C Quick Reference

Data types

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
<th>printf/scanf</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>42</td>
<td>%d</td>
</tr>
<tr>
<td>char</td>
<td>'d'</td>
<td>%c</td>
</tr>
<tr>
<td>float</td>
<td>4.3, 5.6e7</td>
<td>%f</td>
</tr>
<tr>
<td>double</td>
<td>4.3, 5.6e7</td>
<td>%lf</td>
</tr>
<tr>
<td>char *</td>
<td>&quot;a string&quot;</td>
<td>%s</td>
</tr>
</tbody>
</table>

Pointer types

Declare a pointer to int with

```
int *myptr;
```  
Set `myptr` to point to `a` with

```
myptr = &a;
```  
and use it with

```
*myptr = 42; // same as a = 42
```  

Operators and relations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>increment, decrement</td>
</tr>
<tr>
<td>--</td>
<td>increment, decrement</td>
</tr>
<tr>
<td>()</td>
<td>function call</td>
</tr>
<tr>
<td>[]</td>
<td>array subscript</td>
</tr>
<tr>
<td>.</td>
<td>struct field</td>
</tr>
<tr>
<td>-&gt;</td>
<td>struct field via ptr</td>
</tr>
<tr>
<td>-</td>
<td>unary minus</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>(type)</td>
<td>cast</td>
</tr>
<tr>
<td>*</td>
<td>dereference</td>
</tr>
<tr>
<td>&amp;</td>
<td>address of</td>
</tr>
<tr>
<td>* / %</td>
<td>multn, divn, mod</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>gtr/less than/or equal</td>
</tr>
<tr>
<td>== !=</td>
<td>(not) equal</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
</tbody>
</table>

Control flow

Assignment: `a = b+3;`

Conditional:

```
if ( a > 3 ) {
    b++;  
} else {
    b--; 
}
```  

Iteration:

```
while ( i < n ) {
    ... 
}
```  
```
for (i = 0; i < n; i++) {
    ... 
}
```  
```
bREAK; terminates a for or while loop. 
cONTINUE; skips the rest of the loop body this time round
```

Switch statement:

```
switch ( i ) {
    case 0:
        ...
        break;
    case 1: case 2:
        ...
        break;
    default:
        ...
}
```  

Functions

Prototype:

```
char nthcharof(char *, int);
```  
Definition:

```
char nthcharof(char *str, int n) {
    return str[n];
}
```  
Use:

```
char *mystr = "a boring string";
char c;
c = nthcharof(mystr,7);
```  

Structs

Declaring a struct type with typedef:

```
typedef struct {
    int n;
    char *str;
} mytype;
```  
Using structs:

```
mytype x;
x.n = 42;
x.str = "forty-two";
```  

Pointers and structs:

```
mytype *xptr = &x;
xptr->n = 43;
xptr->str = "forty-three";
```  

Enums

```
typedef enum { FIRST, SECOND } num_t;
```  
```
typedef enum { APPLE = 3, PEAR, ORANGE } fruit_t
```  
```
typedef enum { LEEK = 5, TOM = 8, CARROT = 10 } veg_t
```  

Strings

```
typedef enum { FIRST, SECOND } num_t; Enums are ints; start from zero by default. Can do:
typedef enum { APPLE = 3, PEAR, ORANGE } fruit_t
```  
```
typedef enum { LEEK = 5, TOM = 8, CARROT = 10 } veg_t
```  
for arbitrary values

```
are null-terminated arrays of chars.
```  
```
Useful functions (#include <string.h>):
char *s, *s1, *s2;
```  
```
strlen(s) length of s, excluding final null
```  
```
strcpy(s1,s2); copy contents of s2 into s1
```  
```
strcmp(s1,s2); return -1, 0, 1 as s1 is <,=,> s2 in
```  
```
lexicographic order
```  
```
sprintf(s,...) print into s
```  
```
sscanf(s,...) read from s
```
Arrays

Declaring:
```c
int myarray[10];
```
Using
```c
myarray[i+1] = 2*myarray[i];
```
Arrays and pointers: `myarray[i]` is the same as `*(myarray+i)`, and `&(myarray[i])` is the same as `myarray+i`
Arrays and strings: after
```c
char achar[] = { 'a', 'b', 'c', 'd', 0 };
char *mystr = achar;
```
`mystr` is the string "abcd", and `achar[i] == mystr[i]`

Basic i/o

Printing formatted strings:
```c
int n; double f; char c; char *str;
printf("n is %d, f is %f, c is %c, str is %s\n",
n, f, c, str);
```
Reading variables from input:
```c
scanf("%d %lf %c %s",&n,&f,&c,str)
```
(N.B. no & for %s)
scanf skips white space before numbers or strings, and when there is a space in the format string. Skipping a value: "%s"
Useful printf format modifiers:
`%3d` pad with blanks on left to 3 columns
`%03d` pad with zeros on left to 3 columns
`%.3f` print to 3 decimal places
Character i/o:
```c
int c; /* N.B. int NOT char */
c = getchar();
putchar(c);
```

File i/o

Input:
```c
char *filename = "foo.txt";
FILE *infile;
infile = fopen(filename,"r");
scanf(infile,"%d",&n);
char c = fgetc(infile);
```
Output:
```c
FILE *outfile;
outfile = fopen(filename,"w");
fprintf(outfile,"n is %d\n",n);
```

Character identification

```c
#include <ctype.h>
```
provides
```c
int isalpha(int c); and similarly isdigit, isupper,
islower, isspace
and int toupper(int c); and similarly tolower.
```