Duane's Incredibly Brief Introduction to the C Programming Language

The one best book on C is The C Programming Language by Kernighan and Richie.
The 'g' in 'Kernighan' is silent.

CODE
  Code for execution goes into files with ".c" suffix.
  Shared decl's (included using #include "mylib.h") in "header" files, end ".h"

COMMENTS
  Characters to the right of // are not interpreted; they're a comment.
  Text between /* and */ (possibly across lines) is commented out.

DATA TYPES

Name	Typ. size	Description
char	1 byte	an ASCII value: e.g. 'a' (see: man ascii)
int/long	4 bytes	a signed integer: e.g. 97 or hex 0x61, oct 0x141
long long	8 bytes	a longer mult-byte signed integer
float	4 bytes	a floating-point (possibly fractional) value
double	8 bytes	a double length float

char, int, and double are most frequently & easily used in small programs
sizeof(double) computes the size of a double in addressable units (bytes)
Zero values represent logical false, nonzero values are logical true.
Math library (#include <math.h>, compile with -lm) prefers double.

CASTING
Preceding an primitive expression with an alternate parenthesized type converts or "casts" value to new value equivalent in new type:
```
   int a = (int)3.141; // assigns a=3, without complaint.
```
Preceding any other expr'n with a cast forces new type for unchanged value.
```
   double b = 3.141;
   int a = *(int*)&b; // interprets the double b as an integer (not 3!)
```

STRUCTS and ARRAYS and POINTERS and ADDRESS COMPUTATION
Structs collect several fields into a single logical type:
```
   struct { int n; double root; } s; // s has two fields, n and root
   s.root = sqrt((s.n = 7)); // ref fields (N.B. double parens=>assign OK!)
```
Arrays indicated by right associative brackets ([]) in the type declaration:
```
   int a[10];  // a is a 10 int array.  a[0] is first element.  a[9] is last.
   char b[]     // in function header, b is array of chars w/unknown length
   int c[2][3]; // c is an array of 2 arrays of 3 ints. a[1][0] follows a[0][2]
```
 Array variables (e.g. a,b,c) cannot be made to point to other arrays.
 Strings are represented as character arrays terminated by ASCII zero.
Pointers indicated by left associative asterisk (*) in the type declaration:
```
   int *a;    // a is a pointer to an integer
   char *b;   // b is a pointer to a character
   int *c[2]; // c is an array of 2 pointers to ints; same as int *(c[2]);
   int (*d)[2]; // d is a pointer to an array of 2 integers.
```
 Pointers are simply addresses. Pointer variables may be assigned.
  Adding 1 computes p't'r to next value by adding sizeof(X) for base type X.
  General int adds to ptr (even negative or zero) follow in a obvious manner.
Addresses may be computed with the ampersand (&) operator.
  An array without an index or a struct without field computes its address:
```
   int a[10],b[20]; // two arrays
   int *p = a;      // p points to first int of array a
   p = b;           // p now points to first int of array b
```
  An array OR POINTER with an index n in square brackets returns nth value:
```
   int a[10];  // an array
   int *p;
   int i = a[0]; // i is second element of a
   i = *a; // pointer dereference
   p = a; // same as p = &a[0];
   p++;   // same as p = p+1; same as p = &a[1]; same as p = a+1
```
  Bounds are never checked; your responsibility. Never assume.
  An arrow (-> no spaces!) dereferences a pointer to a field:
```
   struct { int n; double root; } s[1]; // s is pointer to struct or array of 1
   s->root = sqrt(s->n = 7); // s->root same as (*s).root or s[0].root
   printf("%g\n",s->root);
```

FUNCTIONS
A function is a pointer to some code, parameterized by formal parameters, that may be executed by providing actual parameters. Functions must be declared before they are used, but code may be provided later. A sqrt function for positive n might be declared
```
  double sqrt(double n) {
     double guess;
     for (guess = n/2.0; abs(n-guess*guess)>0.001; guess = (n/guess+guess)/2);
     return guess;
  }
```
This function has type double (*sqrt)(double).
```
  printf("%g\n",sqrt(7.0)); // calls sqrt; actuals always passed by value
```
Function parameters are always passed by value. Functions must return a value. The return value need not be used.Function name with no parameters returns the function pointer. An alias for sqrt may be declared:
```
  double (*root)(double) = sqrt
  printf("%g\n",root(7.0));
```
Procedures or valueless functions return 'void'.
There must always be a main function that returns an int.
```
   int main(int argc, char **argv)
```
Programs arguments may be accessed as strings through main's array argv with argc elements. First is the program name. Function decl's are never nested.

OPERATIONS

+, -, *, /, %	Arithmetic ops. / truncates on integers, % is remainder.
++i --i	Add or subtract 1 from i, assign result to i, return new val
i++ i--	Remember i, inc or decrement i, return remembered value
&& || !	Logical ops. Right side of && and || not eval'd unless nec.
& | ^ ~	Bit logical ops: and, or, xor, complement.
>> <<	Shift right and left: int n = 10; n << 2 computes 40.
=	Assignment is an operator. Result is value assigned.
+= -= *= etc	Perform binary op on lft and rght, assign result to left
== != < > <= >=	Comparison operators (useful only on primitive types)
?:	If-like expression: (x%2==0)?"even":"odd"
,	compounding; value is last: a = b,c,d; exec's b,c,d then a=d

STATEMENTS
Angle brackets identify syntactic elements and don't appear in real statements.
```
  <expression> ;          // semi indicates end of simple statement
  break; // quits tightest loop or switch prematurely
  continue; // jumps to next loop test, skipping rest of loop body
  return x; // quits this function, returns x as value
  { <statements> }  // curly-bs group statements into 1 compound. Note: no semi
  if (<condition>) <statement> // stmt executed if cond true (nonzero)
  if (<condition>) <statement> else <statement> // 2-way condition
  while (<condition>) <statement> // repeatedly exec stmt only if cond true
  do <statement> while (<condition>); // note semi.  statement often compound.
  for (<init>; <condition>; <step>) <statement>
    // <init> and <step> are assignents. above for is similar to
    <init> while (<condition>) { <statement> <step> }
  switch (<expression>) {          // traditional "case statement"
    case <value>: <statement>      // this statement exec'd if val==expr
                  break;           // quit this statment when val==expr
    case <value2>: <statement2>    // exec'd if val2==expr
    case <value3>: <statement3>    // exec'd if val3==expr OR val2==expr
                  break;           // quit
    default: <statement4>          // if matches no other value; may be first
             break;                // optional (but encouraged) quit
  }
```

KEY WORDS

unsigned	before primitive type suggests unsigned operations
extern	in global declaration => symbol is for external use (e.g. main)
static	in global declaration => symbol is local to this file
	in local decl => don't place on stack; keep value between calls
typedef	before declaration defines a new type name, not a new variable
mom	not a keyword; true love, instantiated; call her

Duane’s Incredibly Brief Introduction to the C Programming Language

```
I/O (#include <stdio.h>)
Default input comes from "stdin"; output goes to "stdout"; errors to "stderr".
Standard input and output routines are declared in stdio.h: #include <stdio.h>
  Function         Description
  fopen(name,"r")  opens file name for read, returns FILE *f; "w" allows write
  fclose(f)        closes file f
  getchar()        read 1 char from stdin or pushback; is EOF (int -1) if none
  ungetch(c)       pushback char c into stdin for re-reading; don’t change c
  putchar(c)       write 1 char, c, to stdout
  fgetc(f)         same as getchar(), but reads from file f
  ungetc(c,f)      same as ungetch(c), but onto file f
  fputc(c,f)       same as putchar(c), but onto file f
  fgets(s,n,f)     read strng of n-1 chars to s from f, or til eof or nl (kept)
  fputs(s,f)       writes string s to f: e.g. fputs("Hello world\n",stdout);
  scanf(p,...)     reads ... args using format p (below); put & w/non-pointers
  printf(p,...)    write ... args using format p (below); pass args as-is
  fprintf(f,p,...) same, but print to file f
  fscanf(f,p,...)  same, but read from file f
  sscanf(s,p,....) same as scanf, but from string s
  sprintf(s,p,...) same as printf, but to string s
  feof(f)          return true iff at end of file f
 Formats use format characters preceded by escape %; other chars written as-is.
   char    action                      char      meaning
   %c      character                    \n       newline (control-j)
   %d      decimal integer              \t       tab (control-i)
   %s      string                       \\       slash
   %g      general floating point       %%       percent

MEMORY (#include <stdlib>)
  malloc(n)     alloc n bytes of memory; for type T: p = (T*)malloc(sizeof(t));
  free(p)       free memory pointed at p; must have been alloc’d; don’t re-free
  calloc(n,s)   alloc n-array size s & clear; typ: a = (T*)calloc(n,sizeof(T));

MATH (#include <math.h> and link -lm; sometimes documented in man math)
 All functions take & return double unless otherwise noted:
  sin(a),cos(a),tan(a) sine, cosine, tan of double radian angle a
  asin(y),acos(x),atan(r)  principal inverse of above
  atan2(y,x)           principal inverse of tan(y/x) in same quadrant as (x,y)
  sqrt(x)              root of x
  log(x)               natural logarithm of x; others: log2(x) and log10(x)
  exp(p)               e to the power of p; others: exp2(x) and exp10(x)
  pow(x,y)             x to the power of y; like exp(y*log(x))
  ceil(x)              smallest integer (returned as double) no less than x
  floor(x)             largest integer (returned as double) no greater than x
 #include <stdlib.h> for these math functions:
  abs(x)               absolute value of x
  random()      returns random long
  srandom(seed) sets random generator to use new long seed

STRINGS (#include <string.h>)
  strlen(s)       return length of string; number of characters before ASCII 0
  strcpy(d,s)     copy string s to d and return d; N.B. parameter order like =
  strncpy(d,s,n)  copy at most n characters of s to d and terminate; returns d
  stpcpy(d,s)     like strcpy, but returns pointer to ASCII 0 terminator in d
  strcmp(s,t)     compare strings s and t and return first difference; 0=>equal
  strncmp(s,t,n)  stop after at most n characters; needn’t be zero terminated
  memcpy(d,s,n)   copy exactly n bytes from s to d; may fail if s overlaps d
  memmove(d,s,n)  (slow) copy n bytes from s to d; won’t fail if s overlaps d

COMPILING
  gcc prog.c    # compiles prog.c into a.out  run result with ./a.out
  gcc -o prog prog.c # compiles prog.c into prog; run result with ./prog
  gcc -g -o prog prog.c # as above, but allows for debugging
  gcc -O -o prog prog.c lib.c # compiles, links prog and lib together, optimize
  gcc -O3 -o prog prog.c -lX  # link to lib libX (X=m for math); heavy optimize
  gcc -g -c prog.c # generate object file (not exec) prog.o for later linking
[This is http://www.cs.williams.edu/~bailey/c.pdf]
```

```
A GOOD FIRST PROGRAM
#include <stdio.h>
int main()
{
    printf("Hello, world.\n");
}

WORD COUNT (wc)
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
   int charCount = 0, wordCount = 0, lineCount = 0;
   int doChar=0, doWord=0, doLine=0, inWord = 0;
   int c;
   char *fileName = 0;
   FILE *f = stdin;
   while (argv++, --argc) {
       if (!strcmp(*argv,"-c")) doChar=1;
       else if (!strcmp(*argv,"-w")) doWord=1;
       else if (!strcmp(*argv,"-l")) doLine=1;
       else if (!(f = fopen((fileName = *argv),"r")))
          { printf("Usage: wc [-l] [-w] [-c]\n"); return 1; }
   }
   if (!(doChar || doWord || doLine)) doChar = doWord = doLine = 1;
   while (EOF != (c = fgetc(f))) {
      charCount++;
      if (c == '\n') lineCount++;
      if (!iswspace(c)) {
        if (!inWord) { inWord = 1; wordCount++; }
      } else { inWord = 0; }
   }
   if (doLine) printf("%8d",lineCount);
   if (doWord) printf("%8d",wordCount);
   if (doChar) printf("%8d",charCount);
   if (fileName) printf(" %s",fileName);
   printf("\n");
}

NOTES:
```