LECTURE 5: OPERATORS, ARRAYS, STRINGS, FUNCTIONS, PRECEDENCE (K&R §§ 1.6–1.9)

character counting (K&R, page 22)

```
#include <stdio.h>
int main(void) {
  int c, i, nwhite, nother;
  int ndigit[10];
  nwhite = nother = 0;

  for (i = 0; i < 10; ++i)
    ndigit[i] = 0;

  while ((c = getchar()) != EOF) {
    if (c >= '0' && c <= '9')
      ++ndigit[c-'0'];
    else if (c == ' ' || c == '\n' || c == '\t')
      ++nwhite;
    else
      ++nother;
  }

printf("digits =");
for (i = 0; i < 10; ++i)
  printf(" %d", ndigit[i]);
printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

LOGICAL OPERATORS

Apply logic functions to boolean arguments (arguments that evaluate to true or false).

Recall that in C:	0 is false
	nonzero is true

Evaluated left-to-right.
Evaluation stops as soon as truth or falsehood is known.

not	!x	converts a nonzero operand into 0 zero operand into 1
and	x && y && ... && z	1 if all operands are true, 0 otherwise
or	x || y || ... || z	1 if any operand is true, 0 otherwise

ARRAYS

array	a way to store many values under one name
	array[] is a pointer to sequential memory locations containing elements of defined type.

int ndigit[10];	int	the type of elements in the array
	ndigit	the name of the array
	[10]	the number of elements in the array

Array indexes always start at 0, so the elements of ndigit correspond to the indexes 0 through 9.

index/ subscript	Number used to indicate an element of an array. Enclosed in brackets following array name. ndigit[0] refers to the first element in the array named ndigit.

CHARACTERS AS INTEGERS

how does this if statement work?

```
if (c >= '0' && c <= '9')
  ++ndigit[c-'0'];
```

	(c >= '0' && c <= '9')
1.	each character constant has an ASCII encoding
	the ASCII encodings for the characters 0 through 9 are numbered in ascending order (see ASCII table)
	so, we can test whether the ASCII encoding of a character falls within the range that is only the digits 0 through 9.
	in other words, we are checking: 48 <= the ASCII encoding of c <= 57 where 48 is the decimal ASCII encoding of 0 57 is the decimal ASCII encoding of 9
	++ndigit[c-'0'];
2.	if we are here, then c must be a digit 0-9
	we can now subtract the encoding of '0' from c to determine which digit and the increment the corresponding counter.
	e.g., if c is 7 (ASCII encoding 55) c - '0' is equivalent to 55 - 48 = 7 ++ndigit[7] then increments our counter for 7s

CHARACTER ARRAYS / CHARACTER STRINGS

string constant/ string literal	a sequence of 0 or more characters surrounded by double quotes
	ends with a null character (\0)
quotes are not part of the string; serve only to delimit	
stored as an array of characters	

We can define/initialize an array to contain the string "hello" and the end of line character:

```
char array[7] = "hello\n";
```

Why do we need 7 slots in this array?

Recall that escape sequences count as 1 character.

This definition sets up memory locations as follows:

array[0]	array[1]	array[2]	array[3]	array[4]	array[5]	array[6]
'h'	'e'	'l'	'l'	'o'	'\n'	'\0'

MORE ON FUNCTIONS

Functions provide a convenient way to encapsulate some computation, which can then be used without worrying about its implementation.

power (K&R, page 26)

```
#include <stdio.h>

int power(int m, int n);

/* test power function */
main() {
  int i;
  for (i = 0; i < 10; ++i)
    printf("%d %d %d\n", i, power(2,i), power(-3,i));
  return 0;
}

/* power: raise base to n-th power; n >= 0 */
int power(int base, int n) {
  int i, p;
  p = 1;
  for (i = 1; i <= n; ++i)
    p = p * base;
  return p;
}
```

function definition

```
form of a function definition:
return-type function-name(parameter declarations, if any){
  declarations
  statements
}
```

return type	int
function-name	power
parameter declarations	int base, int n note that these parameter names are local to the function
declarations	int i, p; variables declared within the function are also local

statements	p = 1;
	for (i = 1; i <= n; ++i)
	p = p * base;
	return p;

function declaration

```
int power(int m, int n);
```

declared before main, says that power is a function that expects two int arguments and returns an int.

This declaration is called a function prototype.

If the definition or any use of this function do not follow the pattern in the prototype, you will get an error.

Note:	parameter names are optional in the prototype
	used often, however, for clarity

CALL BY VALUE vs CALL BY REFERENCE

call by value
In C all function arguments are passed "by value."
The called function is given the values of its arguments in temporary variables, distinct from the originals.
This means that anything you do to a variable inside a function has no effect on that variable outside of the function.
call by reference
We can alter an argument outside of the function with actions inside the function if we pass its address as an argument.

FUNCTION EXAMPLE

maxline (K&R, page 29)

```
#include <stdio.h>

#define MAXLINE 1000 /* maximum input line length */

/* function declarations */
int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print the longest input line */
main() {
  int len; /* current line length */
  int max; /* maximum length seen so far */
  char line[MAXLINE]; /* current input line */
  char longest[MAXLINE]; /* longest line saved here */
  max = 0;
  while ((len = getline1(line, MAXLINE)) > 0)
    if (len > max) {
      max = len;
      copy(longest, line);
    }
  if (max > 0) /* there was a line */
    printf("%s", longest);
  /* return statement in main is optional as C99
   * specification returns 0 from main by default */
  return 0;
}
```

```
/* getline1: read a line into s, return length */
int getline1(char s[],int lim) {
  int c, i;
  for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
    s[i] = c;
    if (c == '\n') {
      s[i] = c;
      ++i;
    }
  s[i] = '\0';
  return i;
}
```

```
/* copy: copy 'from' into 'to'; assume to is big enough */
void copy(char to[], char from[]) {
  int i;
  i = 0;
  while ((to[i] = from[i]) != '\0')
    ++i;
}
```

NOTES ON THE DETAILS

unsized array arguments
The `char s[]`, `char to[]`, and `char from[]` arguments all have unspecified length.
This is ok as the lengths of those arrays are set in main.

operator precedence
`i < lim-1 && (c = getchar())!= EOF && c != '\n'`
How do we know what to do first? Consult precedence table.

pass by value with pointers
An array name is actually an address.
This is how copy is able to make changes to an array that is passed to it as an argument.

OPERATOR PRECEDENCE

	Operators	Associativity
first	`() [] -> .`	left to right
	`! ~ ++ -- + - * (type) sizeof`	right to left
	`* / %`	left to right
	`+ -`	left to right
	`<< >>`	left to right
	`< <= > >=`	left to right
	`== !=`	left to right
	`&`	left to right
	`^`	left to right
	`|`	left to right
	`&&`	left to right
	`||`	left to right
	`?:`	right to left
	`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
last	`,`	left to right

ASSOCIATIVITY	determines order if the operators have the same precedence, e.g.:
`x = y += z -= 4`	go right to left per above table
`x = y += (z -= 4)`	
`x = (y += (z -= 4))`	

PARAMETERS vs ARGUMENTS

There are referred to in a variety of ways:

formal parameter	The names given to the arguments in the function definition.
	Often referred to as parameters.
actual parameter	The named supplied in a function call.
	Often referred to as arguments.