Constants

1234	Integer decimal.
0Xaa55	Integer hexadecimal.
0177	Integer octal.
'a'	Character
1234L	Long decimal.
0XAA55L	Long hexadecimal.
0177L	Long octal.
1234LL	Long long decimal.
0Xaa55LL	Long long hexadecimal.
0177LL	Long long octal.
32.5	Long floating point.
1.2e-5	Long floating point with exponent (e or E).
"abcd"	Address of null-terminated char string.

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

Intrinsic Types

char a;	Signed, one byte (8 bits).
int i,j,k;	Signed integers (32 bits).
long sum;	Signed long integer (32 bits).
long long sum;	Signed long long integer (64 bits).
short x,y;	Signed short integers (16 bits).
unsigned limit=0xffff;	Unsigned integer, initialized.
float eps;	Floating point (32 bits).
double big;	Long floating point (64 bits).
char *ptr;	Variable pointer to char data (32 bits).
void func();	Function returns no value.

Note: short int, long int, long long int, unsigned int, long float, and unsigned char are also valid.

Type Constructors

char msg[ ]="testing\n";	Initialized array with zero terminator.

struct word {
  char first [10];
  char last[20];
  unsigned case: 1;
} letter;
	Define complex data type.
	
	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.

Allocation Classes

register short quick;	Specify that the variable is used often.
extern int flag, open();	Variable and function defined elsewhere.
static char arg;	Local permanent storage.
auto long arg;	Dynamic storage (default for all but externs and statics).

Special Characters

\n	newline	\t	horizontal tab
\b	backspace	\r	carriage return
\f	form feed	\\	backslash
\ddd	octal constant	\'	single quote
\"	double quote	\0	null

Operators

Operator	Associativity
( ) [ ] function array element -> . structure pointer structure member	left to right
- (datatype) * & sizeof minus cast indirect address ! ~ ++ -- not 1's comp increment decrement	right to left
* / % multiply divide modulus	left to right
+ - add subtract	left to right
<< >> shift left shift right	left to right
< <= > >= relational operators	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.

Formatted I/O

printf (control, arg1, arg2, ...)	To standard output.
fprintf (stream, control, arg1, arg2,...)	To specified stream.
sprintf (string, control, arg1, arg2, ...)	To string buffer.
scanf (control, &arg1, &arg2, ...)	From standard input.
fscanf (stream, control, &arg1, &arg2, ...)	From specified stream.
sscanf (string, control, &arg1, &arg2, ...)	From string buffer.

Format Specifier

% [-] [* | W] [.M | .*] [L [L]] <conversion character>

notation

-	Left justify (output only).
*	Get width from argument list.
W	Width (leading 0 means zero pad to left).
.M	Precision (for output).
.*	Get precision from argument list.
L	Long integer or double.
LL	Long integer or double.

conversion characters

c	Single character.
d	Signed decimal integer.
e	Scientific notation for float or double (output only).
f	Fixed-point notation for float or double.
g	Use %e or %f, whichever is shorter (output only).
h	Unsigned hexadecimal short integer (input only).
o	Unsigned octal integer.
s	Null terminated string.
u	Unsigned decimal integer (input only).
x	Unsigned hexadecimal integer.

Low-Level I/O Calls

```
Unless specified below, arguments and return values are integer.
    char buffer[ ], ch, *name, *ptr, *s_mode;
    long offset;
    struct stat *stat_buf;
    FILE *stream;
close (filedes);
creat (name, mode);
filedes = open (name, mode, flags);
   (where mode = 0:read,  1:write,  2:both)
long lseek (filedes, offset, from);
   (where from = 0:beginning, 1:current, 2:end)
read (filedes, buffer, count);
write (filedes, buffer, count);
```

Standard I/O Library Calls

```
char * gets (buffer);  NULL at end of file; deletes newlines
fclose(stream);
fgets (buffer,n,stream); NULL at end; keeps newlines
FILE * fopen (name, S_mode); (mode is "r" for read; "w" for write;
  "a" for appending; "r+" for read/write, start at beginning;
  "w+" for read/write, truncate and start anew; "a+" read/write,
  start at end)
FILE * fdopen (filedes, S_mode);
FILE * freopen (name, S_mode, stream);
frewind (stream);
fseek (stream, offset, from);
   (where from = 0:beginning, 1:current, 2:end)
fwrite (ptr, item_size, count, stream);
getc (stream); EOF at end
getchar (); EOF at end
putc (ch,stream);
putchar (ch);
puts (buffer);
```

CONVEX

CONVEX C Language Reference Card
Document No. 720-000152-200

Statement Formats

/* comment */	Comments.
expression;	Simple statement.
{statement; statement;...}	Compound statement.
if (expression) statement;	Do statement if expression nonzero.
if (exp) stmt1 else stmt2;	If exp true, do stmt1, otherwise, do stmt2.
while (expression) statement;	Do statement while expression is nonzero.
do statement while (expression);	Do statement while expression true.
for (exp1; exp2; exp3) statement;	Initialize with exp1, do statement, increment by exp3 while exp2 true.

switch (expression)
{
  case constant_exp: statement;
  ...
  default: statement;
}
	Evaluate expression and go to correct case statement.
	
	Default if no case matched.

break;	End smallest enclosing *while*, *do*, *for*, or *switch*.
continue;	Go to bottom of *while*, *do*, or *for* loop.
return expression;	Exit function and return optional expression.
goto label;	Go to label.
label: statement	Define label.
;	Null statement.

Preprocessor Statements

#define identifier token-string	Replace identifier with token-string.
#define identifier (identifier, ..., identifier) token-string	Replace identifier with macro.
#include "filename"	Replace this line with contents of filename.
#include <filename>	Replace this line with contents of /usr/include/filename.
#line n identifier	Next source line is n; current input file is identifier.
#if constant_expression	Compile if constant_expression true.
#else	Compile when previous *if* condition false.
#endif	Terminate conditional compile.
#ifdef identifier	Compile if identifier is defined.
#ifndef identifier	Compile if identifier is not defined.
#undef identifier	Undo previous define.