

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;
`int main (int argc, char *argv [])`

Identifiers

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
```

Statements

```
expression; { statements... }
if (condition) statement [else statement]
while (condition) statement
return [value]; return (optional) value from function
```

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 - negate ! logical-NOT

* dereference & reference ('address-of')
~ bitwise-NOT (1s-complement) (typename) type cast
* / % + - arithmetic << >> left/right bitshift
< <= > >= relational operators == != (in)equality
& bitwise-AND | bitwise-OR ^ bitwise-XOR
&& logical-AND || logical-OR ?: ternary
= += -= *= /= %= (arithmetic on) assignment
, sequential comma

sizeof (typename) - bytes required to hold typename

sizeof variable - bytes required to hold variable

Literals

integers (int): 123 -4 0xAf0C 057
reals (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

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

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 an (long) int.
double atof(char *str);
    converts string str to a double.
int abs(int x);
```

// in string.h

```
size_t strlen(char *str);
    the length of str without trailing NUL.
char *strcpy(char *dst, char *src);
size_t strncpy(char *dst, char *src, size_t sz);
char *strcat(char *dst, char *src);
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);
    points to 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 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 math.h

```
// compile and link -lm if not using dcc
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);
double log10(double x);
    returns exp, loge, log10 of x
double pow(double x, double y);
    returns xy
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 mod y
```

// 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 sprintf(char *str, char *fmt, ...);
    print text per fmt to stdout, fh or str.
    formatting commands: "%mw.p c"
    field width in w; < 0 left-justifies. double 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
```

ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL '\0' (null)	32	20	Space	64	40	@	96	60	`
1	01	SOH (start of header)	33	21	!	65	41	A	97	61	a
2	02	STX (start of text)	34	22	"	66	42	B	98	62	b
3	03	ETX (end of text)	35	23	#	67	43	C	99	63	c
4	04	EOT (end of transmission)	36	24	\$	68	44	D	100	64	d
5	05	ENQ (enquiry)	37	25	%	69	45	E	101	65	e
6	06	ACK (acknowledgment)	38	26	&	70	46	F	102	66	f
7	07	BEL '\a' (bell)	39	27	'	71	47	G	103	67	g
8	08	BS (backspace)	40	28	(	72	48	H	104	68	h
9	09	TAB '\t' (horizontal tab)	41	29	)	73	49	I	105	69	i
10	0A	LF '\n' (line feed, new line)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT '\v' (vertical tab)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF '\f' (form feed)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR '\r' (carriage return)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO (shift out)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI (shift in)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (data link escape)	48	30	0	80	50	P	112	70	p
17	11	DC1 (device control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (device control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (device control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (device control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (negative ack)	53	35	5	85	55	U	117	75	u
22	16	SYN (synchronous idle)	54	36	6	86	56	V	118	76	v
23	17	ETB (end of trans. block)	55	37	7	87	57	W	119	77	w
24	18	CAN (cancel)	56	38	8	88	58	X	120	78	x
25	19	EM (end of medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (file separator)	60	3C	<	92	5C	\	124	7C	
29	1D	GS (group separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (record separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (unit separator)	63	3F	?	95	5F	_	127	7F	DEL