local variable, however, its value is retained even after the function exits
- example:

```
void b ( void ) {
      static int x = 50;
      printf( "in b, x=%d\n",x );
      ++x;
      printf( "in b, x=%d\n",x );
} /* end of b() */
```


scope:

example.

```
#include <stdio.h>
void a( void ); /* function prototype */
void b( void ); /* function prototype */
void c( void ); /* function prototype */
int x = 1;      /* global variable */

int main( void ) {
   int x = 5;    /* local variable */
   printf( "in main, x = %d\n",x );
   a();
   b();
   c();
   a();
   b();
   c();
   printf( "in main, x = %d\n",x );
   return( 0 );
} /* end of main() */
```

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

void b ( void ) {
   static int x = 50;
   printf( "in b, x=%d\n",x );
   ++x;
   printf( "in b, x=%d\n",x );
} /* end of b() */

void c ( void ) {
   printf( "in c, x=%d\n",x );
   x *= 10;
   printf( "in c, x=%d\n",x );
} /* end of c() */
```

scope: example, output.

```
in main, x = 5
in a, x = 25
in a, x = 26
in b, x = 50
in b, x = 51
in c, x = 1
in c, x = 10
in a, x = 25
in a, x = 26
in b, x = 51
in b, x = 52
in c, x = 10
in c, x = 100
in main, x = 5
```


functions, pointers, 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
```


functions, pointers, arguments, 2.

- this is known as "*call by value*"
- a copy of the argument's value is made and passed to the function
- so if the value of the argument changes inside the function, this value will not be retained after the function exits
- 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
```


functions, pointers, arguments, 3.

- in C, you can simulate what is known as "*call by reference*" by passing a pointer to the value instead
- this way, a copy of the argument's *location* is made and passed to the function
- so changes to the value the location points to will be retained after the function exits
- 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
```


type casting.

- not only can you pass an address to a function, you can also pass a variable of another data type from what the function is expecting
- the behavior may be unpredictable, or what you want
- some conversions are convenient:
  - char ? int
  - float ? int
- example:

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

int main( void ) {
  int x = 65;
  printf( "in main, x = %d\n", x );
  a ( (char)x );
  printf( "in main, x = %d\n", x );
} /* end of main() */

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

output:

```
in main, x = 65
in a, i = A
in main, x = 65
```


reading.

- material covered today:
  - DD: 5.8, 5.12, pp 73-74
- EXAM #2 will be on MON 19 MARCH
- EXAM #3 will be on WED 11 APRIL
- OFFICE HOURS:
  - no office hours today
  - regular times on Wednesday 14th
  - additional hours on Friday 16th: 3-5pm