



CONVEX C V4.1 Quick Reference

Document No. 720-003030-002

Fifth Edition
April 1991

CONVEX Press
Richardson, Texas USA

CONVEX C Quick Reference
Released with CONVEX C V4.1
Order No. DSW-087
Fifth Edition

© 1987, 1988, 1989, 1990, 1991 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, stored electronically, or reduced to machine-readable form without prior written consent from CONVEX Computer Corporation.

Although the material contained herein has been carefully reviewed, CONVEX Computer Corporation (CONVEX) does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

CONVEX and the CONVEX logo ("C") are registered trademarks of
CONVEX Computer Corporation.

Printed in the United States of America

Keywords

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

* CONVEX extension

Statement Formats

expression:	Simple statement.
{statement; statement;...}	Compound statement.
while (exp) statement	Do statement while (exp) is true; test before each iteration.
do-statement while (exp);	Do statement while (exp) is true; test after each iteration.
for (exp1; exp2; exp3) statement	Execute exp1, while (exp2) do {statement; exp3}.
if (exp) statement	If (exp) is true, do statement.
if (exp) stmt1 else stmt2	If (exp) is true do stmt1, otherwise do stmt2.
switch (exp)	Evaluate (exp) and go to matching case label.
{	
case constant_exp: statement	
...	
default: statement	Default if no case matched.
}	
goto label;	Go to labeled statement.
label: statement	Define statement as target for goto.
break;	End smallest enclosing <i>while</i> , <i>do</i> , <i>for</i> , or <i>switch</i> .
continue;	Begin next iteration of loop.
return expression;	Exit function and return optional expression.
;	Null statement.

Allocation Classes

register short quick;	Try to assign variable to register.
extern int flag. open();	Defined externally.
static char arg;	Local permanent storage.
auto long arg;	Dynamic storage.

Data Types

[signed] char	Signed, one byte (8 bits).
[signed] short [int]	Signed integer (16 bits).
[signed] [long] int	Signed integer (32 bits).
[signed] long	Signed integer (32 bits).
signed	Signed integer (32 bits).
[signed] long long [int]	Signed long long integer (64 bits).*
unsigned char	Unsigned, one byte (8 bits).
unsigned short [int]	Unsigned short integer (16 bits).
unsigned long	Unsigned long integer (32 bits).
unsigned int	Unsigned integer (32 bits).
unsigned long [int]	Unsigned integer (32 bits).
unsigned long long	Unsigned long long integer (64 bits).*
float	Floating point (32 bits).
double	Long floating point (64 bits).
long double	Long floating point (64 bits).
long float	Long floating point (64 bits).^
void	Type with no value.

* CONVEX extensions.

^ CONVEX extension in the backward-compatible mode.

Type Constructors

char msg[]="testing\n";	Initializing array of char to null-terminated string.
struct word { char first [10]; char last[20]; unsigned case: 1; } letter;	Define complex data type. Declare variable "case" with one bit. Declare variable "letter" of type "struct word".
union identifier { char c; float f; }mixed;	Define overlay of different data types. Declare variable "mixed" of type "union identifier".
float matrix [10][50];	Two-dimensional floating-point array (32 bits).
typedef char *string;	Define new data type name "string".
enum hue {red, green, blue};	Enumeration constant data.

Type Qualifiers

const int i	Constant integer.
int *const i	Constant pointer to integer.
const int *const i	Constant pointer to constant integer.
volatile int i	Volatile integer.
int *volatile i	Volatile pointer to integer.
volatile int *volatile i	Volatile pointer to volatile integer.
const volatile int i	Constant volatile integer— unchangeable by program, but changeable by external forces.

Constants

'a'	Character.
L'a', L"abc"	Wide characters.
1234	Integer decimal.**
0Xaa55	Integer hexadecimal.**
0177	Integer octal.**
1234U	Unsigned integer decimal.**
0Xaa55U	Unsigned integer hexadecimal.**
0177U	Unsigned integer octal.**
32.5, 123F	Double.
32.5D	Double.*
33.13F	Float.
1.2e-5, 1.2E-5	Double with exponent.
1.2e-5F, 1.2E-5F	Double with exponent.
"abcd"	Address of null-terminated char string.

* CONVEX Extension.

** Integer constants can also have the suffix L (long) or LL (long long). LL is a CONVEX extension.

Note: Hexadecimal and octal require a leading zero. The F, L, LL, U, X, and hexadecimal digits A through F notations can be specified in lowercase.

Operators

Operator	Associativity
() [] call array element -> structure pointer structure member	left to right
(datatype) * & sizeof - cast indirect address (bytes) unary minus ! ~ ++ -- + not 1's comp increment decrement unary plus	right to left
* / % multiply divide modulus	left to right
+ - add subtract	left to right
<< >> shift left shift right	left to right
< <= > >= less-than less-or-equal greater-than greater-or-equal	left to right
== != equals not equal	left to right
& bitwise AND	left to right
^ bitwise exclusive OR	left to right
 bitwise OR	left to right
&& logical AND (ANDIF)	left to right
 logical OR (ORIF)	left to right
condition ? true_expr : false_expr	right to left
>>= <<= &= ^= = = += -= *= /= %= assignment operators	right to left
. comma operator	left to right

Note: Operators appear in decreasing order of precedence. Operators grouped together have the same precedence.

Escape Sequences

<code>\a</code>	alert	<code>\r</code>	carriage return
<code>\b</code>	backspace	<code>\t</code>	horizontal tab
<code>\f</code>	form feed	<code>\v</code>	vertical tab
<code>\n</code>	newline		
<code>\'</code>	single quote	<code>\"</code>	double quote
<code>\\</code>	backslash	<code>\?</code>	question mark
<code>\ddd</code>	octal character constant	<code>\xdd</code>	hex character constant

Unformatted I/O

Declarations

```
char buffer[ ], ch, *path, *ptr, *mode;
long offset;
FILE *stream;

FILE * fopen(path, mode);
/* mode: 'r' read, 'w' write, 'a' append, 'r+' read/write */
/* (start at beginning), 'w+' read/write (truncate, start new) */
/* 'a+' read/write (start at end) Also rb, wb, ab, r+b, w+b, */
/* a+b, rb+, wb+, ab+ for binary files */

FILE * freopen(path, mode, stream);
fclose(stream);
fread(ptr, item_size, nmemb, stream);
fwrite(ptr, item_size, count, stream);
fread
getc(stream);
getchar();
char * gets(buffer); /* NULL at EOF; delete newlines */
fgets(buffer,n,stream); /* NULL at EOF; keep newlines */
putc(ch,stream);
putchar(ch);
puts(buffer);
```

Formatted I/O

<code>printf(control, arg1, arg2, ...)</code>	To standard output.
<code>fprintf(stream, control, arg1, arg2, ...)</code>	To specified stream.
<code>vprintf(control, arg_list)</code>	To standard output.
<code>sprintf(string, control, arg1, arg2, ...)</code>	To string buffer.
<code>scanf(control, &arg1, &arg2, ...)</code>	From standard input.
<code>fscanf(stream, control, &arg1, &arg2, ...)</code>	From specified stream.
<code>sscanf(string, control, &arg1, &arg2, ...)</code>	From string buffer.

Output Format Specifier (printf and variants)

% [Flags] [Width] [Prec] [Length] <conversion character>

Flags

- Left justify.
- + Use + sign for positive values, - sign for negative.
- <blank> Use space for positive values, - sign for negative.
- # Use alternate forms.
- 0 Pad with leading zeroes.

Width

- W Minimum field width W digits.
- * Get width from argument list.

Prec

- .M M digits of precision.
- * Get precision from argument list.

Length

- l Long integer or double (letter el).
- L Long double.
- ll Long long int.*
- LL Long long int.*

Conversion characters

- c Unsigned character.
- d Signed decimal integer.
- e Double (scientific notation), use 'e' for exp.
- E Double (scientific notation), use 'E' for exp.
- f Double (fixed-point notation).
- g %e or %f, whichever is shorter.
- G %E or %f, whichever is shorter.
- h Short (converted to int).
- i Signed decimal integer.
- l Long (converted to int).
- L Long double (converted to float).
- n Returns number of characters printed so far. int*.
- o Unsigned octal integer.
- p Pointer value.
- s Null-terminated string.
- u Unsigned decimal integer.
- x Unsigned hex integer.

* CONVEX extensions.

Input Format Specifier (scanf and variants)

% [Flag] [Width] [Prec] [Length] <Conversion character>

Flag

* Suppress assignment.

Width

W Width.

Length

h Short int or unsigned short int.

l Long int, unsigned long int, or double (letter e).

L Long double.

ll Long long int.*

LL Long long int.*

Conversion characters

c Reads single char or sequence of W chars into char* or char[] argument.

d Reads signed decimal integer into int* arg.

e, f, g, Reads floating-point number in

E, F scientific notation into float*.

h Short int, signed or unsigned.

i Reads integer into int*.

n Returns number of characters read so far. int*.

o Reads octal integer into unsigned int*.

p Reads hex integer (no 0x prefix) into void*.

s Reads string into char[] or char*.

u Reads decimal integer into unsigned int*.

x Reads hex integer into unsigned int*.

% Matches a single %. No argument.

* CONVEX extensions.

Compilation

Invoking the compiler:

```
cc [options] files [loader_options]
```

where

options	is zero or more compiler options (see below).
files	is one or more C source files, object files, assembly files, or libraries.
loader_options	is zero or more loader options.

Preprocessor Options

-C	Do not delete comments.
-Dname[=def]	Define name as if by #define.
-E	Run the preprocessor only.
-Idir	Search alternate directory for #include files.
-k	Generate dependency descriptions for make.
-Uname	Remove initial definition of name.

Optimization Options

-alias array_args	Assume formal array parameters don't overlap each other or other variables.
-alias ptr_args	Assume pointer formal parameters don't overlap each other or other variables.
-alias cautious	Performs most optimal aliasing.
-alias standard	Performs aliasing based on the pointer alias assumptions of ANSI C.
-alias worst	Assume worst case aliasing. Objects may be modified by pointers of different type.
-ds	Perform dynamic selection on loops.
-ep n	Specify expected number of CPUs.
-no	Perform no optimization.
-O	Perform highest scalar optimization.
-O0	Perform local scalar optimization.
-O1	Perform global scalar optimization.
-O2	Perform vectorization.
-O3	Perform automatic parallelization.
-rI	Perform loop replication.
-uo	Perform potentially unsafe optimizations.
-ur	Perform loop unrolling.

Code-Generation Options

-c	Suppress linking.
-extern distinct	Identical uninitialized file scope variables in different files are different variables.
-extern same	Identical uninitialized file scope variables in different files are the same variable. Default.
-fd	Synonym for -float sp_ops command line option.
-fi	Use IEEE floating-point format.
-float dp_const	Unsuffix floating-point constants are double.
-float dp_ops	Floating-point operations use double precision.
-float sp_const	Unsuffix floating-point constants are float.
-float sp_ops	Floating-point operations use float precision.
-fn	Use native floating-point format.
-parens explicit	Parentheses of floating-point expressions are always honored.
-parens ignore	Compiler can ignore parentheses in floating-point expressions.
-parens implicit	Compiler honors parentheses, grammar, and associativity rules in floating-point expressions.
-re	Generate reentrant code.
-S	Generate symbolic assembly code.
-string read_only	String literals cannot be modified.
-string read_write	String literals can be modified.
-tm target	Generate code for target machine (C1 or C2).

Compatibility Options

- ext Compile ANSI C, POSIX, and CONVEX extensions. This is the default.
- std Compile ANSI C and POSIX mode.
- str Compile ANSI C mode.
- pcc Select backward-compatible mode.

Debugging and Profiling Options

- cxdb Compile for debugging with CXdb.
- db Compile for debugging with csd or pmd.
- p Compile for profiling with prof.
- pb Compile for profiling with bprof.
- pg Compile for profiling with gprof.
- pa Compile for routine- and loop-level profiling with CXpa.
- pab Compile for block-level profiling with CXpa.
- par Compile for routine-, loop-, and region-level profiling with CXpa.

Message and Listing Options

- d name Turn off named diagnostic message.
- d name=e Report message name as an error.
- d name=w Report message name as a warning.
- or table Produce optimization report on loops, arrays, both, or none.
- sc Syntax check only.
- w, -nw Suppress warning diagnostics.

Miscellaneous Options

- Bdirectory Find substitute compiler in named directory.
- o name Specify name for executable module.
- tl n Set maximum compile time to n minutes.
- vn Identify compiler version.

Preprocessor Statements

<code>#define identifier token-string</code>	Replace identifier with token-string.
<code>#define identifier (identifier. ... identifier) token-string</code>	Replace identifier with macro.
<code>#error "message"</code>	Produce compile-time error message.
<code>#include "filename"</code>	Replace this line with contents of filename.
<code>#include <filename></code>	Replace this line with contents of /usr/include/filename.
<code>#include token</code>	Replace this line with contents of file defined by token.
<code>#line n identifier</code>	Next source line is n; current input file is identifier.
<code>#if const_expression</code>	Compile if constant_expression true.
<code>#else</code>	Compile if previous <i>if</i> condition is false.
<code>#elif const_expression</code>	Else-if. Compile if previous <i>if</i> condition is false and <i>elif</i> condition is true.
<code>#endif</code>	Terminate conditional compile.
<code>#ifdef identifier</code>	Compile if identifier is defined.
<code>#ifndef identifier</code>	Compile if identifier is not defined.
<code>defined(name)</code>	True if name is defined; otherwise false.
<code>#pragma compiler_info</code>	Pass implementation-defined information to the compiler.
<code>#undef identifier</code>	Undo previous define.
<code># blank line</code>	Null statement.
<code># string</code>	Convert the unexpanded parameter to a string and replace.*
<code>tkn1 ## tkn2</code>	Concatenate unexpanded tokens.*

* Used only in macro definitions.

Pragmas

General format:

```
#pragma _CNX pragma-name [options]
```

Pragma can be

<code>begin_tasks</code>	Begin task group.
<code>end_tasks</code>	End task group.
<code>force_parallel</code>	Parallelize loop. Does not interchange outer loop for vectorization.
<code>force_parallel_ext</code>	Parallelize loop. Can interchange outer loop for vectorization.
<code>force_vector</code>	Vectorize loop.
<code>max_trips (n)</code>	Specify maximum trip count.
<code>next_task</code>	Begin individual task.
<code>no_parallel</code>	Do not parallelize loop.
<code>no_recurrence</code>	Ignore apparent recurrences.
<code>no_side_effects (f [.f]...)</code>	Functions have no side effects.
<code>no_vector</code>	Do not vectorize loop.
<code>prefer_parallel</code>	Prefer to parallelize loop.
<code>prefer_parallel_ext</code>	Perform <code>force_parallel_ext</code> if safe.
<code>prefer_vector</code>	Prefer to vectorize loop.
<code>pstrip (n)</code>	Specify parallel strip-mine length.
<code>scalar</code>	Process loop as scalar.
<code>select (v. p. pv)</code>	Generate and dynamically select scalar, vector, parallel, and parallel/vector versions of a loop, based on specified trip count.
<code>synch_parallel</code>	Parallelize loop, insert <code>synch</code> code to honor dependencies.
<code>unroll</code>	Unroll the following loop.
<code>vstrip (n)</code>	Specify vector strip-mine length.

ANSI C function prototypes

assert.h

```
void assert( expression ) /* macro */
```

ctype.h

```
int isalnum(int c);  
int isalpha(int c);  
int iscntrl(int c);  
int isdigit(int c);  
int isgraph(int c);  
int islower(int c);  
int isprint(int c);  
int ispunct(int c);  
int isspace(int c);  
int isupper(int c);  
int isxdigit(int c);  
int tolower(int c);  
int toupper(int c);
```

locale.h

```
struct lconv *localeconv( void );  
char *setlocale( int category, const char *locale );
```

math.h

```
double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double numer, double denom);
double cos(double x);
double sin(double x);
double tan(double x);
double cosh(double x);
double sinh(double x);
double tanh(double x);
double exp(double x);
double frexp(double x, int *exp);
double ldexp(double x, int exp);
double log(double x);
double log10(double x);
double modf(double value, double *iptr);
double pow(double x, double exp);
double sqrt(double x);
double ceil(double x);
double fabs(double x);
double floor(double x);
double fmod(double x, double y);
```

setjmp.h

```
int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int retval);
```

signal.h

```
void (*signal(int signal, void (*func)(int)))(int);
int raise(int signal);
```

stdarg.h

```
void va_start(va_list ap, parmN);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

stddef.h

```
offsetof(type, member_designator); /* macro */
```

stdio.h

```
void clearerr(FILE *stream);
int fclose(FILE *stream);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
char *fgets(char *s, int n, FILE *stream);
FILE *fopen(const char *filename, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *s, FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
FILE *freopen(const char *filename, const char *mode,
FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
int fseek(FILE *stream, long int offset, int ref_pt);
int fsetpos(FILE *stream, const fpos_t *pos);
long int ftell(FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *gets(char *s);
void perror(const char *s);
int printf(const char *format, ...);
int putc(int c, FILE *stream);
int putchar(int c);
int puts(const char *s);
int remove(const char *filename);
int rename(const char *old, const char *new);
void rewind(FILE *stream);
int scanf(const char *format, ...);
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
int sprintf(char *s, const char *format, ...);
int sscanf(const char *s, const char *format, ...);
FILE *tmpfile(void);
char *tmpnam(char *s);
int ungetc(int c, FILE *stream);
int vfprintf(FILE *stream, const char *format, va_list arg);
int vprintf(const char *format, va_list arg);
int vsprintf(char *s, const char *format, va_list arg);
```

stdlib.h

```

void abort(void);
int atexit(void (*func) (void));
double atof(const char *nptr);
int atoi(const char *nptr);
int atol(const char *nptr);
int abs(int j);
void *bsearch(const void *key, const void *base, size_t nmemb,
              size_t size, int (*compar) (const void *, const void *));
void *calloc(size_t nmemb, size_t size);
div_t div(int numer, int denom);
void exit(int status);
void free(void *ptr);
char *getenv(const char *name);
long int labs(long int j);
ldiv_t ldiv(long int numer, long int denom);
void *malloc(size_t size);
int mblen(const char *s, size_t n);
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
int mbtowc(wchar_t *pwc, const char *s, size_t n);
void qsort(void *base, size_t nmemb, size_t size,
            int (*compar) (const void *, const void *));
int rand(void);
void *realloc(void *ptr, size_t size);
void srand(unsigned int seed);
double strtod(const char *nptr, char **endptr);
long int strtol(const char *nptr, char **endptr, int base);
unsigned long int strtoul(const char *nptr, char **endptr, int base);
int system(const char *string);
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
int wctomb(char *s, wchar_t wchar);

```

string.h

```
void *memcpy(void *s1, const void *s2, size_t n);
void *memmove(void *s1, const void *s2, size_t n);
char *strcpy(char *s1, const char *s2);
char *strncpy(char *s1, const char *s2, size_t n);
char *strcat(char *s1, const char *s2);
char *strncat(char *s1, const char *s2, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
int strcmp(const char *s1, const char *s2);
int strcoll(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
size_t strxfrm(char *s1, const char *s2, size_t n);
void *memchr(const void *s, int c, size_t n);
char *strchr(const char *s, int c);
size_t strcspn(const char *s1, const char *nset);
char *strpbrk(const char *s1, const char *set);
char *strrchr(const char *s, int c);
size_t strspn(const char *s1, const char *set);
char *strstr(const char *s1, const char *pattern);
char *strtok(char *s1, const char *delimiters);
void *memset(void *s, int c, size_t n);
char *strerror(int errnum);
size_t strlen(const char *s);
```

time.h

```
clock_t clock(void);
double difftime(time_t time1, time_t time0);
time_t mktime(struct tm *timeptr);
time_t time(time_t *time);
char *asctime(const struct tm *timeptr);
char *ctime(const time_t *timer);
struct tm *gmtime(const time_t *timer);
struct tm *localtime(const time_t *timer);
size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timeptr );
```


