Quick and Dirty Guide to C

The single best book on C is The C Programming Language by Kernighan and Richie.

CODE:

Code for execution goes into files with ".c" suffix.
Shared decl's (included using #include "mylib.h") in "header" files, end in ".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	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 multi-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 and 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 a primitive expression with an alternate parenthesized type converts or "casts" value to a new value equivalent in new type:

```
  int a - (int) 3.131;    //assigns a=3 without complaint
```

Preceding any other expression with a cast forces new type for unchanged value.

```
  double b = 3.131;
  int a = *(int*)&b; //interprets the double b as an integer (not necessarily 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 10int array. a[0] is the first element. a[9] is the last
  char b[];    //in a function header, b is an array of chars with unknown length
  int c[2][3]; //c is an array of 2 arrays of three ints. a[1][0] follows a[0][2]
```

Array variables (e.g. a,b,c above) cannot be made to point to other arrays
Strings are represented as character arrays terminated by ASCII zero.

Pointers are indicated by left associative asterisk (*) in the type declarations:

```
  int a*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 two 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 pointer to the next value by adding sizeof(X) for type X
General int adds to pointer (even 0 or negative values) behave in the same way

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 the first int of array b
```

An array or pointer with an index n in square brackets returns the nth value:

```
  int a[10];        // an array
  int *p;
  int i = a[0];     // i is the first 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 not checked; your responsibility not to run off. Don't 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 as:

```
  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 (s*sqrt)(double).

```
  printf("%g\n", sqrt(7.0)); //calls sqrt; actuals are always passed by value
```

Functions parameters are always passed by value. Functions must return a value. The return value need not be used. Function names with parameters returns the function pointer. Thus, 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)   OR  int main(int argc, char *argv[])
```

Program arguments may be accessed as strings through main's array argv with argc elements. First is the program name. Function declarations 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 || unless necessary
& | ^ ~	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 left and right, assign result to left
== != < > <= >=	Comparison operators (useful only on primitive types)
?:	If-like expression: (x%2==0)?"even":"odd"
,	computing 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> ;          //semicolon indicates end of a simple statement
  break;                  //quits the tightest loop or switch immediately
  continue;               //jumps to next loop test, skipping rest of loop body
  return x;               //quits this function, returns x as value
  { <statements> }        //curly-brace groups statements into 1 compound (no ;)
  if (<condition>) <stmt> //stmt executed if cond true (nonzero)
  if (<condition>) <stmt> else <stmt> // two-way condition
  while (<condition>) <stmt>    //repeatedly execute stmt only if condition true
  do <stmt> while (<condition>); //note the semicolon, executes at least once
  for (<init>; <condition>; <step>) <statement>

  switch (<expression>) {         //traditional "case statement"
    case <value>: <statement>     // this statement exec'd if val==expr
                break;            // quit this when value == expression
    case <value2>: <statement2>   //executed if value2 = expression
    case <value3>: <statement3>   //executed if value3 = expression
                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
static	in global declaration => symbol is local to this file
	in local decl'n => don't place on stack; keep value betw'n calls
typedef	before declaration defines a new type name, not a new variable

Quick and Dirty Guide to C

Content borrowed and updated (with permission) from Duane A. Bailey’s guidelines from 2007.

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 ungetchar() but onto file f
fputc(c,f)	same as putchar(c), but onto file f
fgets(s,n, f)	read string of n-1 chars to a s from f or til eof or \n
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, but read 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	meaning	char	meaning
%c	character	\n	newline (control-j)
%d	decimal integer	\t	tab (control-i)
%s	string	\\	slash
%g	general floating point	%%	perent

MEMORY (%include <stdlib.h>)

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 and return double unless otherwise noted:

sin(a), cos(a), tan(a)	sine, cosine, tangent of double (in radians)
asine(y),acos(x),atan(r)	principle 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 (expy*log(x))
ceil(x)	smallest integer (returned as double) no less than x
floor(x)	largest integer (returned as double) no greater than y

#include <stdlib.h> for these math functions

abs(x)	absolute value of x
random()	returns a random long
srandom(seed)	seeds the random generator with a new random 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 terminarot 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 null 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
```

A GOOD FIRST PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv){
   printf(“Hello, world.\n”);
   return 0;
}
```

A 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 (!iswpace(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”);
 }
```

ADD YOUR NOTES HERE: