QUICK REFERENCE

CONVEX C V5.0 Quick Reference

Order No. DSW-087

Sixth Edition
January 1993

CONVEX COMPUTER CORPORATION

CONVEX C V5.0
Quick Reference

Order No. DSW-087
Sixth Edition

Released with CONVEX C V5.0

Printed in the United States of America

cd

bz -n alex -rel

H = Hilfsbildschirm

Contents

606

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.

`-D`*name*[`=`*def*]
Define *name* as if by `#define`.

`-E`
Run the preprocessor only.

`-I`*dir*
Search alternate directory for `#include` files.

`-I-`
Inhibits current directory for first search directory for `#include` files.

`-k`
Generate dependency descriptions for make.

`-P`
Gives preprocessed code without control lines.

`-U`*name*
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 restrict_args`
Instructs the compiler to treat the code as though a `restrict` qualifier was applied to each pointer parameter.

`-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.

`-nopeel`
Disallows loop boundary value peeling.

`-noptst`
Disallows test promotion.

`-nptr`
Disable pointer tracking.

`-peel`
Causes removal of first/last iterations of loop.

`-peelall`
Same as `-peel`.

`-ptst`
Causes test to be promoted out of enclosing loop by replicating containing loop for each branch.

`-ptstall`
Same as `-ptst`.

`-O`
Perform highest scalar optimization.

`-O0`
Perform local scalar optimization.

`-O1`
Perform global scalar optimization.

`-O2`
Perform vectorization.

`-O3`
Perform automatic parallelization.

`-rl`
Perform loop replication.

`-uo`
Perform potentially unsafe optimizations.

`-ur`
Perform loop unrolling.

`-va`
A synonym for `-alias array_args`.

Code-generation options

`-c`
Suppress linking.

`-compat rrf=stack`
Controls backward compatibility options. Calling function allocates memory for record being returned.

`-compat rrf=old`
Controls backward compatibility options. Called function allocates memory for record being returned.

`-except precise`
Generates code that ensures that any arithmetic exceptions generated within functions before the return will be received by the program.

`-except default`
Cancels the effect of `-except precise`.

`-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`
Unsuffixed floating-point constants are double.

`-float dp_ops`
Floating-point operations use double precision.

`-float sp_const`
Unsuffixed floating-point constants are float.

`-float sp_ops`
Floating-point operations use float precision.

`-fn`
Use native floating-point format.

`-mi` *n*
Specifies expected memory interleave on target machine.

`-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.

`-sc`
Check program for compilation errors. No optimization, vectorization, or code generation performed.

`-string read_only`
String literals cannot be modified.

`-string read_write`
String literals can be modified.

`-tm` *target*
Generate code for target machine where *target* = `C1`, `C2`, `C3`, `C34`, `C34j`, or `C38`.

Compatibility options

`-ext`
Compile ANSI C, POSIX, and CONVEX extensions. This is the default.

`-std`
Compile in ANSI C and POSIX mode.

`-str`
Compile in 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`.

`-metrics`
Produces source metrics and cross reference information for CXmetrics.

`-p`
Compile for profiling with `prof`.

`-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.

`-pb`
Compile for profiling with `bprof`.

`-pg`
Compile for profiling with `gprof`.

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.

`-nv`
Suppresses vectorization summary messages.

`-or {array|loop|none}`
Produce optimization report on loops, arrays, all, or none.

`-sc`
Syntax check only.

`-w` or `-nw`
Suppresses warning diagnostics.

Miscellaneous options

`-altcc` *path*
Invokes alternate compiler driver.

`-B`*directory*
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.

Language description

Preprocessor statements

`#define` *identifier token-string*
Replace *identifier* with *token-string*.

`#define` *identifier*`(`*identifier*`, ..., `*identifier*`)` *token-string*
Replace *identifier* with expanded macro.

`#error "`*message*`"`
Produce compile-time error message.

`#include "`*filename*`"`
Replace this line with contents of *filename*.

`#include <`*filename*`>`
Replace this line with contents of /usr/include/*filename*.

`#include` *token*
Replace this line with contents of file defined by *token*.

`#line` *n identifier*
Next source line is *n*; current input file is *identifier*.

`#if` *const_expression*
Compile if *const_expression* true.

`#else`
Compile if previous `if` condition is false.

`#elif` *const_expression*
Else-if. Compile if previous `if` condition is false and `elif` condition is true.

`#endif`
Terminate conditional compile.

`#ifdef` *identifier*
Compile if *identifier* is defined.

`#ifndef` *identifier*
Compile if *identifier* is not defined.

`defined(`*name*`)`
True if *name* is defined; otherwise false.

`#pragma` *compiler_info*
Pass implementation-defined information to the compiler.

`#undef` *identifier*
Undo previous define.

`#` *blank line*
Null statement.

`#` *string*
Convert the unexpanded parameter to a string and replace.†

tkn1 `##` *tkn2*
Concatenate unexpanded tokens.†

† Used only in macro definitions.

Pragmas and directives

General format:

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

`/* $dir` *pragma-name* [*options*] `*/`

pragma_name can be:

`begin_tasks`
Begin task group.

`end_tasks`
End task group.

`force_parallel`
Parallelize loop. Does not interchange outer loop for vectorization.

`force_parallel_ext`
Parallelize loop. Can interchange outer loop for vectorization.

`force_vector`
Vectorize loop.

`max_trips (`*n*`)`
Specify maximum trip count.

`next_task`
Begin individual task.

`no_parallel`
Do not parallelize loop.

`no_recurrence`
Ignore apparent recurrences.

`no_side_effects` (*f* [,*f*]...)
Functions have no side effects.

`no_vector`
Do not vectorize loop.

`prefer_parallel`
Prefer to parallelize loop.

`prefer_parallel_ext`
Perform `force_parallel_ext` if safe.

`prefer_vector`
Prefer to vectorize loop.

`pstrip` (*n*)
Specify parallel strip-mine length.

`returns_unique_pointer` (*f* [,*f*]...)
Tells the compiler that the pointer returned by a funtion points to a memory location that no other pointer references.

`scalar`
Process loop as scalar.

`select` (*v*, *p*, *pv*)
Generate and dynamically select scalar, vector, parallel, and pararallel vector versions of a loop, based on specified trip count.

`synch_parallel`
Parallelize loop and insert synchronization code to honor dependencies.

`unroll`
Unroll the following loop.

`vstrip` (*n*)
Specify vector strip-mine length.

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`
`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)
{
    case constant_exp: statement
    ...
    default: statement
}
```

Evaluate (*exp*) and go to matching `case` label. Go to `default` if no `case` matched

`goto` *label*;
Go to labeled statement.

label: *statement*
Define statement as target for `goto`.

`break;`
End smallest enclosing `while`, `do`, `for`, or `switch`.

`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 characters to null-terminated string.

`struct word {`
`  char first [10];`
`  char last[20];`
`  unsigned bit: 1;`
`} letter;`
	Define structure.
	
	Declare bit-field "bit" 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 p`	Constant pointer to integer.
`const int *const p`	Constant pointer to constant integer.
`volatile int i`	Volatile integer.
`int *volatile p`	Volatile pointer to integer.
`volatile int *volatile p`	Volatile pointer to volatile integer.
`const volatile int i`	Constant volatile integer—unchangeable by program, but changeable by external forces.
`double *restrict p`	Restricted pointer to double.

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"`	Null-terminated character 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 (grouped by precedence)

Description	Operator	Associativity
Subscript	`X[Y]`	left to right
Function call	`X(Y)`	left to right
Select member	`X.Y`	left to right
Point at member	`X->Y`	left to right
Sizeof	`sizeof X`	right to left
Postincrement	`X++`	right to left
Postdecrement	`X--`	right to left
Preincrement	`++X`	right to left
Predecrement	`--X`	right to left
Address of	`&X`	right to left
Indirection	`*X`	right to left
Plus	`+X`	right to left
Minus	`-X`	right to left
Bitwise NOT	`~X`	right to left
Logical NOT	`!X`	right to left
Type cast	(*type*)`X`	right to left
Multiply	`X*Y`	left to right
Divide	`X/Y`	left to right
Remainder	`X%Y`	left to right
Add	`X+Y`	left to right
Subtract	`X-Y`	left to right
Left shift	`X<<Y`	left to right
Right shift	`X>>Y`	left to right
Less than	`X<Y`	left to right
Less than or equal	`X<=Y`	left to right
Greater than	`X>Y`	left to right

Description	Operator	Associativity
Greater than or equal	`X>=Y`	left to right
Equals	`X==Y`	left to right
Not equals	`X!=Y`	left to right
Bitwise AND	`X&Y`	left to right
Bitwise exclusive OR	`X^Y`	left to right
Bitwise inclusive OR	`X|Y`	left to right
Logical AND	`X&&Y`	left to right
Logical OR	`X||Y`	left to right
Conditional	`Z?X:Y`	right to left
Assignment	`X=Y`	right to left
Multiply assign	`X*=Y`	right to left
Divide assign	`X/=Y`	right to left
Remainder assign	`X%=Y`	right to left
Add assign	`X+=Y`	right to left
Subtract assign	`X-=Y`	right to left
Left shift assign	`X<<=Y`	right to left
Right shift assign	`X>>=Y`	right to left
Bitwise AND assign	`X&=Y`	right to left
Bitwise exclusive OR assign	`X^=Y`	right to left
Bitwise inclusive OR assign	`X|=Y`	right to left
Comma	`X,Y`	left to right

Input/output

Unformatted I/O

```
FILE *fopen(const char *filename,
   const char *mode);†

FILE *freopen(const char *filename, const
   char *mode, FILE *fp);†

int fclose(FILE *file);

size_t fread(void *ptr, size_t size,
   size_t nmemb, 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 *str);

char *fgets(char *s, int n, FILE *fp);

int putc(int ch, FILE *fp);

int putchar(int ch);

int puts(const char *str);
```

†Refer to the access mode table for mode values.

Formatted I/O

```
int printf(const char *format, ...);

int fprintf(FILE *fp,
   const char *format, ...);

int vprintf(const char *format, va_list arg);

int sprintf(char *str,
   const char *format, ...);

int scanf(const char *format, ...);

int fscanf(FILE *fp,
   const char *format, ...);
```

```
int sscanf(const char *str,
   const char *format, ...);
```

Access modes

Mode string	Meaning
`'r'`	Open a text file for reading. Fails if the file does not exist.
`'w'`	If the file exists, open and truncate it to zero length. Otherwise, create the file and open it for writing in the text mode.
`'a'`	Open a text file for appending–writing at the end of the file. Create the file if it does not exist.
`'rb'`	Same as `'r'`–but in binary mode.
`'wb'`	Same as `'w'`–but in binary mode.
`'ab'`	Same as `'a'`–but in binary mode.
`'r+'`	Open a text file for updating–reading as well as writing.
`'w+'`	If the file exists, open it and truncate it to zero length. Otherwise, create it and open it for updating.
`'a+'`	Open a text file for appending. Create the file if it does not exist.
`'r+b'` or `'rb+'`	Open a binary file for updating.
`'w+b'` or `'wb+'`	If the file exists, truncate it to zero length. Otherwise, create a binary file for update operations.
`'a+b'` or `'ab+'`	Open or create a binary file for appending.

Output format specifiers (`printf` and variants)

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

Flags

-	Left justify.
+	Use + sign for positive values.
<blank>	Use space for positive values.
#	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 integer.†
`LL`	Long long integer.†

Conversion characters

`c`	Unsigned character.
`d`	Signed decimal integer.
`e`	Double (scientific notation).
`E`	Double (scientific notation).
`f`	Double (fixed-point notation).
`g`	`%e` or `%f`.
`G`	`%E` or `%f`.
`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 as `int*`.
`o`	Unsigned octal integer.
`p`	Pointer value.
`s`	Null-terminated string.
`u`	Unsigned decimal integer.
`x`	Unsigned hex integer.

† CONVEX extensions.

Input format specifiers (`scanf` and variants)

`%` [*Flag*] [*Width*] [*Prec*] [*Length*] <*conversion character*>

Flag

*	Suppress assignment.

Width

W	Minimum field width in *W* digits.

Length

h	Short integer or unsigned short integer.
l	Long integer.
L	Long double.
ll	Long long integer.†
LL	Long long integer.†

Conversion characters

c	Reads single character or sequence of *W* characters into `char*` or `char[]` argument.
d	Reads signed decimal integer into `int*` arg.
e,f,g,E,F	Reads floating-point number in scientific notation into `float*`.
h	Short integer, 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.

Character escape sequences

`\a`	Alert	`\r`	Carriage return
`\b`	Backspace	`\t`	Horizontal tab
`\f`	Form feed	`\v`	Vertical tab
`\n`	Newline		
`\'`	Single quote	`\"`	Double quote
`\\`	Backslash	`\?`	Question mark

`\ddd` Octal character constant `\xdd` Hex character constant

ANSI C library

assert.h

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

ctype.h

```
int isalnum(int ch);

int isalpha(int ch);

int iscntrl(int ch);

int isdigit(int ch);

int isgraph(int ch);

int islower(int ch);

int isprint(int ch);

int ispunct(int ch);

int isspace(int ch);

int isupper(int ch);

int isxdigit(int ch);

int tolower(int ch);

int toupper(int ch);
```

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 ceil(double x);

double cos(double x);

double cosh(double x);

double exp(double x);

double fabs(double x);

double floor(double x);

double fmod(double x,double y);

double frexp(double x, int *pow);

double ldexp(double x, int pow);

double log(double x);

double log10(double x);

double modf(double value, double *iptr);

double pow(double x, double pow);

double sin(double x);

double sinh(double x);

double sqrt(double x);

double tan(double x);

double tanh(double x);
```

setjmp.h

```
int setjmp(jmp_buf buf);

void longjmp(jmp_buf buf, int retval);
```

signal.h

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

int raise( int signal );
```

stdarg.h

```
void va_start( va_list list, LastParam );

type va_arg( va_list list, type );

void va_end( va_list list );
```

stddef.h

```
offsetof( structure, field); /* macro */
```

stdio.h

```
void clearerr(FILE *fp);

int fclose(FILE *file);

int feof(FILE *fp);

int ferror(FILE *fp);

int fflush(FILE *fp);

int fgetc(FILE *fp);

int fgetpos(FILE *fp, fpos_t *pos);

char *fgets(char *str, int n, FILE *fp);

FILE *fopen(const char *filename,
   const char *mode);

int fprintf(FILE *fp,
   const char *format, ...);

int fputc(int ch, FILE *fp);

int fputs(const char *str, FILE *fp);

size_t fread(void *ptr, size_t size,
   size_t nmemb, FILE *stream);

FILE *freopen(const char *filename, const
   char *mode, FILE *fp);

int fscanf(FILE *fp,
   const char *format, ...);

int fseek(FILE *fp, long int offset,
   int ref_pt);
```

```
int fsetpos(FILE *fp,
   const fpos_t *pos);

long int ftell(FILE *fp);

size_t fwrite(const void *ptr,
   size_t size, size_t nmemb, FILE *stream);

int getc(FILE *stream);

int getchar(void);

char *gets(char *str);

void perror(const char *str);

int printf(const char *format, ...);

int putc(int ch, FILE *fp);

int putchar(int ch);

int puts(const char *str);

int remove(const char *filename);

int rename(const char *old,
   const char *new);

void rewind(FILE *fp);

int scanf(const char *format, ...);

void setbuf(FILE *fp, char *buf);

int setvbuf(FILE *fp, char *buf,
   int buf_type, size_t size);

int sprintf(char *str,
   const char *format, ...);

int sscanf(const char *str,
   const char *format, ...);

FILE *tmpfile(void);

char *tmpnam(char *str);

int ungetc(int ch, FILE *fp);

int vfprintf(FILE *fp,
   const char *format, va_list arg);
```

```
int vprintf(const char *format,
   va_list arg);

int vsprintf(char *str, 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(consts 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 *str, size_t n);

size_t mbstowcs(wchar_t *pwcs,
   const char *str, size_t n);

int mbtowc(wchar_t *pwc, const char *str,
   size_t n);
```

```
void qsort(void *base, size_t nmemb,
   size_t size,
   int (*compare) (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 *str,
   const wchar_t *wcs, size_t n);

int wctomb(char *str, wchar_t wchar);
```

string.h

```
void *memcpy(void *str1, const void *str2,
   size_t n);

void *memmove(void *str1, const void *str2,
   size_t n);

char *strcpy(char *str1, const char *str2);

char *strncpy(char *str1, const char *str2,
   size_t n);

char *strcat(char *str1, const char *str2);

char *strncat(char *str1, const char *str2,
   size_t n);

int memcmp(const void *buf1,
   const void *buf2, size_t n);

int strcmp(const char *str1,
   const char *str2);

int strcoll(const char *str1,
   const char *str2);
```

```
int strncmp(const char *str1,
   const char *str2, size_t n);

size_t strxfrm(char *str1, const char *str2,
   size_t n);

void *memchr(const void *str, int ch,
   size_t n);

char *strchr(const char *str, int ch);

size_t strcspn(const char *str1,
   const char *nset);

char *strpbrk(const char *str1,
   const char *set);

char *strrchr(const  char *str, int ch);

size_t strspn(const char *str1,
   const char *set);

char *strstr(const char *str1,
   const char *pattern);

char *strtok(char *str1,
   const char *delimiters);

void *memset(void *buf, int ch, size_t n);

char *strerror(int errnum);

size_t strlen(const char *str);
```

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 *timer);

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 *str, size_t maxsize,
   const char *format,
   const struct tm *timeptr );
```

CONVEX Computer Corporation
Richardson, Texas USA

Document No. 720-003030-003