C Quick Reference Card

Version 1 **by Eric Roman**

ASCII Table

Least Significant Nibble \ Most Significant Nibble	0_	1_	2_	3_	4_	5_	6_	7_
_0	NUL	DLE	SP	0	@	P	‘	p
_1	SOH	DC1	!	1	A	Q	a	q
_2	STX	DC2	“	2	B	R	b	r
_3	ETX	DC3	#	3	C	S	c	s
_4	EOT	DC4	$	4	D	T	d	t
_5	ENQ	NAK	%	5	E	U	e	u
_6	ACK	SYN	&	6	F	V	f	v
_7	BEL	ETB	`	7	G	W	g	w
_8	BS	CAN	(	8	H	X	h	x
_9	HT	EM	)	9	I	Y	i	y
_A	LF	SUB	*	:	J	Z	j	z
_B	VT	ESC	+	;	K	[	k	{
_C	FF	FS	,	<	L	\	l	|
_D	CR	GS	-	=	M	]	m	}
_E	SO	RS	.	>	N	^	n	~
_F	SI	US	/	?	O	_	o	DEL

Order of Operations

Precedence	Operation	Description
1	`() [] -> .`	Function call, scope, array
2	`! ~ - + * & sizeof( ) ++ --`	(most)unary operators, sizeof and type casts (right to left)
3	`* / % MOD`	Multiplication, division, modulo
4	`+ -`	Addition, subtraction
5	`<< >>` (bitwise)	Shift left and right
6	`< <= > >=`	Comparisons: less-than and greater-than
7	`== !=`	Comparisons: equal and not equal
8	`&` (bitwise)	AND
9	`^` (bitwise)	XOR
10	`|` (bitwise)	OR
11	`&&` (logical)	AND
12	`||` (logical)	OR
13	$expr_1$? $expr_2$: $expr_3$	Conditional expression
14	`= += -= *= /= %= &= |= ^= <<= >>=`	Assignment operators (right to left)

Operators

	Operand	Meaning
Arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	%	Modulus
	++	Increment by 1
	--	Decrement by 1
	++>	Pre-increment and compare
	-->	Pre-decrement and compare
Equality	==	Equal to
	!=	Not equal to
	<	Less than
	>	Greater than
	<=	Less than or equal to
	>=	Greater than or equal to
Logical	&&	And
	||	Or
	!	Not ★
Bitwise	&	AND
	|	OR
	^	XOR
	~	NOT ★
	<<	Left-shift
	>>	Right-shift
Assignment	=	Assign ★
	+=	Add ★
	-=	Subtract ★
	*=	Multiply ★
	/=	Divide ★
	%=	Modulus ★
	<<=	Left-shift ★
	>>=	Right-shift ★
	&=	AND ★
	|=	OR ★
	^=	XOR ★
exp_1?exp_2:exp_3		Conditional expression★

★ right associative (otherwise left associative)

Bit Operations

```
#define CHKBIT(var,bit)  ((var)&(1<<(bit)))
#define SETBIT(var,bit)  ((var)|=(1<<(bit)))
#define CLRBIT(var,bit)  ((var)&=(~(1<<(bit))))
#define FLPBIT(var,bit)  ((var)^=(1<<(bit)))
#define CHKBITMSK(var,msk)   ((var)&(msk))
#define SETBITMSK(var,msk)   ((var)|=(msk))
#define CLRBITMSK(var,msk)   ((var)&=(~(msk)))
#define FLPBITMSK(var,msk)   ((var)^=(msk))
```

Flow Statements

Do	`do` *statement* `while (`*expression*`);`
For	`for (`*initializer*`;` *condition*`;` *increment*`)` *statement*
If	`if (`*expression*`)` *statement* `elseif (`*expression*`)` *statement* `else` *statement*

Switch	`switch (`*expression*`) {`
	`case` *constant*`:` *statements*... `break;`
	`default:` *statements*
	`}`

While	`while (`*expression*`)` *statement*

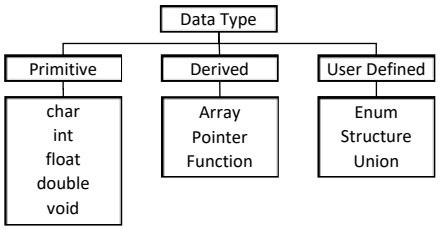
Character Escapes

\0	NULL
\b	Backspace
\f	Form feed (new page)
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\”	Double quote
\\	Backslash
\’	Single quote
\?	Question mark

Data Types with ARM

`bool, char, uint8_t, int8_t`	1 byte
`short, uint16_t, int16_t`	2 bytes
`int, long, float, uint32_t, int32_t, (pointers)`	4 bytes
`long long, double, uint64_t, int64_t`	8 bytes

Data Type Qualifiers



Keywords

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

Storage Classes

Storage Specifier	Storage	Initial Value	Scope	Life
Auto (default for local)	Stack	uninitialized	Within block	End of block
Extern	Data segment	Zero	Global multiple files	Till end of program
Static (default for global)	Data segment	Zero	Within block	Till end of program
register	CPU register	uninitialized	Within block	End of block
(none, use malloc and free)	Heap	Uninitialized	N/A	Dynamic

Type Qualifiers

const	Value does not change; read-only
restrict	For the lifetime of the pointer, the object can only be accessed via the pointer
volatile	Optimizing-compilers must not change

Pointers

& → the reference operator or the address of the variable

* → the dereference operator or the value from the address

`int *pointer = &variable;`	Declare
`pointer = NULL;`	Assign
`value = *pointer;`	Dereferencing

array[i] equivalent to *(array + i)

array[i][j] equivalent to *(*(array + i) + j)

Pointer Constructs

`int i;`	Integer variable ‘i’
`int *p;`	Pointer ‘p’ to an integer
`int a[];`	Array ‘a’ of integers
`int f();`	Function ‘f’ with return value of type integer
`int **pp;`	Pointer ‘pp’ to a pointer to an integer
`int (*pa)[];`	Pointer ‘pa’ to an array of integers
`int (*pf)();`	Pointer ‘pf’ to a function with return value integer
`int *ap[];`	Array ‘ap’ of pointers to an integer
`int *fp();`	Function ‘fp’ with returns a pointer to an integer
`int ***ppp;`	Pointer ‘ppp’ to a pointer to a pointer to an integer
`int (**ppa)[];`	Pointer ‘ppa’ to a pointer to an array of integers
`int (**ppf)();`	Pointer ‘ppf’ to a pointer to a function with return value of type integer
`int *(*pap)[];`	Pointer ‘pap’ to an array of pointers to an integer
`int *(*pfp)();`	Pointer ‘pfp’ to function with return value of type pointer to an integer
`int **app[];`	Array of pointers ‘app’ that point to pointers to integer values
`int (*apa[])[];`	Array of pointers ‘apf’ to functions with return values of type integer
`int (*apf[])();`	Array of pointers ‘apf’ to functions with return values of type integer
`int ***fpp();`	Function ‘fpp’ which returns a pointer to a pointer to a pointer to an int
`int (*fpa())[];`	Function ‘fpa’ with return value of a pointer to array of integers
`int (*fpf())();`	Function ‘fpf’ with return value of a pointer to function which returns an integer

Copyright © 2022 Embedded Wilderness, Eric Roman

C Library Files

<assert.h>	Macros assertions
<complex.h>	Arithmetic with complex numbers
<ctype.h>	Testing characters
<float.h>	Floating types
<limits.h>	Implementation defined constants
<math.h>	Mathematical and trigonometric functions
<regex.h>	Regular expression matching
<stdbool.h>	Boolean type and values
<stddef.h>	Standard type definitions
<stdint.h>	Integer types
<stdio.h>	Standard buffered I/O
<stdlib.h>	Standard library definitions
<string.h>	Several string operations
<strings.h>	Case-insensitive string comparisons
<time.h>	Time and date functions, structures and macros
<types.h>	Various data-types

Character Tests <ctype.h>

Alphanumeric?	isalnum(c)
Alphabetic?	isalpha(c)
Control character?	iscntrl(c)
Decimal digit?	isdigit(c)
Printing character (not incl. space)?	isgraph(c)
Lower case letter?	islower(c)
Printing character (incl. space)?	isprint(c)
Printing char except space, letter, digit?	ispunct(c)
Space, formfeed, newline, cr, tab, vtab?	isspace(c)
Upper case letter?	isupper(c)
Hexadecimal digit?	isxdigit(c)
Convert to lower case	tolower(c)
Convert to upper case	toupper(c)

ASCII Values	Characters	isalnum	isalpha	iscntrl	isdigit	isgraph	islower	isprint	ispunct	isspace	isupper	isxdigit
0x00 – 0x08	/0 (other control characters)			•								
0x09 – 0x0D	/t/n/v/f/r			•						•		
0x0E – 0x1F	(control characters)			•								
0x20	(space)							•		•		
0x21 – 0x2F	!"#$%&'()*+,-./					•		•	•			
0x30 – 0x39	0123456789	•			•	•		•				•
0x3A – 0x40	:;<=>?@					•		•	•			
0x41 – 0x46	ABCDEF	•	•			•		•			•	•
0x47 – 0x5A	GHIJKLMNOPQRSTUVWXYZ	•	•			•		•			•	
0x5B – 0x60	[\]^_					•		•	•			
0x61 – 0x66	abcdef	•	•			•	•	•				•
0x67 – 0x7A	ghijklmnopqrstuvwxyz	•	•			•	•	•				
0x7B – 0x7E	{|}~					•		•	•			
0x7F	(delete)			•								

• function will return non-zero, otherwise will return zero

Integer Type Limits <limits.h>

# of bits in a byte	CHAR_BIT	8
Min for signed char	CHAR_MIN SCHAR_MIN	-128
Max for signed char	CHAR_MAX SCHAR_MAX	127
Max for unsigned char	UCHAR_MAX	255
Min for short	SHRT_MIN	-32768
Max for short	SHRT_MAX	32767
Max for unsigned short	USHRT_MAX	65535
Min for integer	INT_MIN	-2147483648
Max for integer	INT_MAX	2147483647
Max for unsigned integer	UINT_MAX	4294967295

Mathematical Functions <math.h>

Arguments and returned values are double.

Trig functions	sin(x), cos(x), tan(x)
Inverse trig functions	asin(x), acos(x), atan(x)
Arctan(y/x)	atan2(y,x)
Hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
Exponentials & logs	exp(x), log(x), log10(x), frexp(x, e), ldexp(x,e)
Exponentials & logs (2 power)	ldexp(x, n), frexp(x, &e)
Division & remainder	modf(x, ip), fmod(x, y)
Powers	pow(x, y), sqrt(x)
Rounding	ceil(x), floor(x), fabs(x)

Standard I/O <stdio.h>

Open file	file = fopen(s, mode)
Close file	fclose (file)
Print formatted output to file/string	fprintf(file, format, list) sprintf(file, format, list)
Read formatted input from file/string	fscanf(file, format, list) sscanf(file, format, list)
Read character from file	fgetc(file) getc(file)
Write character to file	fputc(char, file) putc(char, file)

String Format Specifiers (scanf and printf)

%[flags][width][.precision][length]specifier

Flags	
–	Left-justify (right-justify is default)
+	Forces plus or minus sign
#	Used with o, x or X specifiers, value preceded with 0, 0x or 0X respectively
0	Left-pads number with zeros
(space)	Blank space before value
Width and Precision	
(number)	Minimum number of characters to be printed. Examples: %.3f → 3.142; %4d → 2020; %2.2f → 19.95
*	Not specified in format string but additional integer has to be formatted.
Length	
h	Short (i, d, o, u, x, X specifiers)
l	Long (i, d, o, u, x, X, c, s specifiers)
L	Long double (e, E, f, g, G specifiers)
Specifier	
%c	Char
%s	String
%d %i	Signed int
%u	Unsigned int
%l	Signed long
%f %F	Float or double
%o	Octal unsigned int
%x %X	Hex unsinged int
%p	Pointer address

Standard Utility Functions <stdlib.h>

Absolute value of int n	abs(n)
Absolute value of long n	labs(n)
Quotient and remainder of ints n, d	div (n, d)
Quotient and remainder of longs n, d	ldiv(n, d)
Pseudo-random integer [0, RAND_MAX]	rand()
Set random seed to n	srand(n)
Convert string s to double	atof(s)
Convert string s to integer	atoi(s)
Convert string s to long	atol(s)
Convert prefix of s to double	strtod(s, &endptr)
Convert prefix of s (base b) to long	strtol(s, &endptr, b)
Convert prefix of s (base b) to unsinged long	strtoul(s, &endptr, b)
Allocate storage	malloc(size), calloc(val, size)
Change size of storage	ptr = realloc(ptr, size)
Deallocate storage	free(ptr)
Search array for key	bsearch(key, array, n, size, cmpf)
Sort array ascending order	qsort(array, n, size, cmpf)

String Operations <string.h>

s is a string; cs, ct are constant strings

Length of s	strlen(s)
Copy ct to s	strcpy(s, ct)
Concatenate ct after s	strcat(s, ct)
Concatenante ct after s using first n characters	strncat(s, ct, n)
Compare cs to ct	strcmp(cs, ct)
Compare cs to ct using only first n chars	strncmp(cs, ct, n)
Pointer to first c in cs	strchr(cs, c)
Pointer to last c in cs	strrchr(cs, c)
Compare string cs to ct	strcoll(cs, ct)
Find the length of cs which consists of characters not in ct	strcspn(cs, ct)
Find the length of cs which consists of characters in ct	strspn(cs, ct)
Find the first occurrence of string ct in cs	strstr(cs, ct)
Find the first occurance in cs of any character specified in ct	strpbrk(cs, ct)
Break string cs into a series of tokens separated by ct	strtok(cs, ct)
Copy n chars from ct to s	memcpy(s, ct, n)
Copy n chars from ct to s (may overlap)	memmove(s, ct, n)
Compare n chars of cs with ct	memcmp(cs, ct, n)
Pointer to first c in first n chars of cs	memchr(cs, c, n)
Put c into first n chars of s	memset(s, c, n)

Time and Date Functions <time.h>

Current processor time	clock()
Current calendar time	time()
Time difference in seconds	difftime(t1, t2)
Convert local time to calendar time	mktime(t)
Convert time to string	asctime(t)
Convert calendar time to local time	ctime(t)
Convert calendar time to GMT	gmtime(t)
Convert calendar time to local time	localtime(t)
Format date and time info	strftime(s, smax, "format", t)

Copyright © 2022 Embedded Wilderness, Eric Roman