

a late-in-season picking of finest bytes
have been brought together to produce this boutique

C reference card

Compilation

```
gcc flags file.c
-c compile only, default output file.o
-o out output to out
```

Lexical Structure & Preprocessor

```
/* a comment, maybe over multiple lines */
// a comment to the end of the line
```

```
#include <system-header.h>
#include "user-header.h"
#define symbol replacement-text
```

.h files: #defines, typedefs, function prototypes

.c files: #defines, structs, statics, function definitions;
optionally also `int main (int argc, char *argv[])`

Identifiers start with a letter, followed by letters, digits, or underscores. Identifiers starting with '_' are reserved for system use. The following words are also reserved:

```
auto break case char const continue default
do double else enum extern float for goto
if inline int long register restrict return
short signed sizeof static struct switch
typedef union unsigned void volatile while
_Bool _Complex _Imaginary
```

Type Qualifiers

extern accessible everywhere
static file-local; value saved across function calls
const can't/won't change **volatile** likely to change
restrict function will use this pointer-type argument
inline function should be inlined at call

Statements

```
expression; { statements... }
if (condition) statement [else statement]
switch (condition) {
case constant: statements... break;
[default: statements...]
}
while (condition) statement
for (initialiser; condition; increment) statement
do statement while (condition);
break; terminate loop or switch
continue; resume next iteration of loop
return [value]; return (optional) value from function
goto label; transfer to label (EVIL)
```

Operators

decreasing precedence downwards left-to-right
operators are left-associative
except cast, ternary, assignment

```
() brackets [v] vth index
. struct field -> struct*'s field ('arrow', 'stab')
++ increment -- decrement + value-of - negate
! logical-NOT * dereference ('value-at')
~ bitwise-NOT & reference ('address-of')
(type) cast to type sizeof(type) #bytes of type
* / % + - arithmetic << >> left/right bitshift
< <= > >= relational operators == != (in)equality
& bitwise-AND | bitwise-OR ^ bitwise-XOR
&& logical-AND || logical-OR ?: ternary
= += -= *= /= %= (arithmetic on) assignment
, sequential comma
```

Literals

```
integers (int): 123 -4 0xAf0C 057
reals (float/double): 3.14159265 1.29e-23
characters (char): 'x' 't' '\033'
strings (char *): "hello" "abc\n" ""
```

Declarations

```
int i, length;
char *str, buf[BUFSIZ], prev;
double x, values[MAX];
typedef enum { FALSE, TRUE } Bool;
typedef struct { char *key; int val; } keyval_t;
```

Initialisation

```
int c = 0;
char prev = '\n';
char *msg = "hello";
int seq[MAX] = { 1, 2, 3 };
keyval_t keylist[] = {
    "NSW", 0, "Vic", 5, "Qld", -1 };

```

Character & String Escapes

<code>\n</code>	line feed ("newline")	<code>\r</code>	carriage return
<code>\t</code>	horizontal tab	<code>\e</code>	escape
<code>'</code>	single quote	<code>"</code>	double quote
<code>\\</code>	backslash	<code>\0</code>	null character
<code>\ddd</code>	octal ASCII value	<code>\xdd</code>	hex ASCII value

The C Standard Library

only a limited, ‘interesting’ subset is listed here.
type modifiers, notably `const`, have been omitted.
consult the relevant *man(1)* or *info(1)* pages.

```
// in stdlib.h
```

```
#define NULL ((void *)0)
void *malloc(size_t size);
void *calloc(size_t number, size_t size);
    allocate size or (number * size) bytes.
    calloc initialises allocated space to zero.
void free(void *obj);
    release allocated pointer obj, no-op if NULL.
void exit(int status);          void abort();
    terminate the current program (ab)normally.
    returns status to the OS or sends SIGABRT.
int atoi(char *str);
long strtol(char *str, char **end, int base);
    converts string str to a (long) int value.
    strtol supports radix  $2 \leq \text{base} \leq 36$ ;
    references first invalid char in end if non-NULL.
double atof(char *str);
double strtod(char *str, char **end);
    converts string str to a double.
    first invalid character address in end if non-NULL.
int abs(int x);                long labs(long x);
    returns  $|x|$ 
```

```
// in ctype.h
```

```
int toupper(int c);           int tolower(int c);
    make ASCII c uppercase or lowercase
int isupper(int c);           int islower(int c);
int isalpha(int c);           int isalnum(int c);
int isdigit(int c);           int isxdigit(int c);
int isspace(int c);           int isprint(int c);
    is ASCII c upper/lowercase, alphabetic, alphanumeric,
    a digit, a hex digit, whitespace, or printable?
```

```
// in stdio.h
```

```
#define EOF (-1)
    special ‘end-of-file’ return value
FILE *stdin, *stdout, *stderr;
    standard input/output/error
FILE *fopen(char *filename, char *mode);
    open file; return new ‘file handle’.
int fclose(FILE *fh);
    close a file; returns non-zero on error.
int fgetc(FILE *fh);          int getchar(void);
    return next character from fh, or EOF on EOF/error.
    getchar equivalent to fgetc(stdin)
char *fgets(char *s, int size, FILE *fh);
    read into s until EOF, newline, or size bytes.
    returns s, or NULL on error.
int fputc(int c, FILE *fh);   int putchar(int c);
    write c to fh; returns EOF on error.
    putchar(k) equivalent to fputc(k, stdout)
int fputs(char *str, FILE *fh);
int puts(char *str);
    write str to fh; returns EOF on error.
    puts(k) equivalent to fputs(k "\n", stdout)
int printf(char *fmt, ...);
int fprintf(FILE *fh, char *fmt, ...);
int snprintf(char *buf, size_t len, char *fmt,
    ...);
    print text per fmt to stdout, fh or buf.
    formatting commands: "%m w. p c"
    field width in w; < 0 left-justifies; decimal places in p.
    code in c: decimal, octal, hexadecimal, char, string,
        fixed-point, general, exp., pointer, literal %
    size in m: long [long]; short [short], size_t, ptrdiff_t.
    arguments with matching types follow fmt
    returns number of characters written, or EOF on error
int scanf(char *fmt, ...);
int fscanf(FILE *fh, char *fmt, ...);
int sscanf(char *str, char *fmt, ...);
    parse text from stdout, fh or str per fmt.
    fmt is not exactly the same as printf formats.
    pointer arguments with matching types follow fmt
    returns number of fields matched, or -1 on error
```

```
// in string.h
```

```
size_t strlen(char *str);
    the length of str without trailing NUL.
char *strcpy(char *dst, char *src);
char *strcat(char *dst, char *src);
    copy or concatenate src onto dst until NUL or sz
int strcmp(char *s1, char *s2);
    return < 0, = 0, > 0 if s1 <, =, > s2
char *strchr(char *str, int c);
char *strrchr(char *str, int c);
    find first/last instance of c in str, or NULL
char *strstr(char *haystack, char *needle);
    find first instance of string needle in haystack
char *strpbrk(char *str, char *any);
    find first of any of any in str.
size_t strspn(char *str, char *any);
size_t strcspn(char *str, char *any);
    length of prefix of any of any (not) in str
char *strsep(char **strp, char *sep);
    find first of any of sep in *strp, writes NUL
    returns original *strp, byte after sep in strp
    replaces old strtok
```

```
// in math.h
```

```
// remember to compile and link -lm
double sin(double x);         double asin(double x);
double cos(double x);         double acos(double x);
double tan(double x);         double atan(double x);
    returns  $\sin, \sin^{-1}, \cos, \cos^{-1}, \tan, \tan^{-1}$  of x
double atan2(double y, double x);
    returns  $\tan^{-1} \frac{y}{x}$ 
double exp(double x);         double log(double x);
    returns  $\exp, \log_e$  of x
double pow(double x, double y);
    returns  $x^y$ 
double sqrt(double x);
    returns  $\sqrt{x}$ 
double floor(double x);       double ceil(double x);
    returns  $\lfloor x \rfloor$  and  $\lceil x \rceil$ 
double fabs(double x);
    returns  $|x|$ 
double fmod(double x, double y);
    returns  $x \bmod y$ 
```