```
**********************************************************
    8630 Fri Jan  3 08:20:33 2014
new/usr/src/cmd/sgs/lex/common/main.c
2926 lex ignores -Y
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2014 Gary Mills
 *
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* Copyright (c) 1988 AT&T */
/* All Rights Reserved */

/* Copyright 1976, Bell Telephone Laboratories, Inc. */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <string.h>
#include "once.h"
#include "sgs.h"
#include <locale.h>
#include <limits.h>

static wchar_t  L_INITIAL[] = {'I', 'N', 'I', 'T', 'I', 'A', 'L', 0};
static void get1core(void);
static void free1core(void);
static void get2core(void);
static void free2core(void);
static void get3core(void);
#ifdef DEBUG
static void free3core(void);
#endif

int
main(int argc, char **argv)
{
        int i;
        int c;
        char *apath = NULL;
        char *ypath;
        char *path = NULL;
        Boolean eoption = 0, woption = 0;

        sargv = argv;
```

```
        sargc = argc;
        (void) setlocale(LC_ALL, "");
#ifdef DEBUG
        while ((c = getopt(argc, argv, "dyctvnewVQ:Y:")) != EOF) {
#else
        while ((c = getopt(argc, argv, "ctvnewVQ:Y:")) != EOF) {
#endif
                switch (c) {
#ifdef DEBUG
                        case 'd':
                                debug++;
                                break;
                        case 'y':
                                yydebug = TRUE;
                                break;
#endif
                        case 'V':
                                (void) fprintf(stderr, "lex: %s %s\n",
                                    (const char *)SGU_PKG,
                                    (const char *)SGU_REL);
                                break;
                        case 'Q':
                                v_stmp = optarg;
                                if (*v_stmp != 'y' && *v_stmp != 'n')
                                        error(
                                        "lex: -Q should be followed by [y/n]");
                                break;
                        case 'Y':
                                apath = (char *)malloc(strlen(optarg) +
                                path = (char *)malloc(strlen(optarg) +
                                    sizeof ("/nceucform") + 1);
                                if (apath == NULL)
                                        error("No available memory "
                                            "for directory name.");
                                else
                                        apath = strcpy(apath, optarg);
                                path = strcpy(path, optarg);
                                break;
                        case 'c':
                                ratfor = FALSE;
                                break;
                        case 't':
                                fout = stdout;
                                break;
                        case 'v':
                                report = 1;
                                break;
                        case 'n':
                                report = 0;
                                break;
                        case 'w':
                        case 'W':
                                woption = 1;
                                handleeuc = 1;
                                widecio = 1;
                                break;
                        case 'e':
                        case 'E':
                                eoption = 1;
                                handleeuc = 1;
                                widecio = 0;
                                break;
                        default:
                                (void) fprintf(stderr,
                                "Usage: lex [-ewctvnV] [-Y directory] "
                                "[-Q(y/n)] [file]\n");
                                "Usage: lex [-ewctvnVY] [-Q(y/n)] [file]\n");
                                exit(1);
                }
        }
        if (woption && eoption) {
                error(
                "You may not specify both -w and -e simultaneously.");
        }
        no_input = argc - optind;
        if (no_input) {
                /* XCU4: recognize "-" file operand for stdin */
                if (strcmp(argv[optind], "-") == 0)
                        fin = stdin;
                else {
                        fin = fopen(argv[optind], "r");
                        if (fin == NULL)
                                error(
                                "Can't open input file -- %s", argv[optind]);
                }
        } else
                fin = stdin;

        /* may be gotten: def, subs, sname, schar, ccl, dchar */
        (void) gch();

        /* may be gotten: name, left, right, nullstr, parent */
        get1core();

        scopy(L_INITIAL, sp);
        sname[0] = sp;
        sp += slength(L_INITIAL) + 1;
        sname[1] = 0;

        /* XCU4: %x exclusive start */
        exclusive[0] = 0;

        if (!handleeuc) {
                /*
                 * Set ZCH and ncg to their default values
                 * as they may be needed to handle %t directive.
                 */
                ZCH = ncg = NCH; /* ncg behaves as constant in this mode. */
        }

        /* may be disposed of: def, subs, dchar */
        if (yyparse())
                exit(1);        /* error return code */

        if (handleeuc) {
                ncg = ncgidtbl * 2;
                ZCH = ncg;
                if (ncg >= MAXNCG)
                        error(
                        "Too complex rules -- requires too many char groups.");
                sortcgidtbl();
        }
        repbycgid(); /* Call this even in ASCII compat. mode. */

        /*
         * maybe get:
         *              tmpstat, foll, positions, gotof, nexts,
         *              nchar, state, atable, sfall, cpackflg
         */
        free1core();
        get2core();
        ptail();
```

```
        mkmatch();
#ifdef DEBUG
        if (debug)
                pccl();
#endif
        sect  = ENDSECTION;
        if (tptr > 0)
                cfoll(tptr-1);
#ifdef DEBUG
        if (debug)
                pfoll();
#endif
        cgoto();
#ifdef DEBUG
        if (debug) {
                (void) printf("Print %d states:\n", stnum + 1);
                for (i = 0; i <= stnum; i++)
                        stprt(i);
        }
#endif
        /*
         * may be disposed of:
         *              positions, tmpstat, foll, state, name,
         *              left, right, parent, ccl, schar, sname
         * maybe get:    verify, advance, stoff
         */
        free2core();
        get3core();
        layout();
        /*
         * may be disposed of:
         *              verify, advance, stoff, nexts, nchar,
         *              gotof, atable, ccpackflg, sfall
         */

#ifdef DEBUG
        free3core();
#endif

        if (handleeuc) {
                if (ratfor)
                        error("Ratfor is not supported by -w or -e option.");
                ypath = EUCNAME;
                path = EUCNAME;
        }
        else
                ypath = ratfor ? RATNAME : CNAME;
                path = ratfor ? RATNAME : CNAME;

        if (apath != NULL)
                ypath = strcat(apath, strrchr(ypath, '/'));
        fother = fopen(ypath, "r");
        fother = fopen(path, "r");
        if (fother == NULL)
                error("Lex driver missing, file %s", ypath);
                error("Lex driver missing, file %s", path);
        while ((i = getc(fother)) != EOF)
                (void) putc((char)i, fout);
        (void) fclose(fother);
        (void) fclose(fout);
        free(apath);
        if (report == 1)
                statistics();
        (void) fclose(stdout);
        (void) fclose(stderr);
        return (0);     /* success return code */
```

```
250 }
______unchanged_portion_omitted_
```

```
**********************************************************
   32535 Fri Jan  3 08:20:33 2014
new/usr/src/man/man1/lex.1
2926 lex ignores -Y
**********************************************************
'\" te
.\" Copyright (c) 2014 Gary Mills
.\"  Copyright (c) 1992, X/Open Company Limited  All Rights Reserved  Portions C
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH LEX 1 "Jan 1, 2014"
.TH LEX 1 "Aug 22, 1997"
.SH NAME
lex \- generate programs for lexical tasks
.SH SYNOPSIS
.LP
.nf
\fBlex\fR [\fB-cntv\fR] [\fB-e\fR | \fB-w\fR] [\fB-V\fR \fB-Q\fR [y | n]] [\fB-Y
\fBlex\fR [\fB-cntv\fR] [\fB-e\fR | \fB-w\fR] [\fB-V\fR \fB-Q\fR [y | n]] [\fIfi
.fi

.SH DESCRIPTION
.sp
.LP
The \fBlex\fR utility generates C programs to be used in lexical processing of
character input, and that can be used as an interface to \fByacc\fR. The C
programs are generated from \fBlex\fR source code and conform to the ISO C
standard. Usually, the \fBlex\fR utility writes the program it generates to the
file \fBlex.yy.c\fR. The state of this file is unspecified if \fBlex\fR exits
with a non-zero exit status. See \fBEXTENDED DESCRIPTION\fR for a complete
description of the \fBlex\fR input language.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-c\fR \fR
.ad
.RS 12n
Indicates C-language action (default option).
.RE

.sp
.ne 2
.na
\fB\fB-e\fR \fR
.ad
.RS 12n
Generates a program that can handle \fBEUC\fR characters (cannot be used with
the \fB-w\fR option). \fByytext[\|]\fR is of type \fBunsigned char[\|]\fR.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR \fR
.ad
.RS 12n
Suppresses the summary of statistics usually written with the \fB-v\fR option.
```

```
If no table sizes are specified in the \fBlex\fR source code and the \fB-v\fR
option is not specified, then \fB-n\fR is implied.
.RE

.sp
.ne 2
.na
\fB\fB-t\fR \fR
.ad
.RS 12n
Writes the resulting program to standard output instead of \fBlex.yy.c\fR.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR \fR
.ad
.RS 12n
Writes a summary of \fBlex\fR statistics to the standard error. (See the
discussion of \fBlex\fR table sizes under the heading \fBDefinitions in
lex\fR.) If table sizes are specified in the \fBlex\fR source code, and if the
\fB-n\fR option is not specified, the \fB-v\fR option may be enabled.
.RE

.sp
.ne 2
.na
\fB\fB-w\fR \fR
.ad
.RS 12n
Generates a program that can handle \fBEUC\fR characters (cannot be used with
the \fB-e\fR option). Unlike the \fB-e\fR option, \fByytext[\|]\fR is of type
\fBwchar_t[\|]\fR.
.RE

.sp
.ne 2
.na
\fB\fB-V\fR \fR
.ad
.RS 12n
Prints out version information on standard error.
.RE

.sp
.ne 2
.na
\fB\fB\fR\fB-Q\fR\fB[y|n]\fR \fR
.ad
.RS 12n
Prints out version information to output file \fBlex.yy.c\fR by using
\fB-Qy\fR. The \fB-Qn\fR option does not print out version information and is
the default.
.RE

.sp
.ne 2
.na
\fB\fB\fR\fB-Y\fR \fBdirectory\fR \fR
.ad
.RS 12n
Designates an alternate directory that contains the driver files
used by \fBlex\fR.
.RE
```

```
.SH OPERANDS
.sp
.LP
The following operand is supported:
.sp
.ne 2
.na
\fB\fIfile\fR \fR
.ad
.RS 9n
A pathname of an input file. If more than one such \fIfile\fR is specified, all
files will be concatenated to produce a single \fBlex\fR program. If no
\fIfile\fR operands are specified, or if a \fIfile\fR operand is \fB\(mi\fR,
the standard input will be used.
.RE

.SH OUTPUT
.sp
.LP
The \fBlex\fR output files are described below.
.SS "Stdout"
.sp
.LP
If the \fB-t\fR option is specified, the text file of C source code output of
\fBlex\fR will be written to standard output.
.SS "Stderr"
.sp
.LP
If the \fB-t\fR option is specified informational, error and warning messages
concerning the contents of \fBlex\fR source code input will be written to the
standard error.
.sp
.LP
If the \fB-t\fR option is not specified:
.RS +4
.TP
1.
Informational error and warning messages concerning the contents of
\fBlex\fR source code input will be written to either the standard output or
standard error.
.RE
.RS +4
.TP
2.
If the \fB-v\fR option is specified and the \fB-n\fR option is not
specified, \fBlex\fR statistics will also be written to standard error. These
statistics may also be generated if table sizes are specified with a \fB%\fR
operator in the \fBDefinitions\fR \fBin\fR \fBlex\fR section (see \fBEXTENDED
DESCRIPTION\fR), as long as the \fB-n\fR option is not specified.
.RE
.SS "Output Files"
.sp
.LP
A text file containing C source code will be written to \fBlex.yy.c\fR, or to
the standard output if the \fB-t\fR option is present.
.SH EXTENDED DESCRIPTION
.sp
.LP
Each input file contains \fBlex\fR source code, which is a table of regular
expressions with corresponding actions in the form of C program fragments.
.sp
.LP
When \fBlex.yy.c\fR is compiled and linked with the \fBlex\fR library (using
the \fB\fR\fB-l\fR\fB l\fR operand with \fBc89\fR or \fBcc\fR), the resulting
program reads character input from the standard input and partitions it into
strings that match the given expressions.
```

```
.sp
.LP
When an expression is matched, these actions will occur:
.RS +4
.TP
.ie t \(bu
.el o
The input string that was matched is left in \fIyytext\fR as a null-terminated
string; \fIyytext\fR is either an external character array or a pointer to a
character string. As explained in \fBDefinitions in lex\fR, the type can be
explicitly selected using the \fB%array\fR or \fB%pointer\fR declarations, but
the default is \fB%array\fR.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The external \fBint\fR \fIyyleng\fR is set to the length of the matching
string.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The expression’s corresponding program fragment, or action, is executed.
.RE
.sp
.LP
During pattern matching, \fBlex\fR searches the set of patterns for the single
longest possible match. Among rules that match the same number of characters,
the rule given first will be chosen.
.sp
.LP
The general format of \fBlex\fR source is:
.sp
.in +2
.nf
\fIDefinitions\fR
%%
\fIRules\fR
%%
\fIUser Subroutines\fR
.fi
.in -2

.sp
.LP
The first \fB%%\fR is required to mark the beginning of the rules (regular
expressions and actions); the second \fB%%\fR is required only if user
subroutines follow.
.sp
.LP
Any line in the \fBDefinitions\fR \fBin\fR \fBlex\fR section beginning with a
blank character will be assumed to be a C program fragment and will be copied
to the external definition area of the \fBlex.yy.c\fR file. Similarly, anything
in the \fBDefinitions\fR \fBin\fR \fBlex\fR section included between delimiter
lines containing only \fB%{\fR and \fB%}\fR will also be copied unchanged to
the external definition area of the \fBlex.yy.c\fR file.
.sp
.LP
Any such input (beginning with a blank character or within \fB%{\fR and
\fB%}\fR delimiter lines) appearing at the beginning of the \fIRules\fR section
before any rules are specified will be written to \fBlex.yy.c\fR after the
declarations of variables for the \fByylex\fR function and before the first
line of code in \fByylex\fR. Thus, user variables local to \fByylex\fR can be
declared here, as well as application code to execute upon entry to
```

```
\fByylex\fR.
.sp
.LP
The action taken by \fBlex\fR when encountering any input beginning with a
blank character or within \fB%{\fR and \fB%}\fR delimiter lines appearing in
the \fIRules\fR section but coming after one or more rules is undefined. The
presence of such input may result in an erroneous definition of the \fByylex\fR
function.
.SS "Definitions in lex"
.sp
.LP
\fBDefinitions\fR \fBin\fR \fBlex\fR appear before the first \fB%%\fR
delimiter. Any line in this section not contained between \fB%{\fR and \fB%}\fR
lines and not beginning with a blank character is assumed to define a \fBlex\fR
substitution string. The format of these lines is:
.sp
.in +2
.nf
\fIname   substitute\fR
.fi
.in -2
.sp

.sp
.LP
If a \fIname\fR does not meet the requirements for identifiers in the ISO C
standard, the result is undefined. The string \fIsubstitute\fR will replace the
string \fI{\fR \fIname\fR \fI}\fR when it is used in a rule. The \fIname\fR
string is recognized in this context only when the braces are provided and when
it does not appear within a bracket expression or within double-quotes.
.sp
.LP
In the \fBDefinitions\fR \fBin\fR \fBlex\fR section, any line beginning with a
\fB%\fR (percent sign) character and followed by an alphanumeric word beginning
with either \fBs\fR or \fBS\fR defines a set of start conditions. Any line
beginning with a \fB%\fR followed by a word beginning with either \fBx\fR or
\fBX\fR defines a set of exclusive start conditions. When the generated scanner
is in a \fB%s\fR state, patterns with no state specified will be also active;
in a \fB%x\fR state, such patterns will not be active. The rest of the line,
after the first word, is considered to be one or more blank-character-separated
names of start conditions. Start condition names are constructed in the same
way as definition names. Start conditions can be used to restrict the matching
of regular expressions to one or more states as described in \fBRegular
expressions in lex\fR.
.sp
.LP
Implementations accept either of the following two mutually exclusive
declarations in the \fBDefinitions\fR \fBin\fR \fBlex\fR section:
.sp
.ne 2
.na
\fB\fB%array\fR \fR
.ad
.RS 13n
Declare the type of \fIyytext\fR to be a null-terminated character array.
.RE

.sp
.ne 2
.na
\fB\fB%pointer\fR \fR
.ad
.RS 13n
Declare the type of \fIyytext\fR to be a pointer to a null-terminated character
string.
.RE
```

```
.sp
.LP
\fBNote:\fR When using the \fB%pointer\fR option, you may not also use the
\fByyless\fR function to alter \fIyytext\fR.
.sp
.LP
\fB%array\fR is the default. If \fB%array\fR is specified (or neither
\fB%array\fR nor \fB%pointer\fR is specified), then the correct way to make an
external reference to \fIyyext\fR is with a declaration of the form:
.sp
.LP
\fBextern char\fR\fI yytext\fR\fB[\|]\fR
.sp
.LP
If \fB%pointer\fR is specified, then the correct external reference is of the
form:
.sp
.LP
\fBextern char *\fR\fIyytext\fR\fB;\fR
.sp
.LP
\fBlex\fR will accept declarations in the \fBDefinitions in lex\fR section for
setting certain internal table sizes. The declarations are shown in the
following table.
.sp
.LP
\fBTable\fR \fBSize\fR \fBDeclaration\fR \fBin\fR \fBlex\fR
.sp

.sp
.TS
box;
c c c
l l l .
\fBDeclaration\fR       \fBDescription\fR       \fBDefault\fR
_
\fB%p\fR\fIn\fR Number of positions     2500
\fB%n\fR\fIn\fR Number of states        500
\fB%a\fR\fI n\fR        Number of transitions   2000
\fB%e\fR\fIn\fR Number of parse tree nodes      1000
\fB%k\fR\fIn\fR Number of packed character classes      10000
\fB%o\fR\fIn\fR Size of the output array        3000
.TE

.sp
.LP
Programs generated by \fBlex\fR need either the \fB-e\fR or \fB-w\fR option to
handle input that contains \fBEUC\fR characters from supplementary codesets. If
neither of these options is specified, \fByytext\fR is of the type
\fBchar[\|]\fR, and the generated program can handle only \fBASCII\fR
characters.
.sp
.LP
When the \fB-e\fR option is used, \fByytext\fR is of the type \fBunsigned\fR
\fBchar[\|]\fR and \fByyleng\fR gives the total number of \fIbytes\fR in the
matched string. With this option, the macros \fBinput()\fR,
\fBunput(\fIc\fR)\fR, and \fBoutput(\fIc\fR)\fR should do a byte-based
\fBI/O\fR in the same way as with the regular \fBASCII\fR \fBlex\fR. Two more
variables are available with the \fB-e\fR option, \fByywtext\fR and
\fByywleng\fR, which behave the same as \fByytext\fR and \fByyleng\fR would
under the \fB-w\fR option.
.sp
.LP
When the \fB-w\fR option is used, \fByytext\fR is of the type \fBwchar_t[\|]\fR
and \fByyleng\fR gives the total number of \fIcharacters\fR in the matched
```

```
string.  If you supply your own \fBinput()\fR, \fBunput(\fIc\fR)\fR, or
\fBoutput(\fR\fIc\fR\fB)\fR macros with this option, they must return or accept
\fBEUC\fR characters in the form of wide character (\fBwchar_t\fR). This allows
a different interface between your program and the lex internals, to expedite
some programs.
.SS "Rules in lex"
.sp
.LP
The \fBRules\fR \fBin\fR \fBlex\fR source files are a table in which the left
column contains regular expressions and the right column contains actions (C
program fragments) to be executed when the expressions are recognized.
.sp
.in +2
.nf
\fIERE action\fR
\fIERE action\fR
\&...
.fi
.in -2

.sp
.LP
The extended regular expression (ERE) portion of a row will be separated from
\fIaction\fR by one or more blank characters. A regular expression containing
blank characters is recognized under one of the following conditions:
.RS +4
.TP
.ie t \(bu
.el o
The entire expression appears within double-quotes.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The blank characters appear within double-quotes or square brackets.
.RE
.RS +4
.TP
.ie t \(bu
.el o
Each blank character is preceded by a backslash character.
.RE
.SS "User Subroutines in lex"
.sp
.LP
Anything in the user subroutines section will be copied to \fBlex.yy.c\fR
following \fByylex\fR.
.SS "Regular Expressions     in lex"
.sp
.LP
The \fBlex\fR utility supports the set of Extended Regular Expressions (EREs)
described on \fBregex\fR(5) with the following additions and exceptions to the
syntax:
.sp
.ne 2
.na
\fB\fB\|.\|.\|.\fR \fR
.ad
.RS 14n
Any string enclosed in double-quotes will represent the characters within the
double-quotes as themselves, except that backslash escapes (which appear in the
following table) are recognized. Any backslash-escape sequence is terminated by
the closing quote. For example, "\|\e\|01""1" represents a single string: the
octal value 1 followed by the character 1.
.RE
```

```
.sp
.LP
\fI<\fR\fIstate\fR\fI>\fR\fIr\fR
.sp
.ne 2
.na
\fB<\fIstate1\fR, \fIstate2\fR, \|.\|.\|.\|>\fIr\fR\fR
.ad
.sp .6
.RS 4n
The regular expression \fIr\fR will be matched only when the program is in one
of the start conditions indicated by \fIstate\fR, \fIstate1\fR, and so forth.
For more information, see \fBActions in lex\fR. As an exception to the
typographical conventions of the rest of this document, in this case
<\fIstate\fR> does not represent a metavariable, but the literal angle-bracket
characters surrounding a symbol. The start condition is recognized as such only
at the beginning of a regular expression.
.RE

.sp
.ne 2
.na
\fB\fIr\fR/\fIx\fR \fR
.ad
.sp .6
.RS 4n
The regular expression \fIr\fR will be matched only if it is followed by an
occurrence of regular expression \fIx\fR. The token returned in \fIyytext\fR
will only match \fIr\fR. If the trailing portion of \fIr\fR matches the
beginning of \fIx\fR, the result is unspecified. The \fIr\fR expression cannot
include further trailing context or the \fB$\fR (match-end-of-line) operator;
\fIx\fR cannot include the \fB^\fR (match-beginning-of-line) operator, nor
trailing context, nor the \fB$\fR operator. That is, only one occurrence of
trailing context is allowed in a \fBlex\fR regular expression, and the \fB^\fR
operator only can be used at the beginning of such an expression. A further
restriction is that the trailing-context operator \fB/\fR (slash) cannot be
grouped within parentheses.
.RE

.sp
.ne 2
.na
\fB\fB{\fR\fIname\fR\fB}\fR \fR
.ad
.sp .6
.RS 4n
When \fIname\fR is one of the substitution symbols from the \fIDefinitions\fR
section, the string, including the enclosing braces, will be replaced by the
\fIsubstitute\fR value. The \fIsubstitute\fR value will be treated in the
extended regular expression as if it were enclosed in parentheses. No
substitution will occur if \fB{\fR\fIname\fR\fB}\fR occurs within a bracket
expression or within double-quotes.
.RE

.sp
.LP
Within an \fBERE,\fR a backslash character (\fB\|\e\e\fR, \fB\e\|a\fR,
\fB\e\|b\fR, \fB\e\|f\fR, \fB\e\|n\fR, \fB\e\|r\fR, \fB\e\|t\fR, \fB\e\|v\fR)
is considered to begin an escape sequence. In addition, the escape sequences in
the following table will be recognized.
.sp
.LP
A literal newline character cannot occur within an \fBERE;\fR the escape
sequence \fB\e\|n\fR can be used to represent a newline character. A newline
character cannot be matched by a period operator.
```

```
.sp
.LP
\fBEscape Sequences in lex\fR
.sp

.sp
.TS
box;
c c c
c c c .
Escape Sequences in lex
_
Escape Sequence Description     Meaning
_
\e\fIdigits\fR  T{
A backslash character followed by the longest sequence of one, two or three octa
T}      T{
The character whose encoding is represented by the one-, two- or three-digit oct
T}
_
\e\fBx\fR\fIdigits\fR   T{
A backslash character followed by the longest sequence of hexadecimal-digit char
T}      T{
The character whose encoding is represented by the hexadecimal integer.
T}
_
\e\fIc\fR       T{
A backslash character followed by any character not described in this table.  (\
T}      The character c, unchanged.
.TE

.sp
.LP
The order of precedence given to extended regular expressions for \fBlex\fR is
as shown in the following table, from high to low.
.sp
.ne 2
.na
\fB\fBNote\fR: \fR
.ad
.RS 10n
The escaped characters entry is not meant to imply that these are operators,
but they are included in the table to show their relationships to the true
operators. The start condition, trailing context and anchoring notations have
been omitted from the table because of the placement restrictions described in
this section; they can only appear at the beginning or ending of an \fBERE.\fR
.RE

.sp

.sp
.TS
box;
c c
l l .
ERE Precedence in lex
_
\fIcollation-related bracket symbols\fR \fB[= =]  [: :]  [. .]\fR
\fIescaped characters\fR        \fB\e<\fR\fIspecial character\fR>
\fIbracket expression\fR        \fB[ ]\fR
\fIquoting\fR   \fB".\|.\|."\fR
\fIgrouping\fR  \fB()\fR
\fIdefinition\fR        \fB{\fR\fIname\fR}
\fIsingle-character RE duplication\fR   \fB* + ?\fR
\fIconcatenation\fR
\fIinterval expression\fR       \fB{\fR\fIm\fR,\fIn\fR}
```

```
\fIalternation\fR       \fB|\fR
.TE

.sp
.LP
The \fBERE\fR anchoring operators (\fB\|^\fR and \fB$\fR\|) do not appear in
the table. With \fBlex\fR regular expressions, these operators are restricted
in their use: the \fB^\fR operator can only be used at the beginning of an
entire regular expression, and the \fB$\fR operator only at the end. The
operators apply to the entire regular expression. Thus, for example, the
pattern (\fB^abc)|(def$\fR) is undefined; it can instead be written as two
separate rules, one with the regular expression \fB^abc\fR and one with
\fBdef$\fR, which share a common action via the special \fB|\fR action (see
below). If the pattern were written \fB^abc|def$\fR, it would match either of
\fBabc\fR or \fBdef\fR on a line by itself.
.sp
.LP
Unlike the general \fBERE\fR rules, embedded anchoring is not allowed by most
historical \fBlex\fR implementations. An example of embedded anchoring would be
for patterns such as (^)foo($) to match \fBfoo\fR when it exists as a complete
word. This functionality can be obtained using existing \fBlex\fR features:
.sp
.in +2
.nf
^foo/[ \e\|n]|
" foo"/[ \e\|n]    /* found foo as a separate word */
.fi
.in -2

.sp
.LP
Notice also that \fB$\fR is a form of trailing context (it is equivalent to
\fB/\e\|n\fR and as such cannot be used with regular expressions containing
another instance of the operator (see the preceding discussion of trailing
context).
.sp
.LP
The additional regular expressions trailing-context operator \fB/\fR (slash)
can be used as an ordinary character if presented within double-quotes,
\fB"\|/\|"\fR; preceded by a backslash, \fB\e\|/\fR; or within a bracket
expression, \fB[\|/\|]\fR. The start-condition \fB<\fR and \fB>\fR operators
are special only in a start condition at the beginning of a regular expression;
elsewhere in the regular expression they are treated as ordinary characters.
.sp
.LP
The following examples clarify the differences between \fBlex\fR regular
expressions and regular expressions appearing elsewhere in this document. For
regular expressions of the form \fIr\fR/\fIx\fR, the string matching \fIr\fR is
always returned; confusion may arise when the beginning of \fIx\fR matches the
trailing portion of \fIr\fR. For example, given the regular expression a*b/cc
and the input \fBaaabcc\fR, \fIyytext\fR would contain the string \fBaaab\fR on
this match. But given the regular expression x*/xy and the input \fBxxxy\fR,
the token \fBxxx\fR, not \fBxx\fR, is returned by some implementations because
\fBxxx\fR matches x*.
.sp
.LP
In the rule ab*/bc, the b* at the end of \fIr\fR will extend \fIr\fR’s match
into the beginning of the trailing context, so the result is unspecified. If
this rule were ab/bc, however, the rule matches the text \fBab\fR when it is
followed by the text \fBbc\fR. In this latter case, the matching of \fIr\fR
cannot extend into the beginning of \fIx\fR, so the result is specified.
.SS "Actions in lex"
.sp
.LP
The action to be taken when an \fBERE\fR is matched can be a C program fragment
or the special actions described below; the program fragment can contain one or
```

```
more C statements, and can also include special actions. The empty C statement
\fB;\fR is a valid action; any string in the \fBlex.yy.c\fR input that matches
the pattern portion of such a rule is effectively ignored or skipped. However,
the absence of an action is not valid, and the action \fBlex\fR takes in such a
condition is undefined.
.sp
.LP
The specification for an action, including C statements and special actions,
can extend across several lines if enclosed in braces:
.sp
.in +2
.nf
ERE <one or more blanks> { program statement
program statement }
.fi
.in -2
.sp

.sp
.LP
The default action when a string in the input to a \fBlex.yy.c\fR program is
not matched by any expression is to copy the string to the output. Because the
default behavior of a program generated by \fBlex\fR is to read the input and
copy it to the output, a minimal \fBlex\fR source program that has just
\fB%%\fR generates a C program that simply copies the input to the output
unchanged.
.sp
.LP
Four special actions are available:
.sp
.in +2
.nf
|       ECHO;      REJECT;      BEGIN
.fi
.in -2
.sp

.sp
.ne 2
.na
\fB|\fR
.ad
.RS 12n
The action | means that the action for the next rule is the action for this
rule. Unlike the other three actions, | cannot be enclosed in braces or be
semicolon-terminated. It must be specified alone, with no other actions.
.RE

.sp
.ne 2
.na
\fB\fBECHO;\fR \fR
.ad
.RS 12n
Writes the contents of the string \fIyytext\fR on the output.
.RE

.sp
.ne 2
.na
\fB\fBREJECT;\fR \fR
.ad
.RS 12n
Usually only a single expression is matched by a given string in the input.
\fBREJECT\fR means "continue to the next expression that matches the current
input," and causes whatever rule was the second choice after the current rule
```

```
to be executed for the same input. Thus, multiple rules can be matched and
executed for one input string or overlapping input strings. For example, given
the regular expressions \fBxyz\fR and \fBxy\fR and the input \fBxyz\fR, usually
only the regular expression \fBxyz\fR would match. The next attempted match
would start after z. If the last action in the \fBxyz\fR rule is \fBREJECT\fR ,
both this rule and the \fBxy\fR rule would be executed. The \fBREJECT\fR action
may be implemented in such a fashion that flow of control does not continue
after it, as if it were equivalent to a \fBgoto\fR to another part of
\fByylex\fR. The use of \fBREJECT\fR may result in somewhat larger and slower
scanners.
.RE

.sp
.ne 2
.na
\fB\fBBEGIN\fR \fR
.ad
.RS 12n
The action:
.sp
\fBBEGIN\fR \fInewstate\fR\fB;\fR
.sp
switches the state (start condition) to \fInewstate\fR. If the string
\fInewstate\fR has not been declared previously as a start condition in the
\fBDefinitions\fR \fBin\fR \fBlex\fR section, the results are unspecified. The
initial state is indicated by the digit \fB0\fR or the token \fBINITIAL\fR.
.RE

.sp
.LP
The functions or macros described below are accessible to user code included in
the \fBlex\fR input. It is unspecified whether they appear in the C code output
of \fBlex\fR, or are accessible only through the \fB\fR\fB-l\fR\fB l\fR operand
to \fBc89\fR or \fBcc\fR (the \fBlex\fR library).
.sp
.ne 2
.na
\fB\fBint\fR \fByylex(void)\fR \fR
.ad
.RS 21n
Performs lexical analysis on the input; this is the primary function generated
by the \fBlex\fR utility. The function returns zero when the end of input is
reached; otherwise it returns non-zero values (tokens) determined by the
actions that are selected.
.RE

.sp
.ne 2
.na
\fB\fBint\fR \fByymore(void)\fR \fR
.ad
.RS 21n
When called, indicates that when the next input string is recognized, it is to
be appended to the current value of \fIyytext\fR rather than replacing it; the
value in \fIyyleng\fR is adjusted accordingly.
.RE

.sp
.ne 2
.na
\fB\fBint\fR\fIyyless(int\fR\fB n\fR\fI)\fR \fR
.ad
.RS 21n
Retains \fIn\fR initial characters in \fIyytext\fR, NUL-terminated, and treats
the remaining characters as if they had not been read; the value in
\fIyyleng\fR is adjusted accordingly.
```

```
.RE

.sp
.ne 2
.na
\fB\fBint\fR \fBinput(void)\fR \fR
.ad
.RS 21n
Returns the next character from the input, or zero on end-of-file. It obtains
input from the stream pointer \fIyyin\fR, although possibly via an intermediate
buffer. Thus, once scanning has begun, the effect of altering the value of
\fIyyin\fR is undefined. The character read is removed from the input stream of
the scanner without any processing by the scanner.
.RE

.sp
.ne 2
.na
\fB\fBint\fR \fBunput(int\fR \fB\fIc\fR\fR\fB)\fR \fR
.ad
.RS 21n
Returns the character \fIc\fR to the input; \fIyytext\fR and \fIyyleng\fR are
undefined until the next expression is matched. The result of using \fIunput\fR
for more characters than have been input is unspecified.
.RE

.sp
.LP
The following functions appear only in the \fBlex\fR library accessible through
the \fB\fR\fB-l\fR\fB l\fR operand; they can therefore be redefined by a
portable application:
.sp
.ne 2
.na
\fB\fBint\fR \fByywrap(void)\fR \fR
.ad
.sp .6
.RS 4n
Called by \fByylex\fR at end-of-file; the default \fByywrap\fR always will
return 1. If the application requires \fByylex\fR to continue processing with
another source of input, then the application can include a function
\fByywrap\fR, which associates another file with the external variable
\fBFILE\fR *\fIyyin\fR and will return a value of zero.
.RE

.sp
.ne 2
.na
\fB\fBint\fR \fBmain(int\fR \fB\fIargc\fR,\fR \fBchar\fR \fB*\fIargv\fR[\|])\fR
\fR
.ad
.sp .6
.RS 4n
Calls \fByylex\fR to perform lexical analysis, then exits. The user code can
contain \fBmain\fR to perform application-specific operations, calling
\fByylex\fR as applicable.
.RE

.sp
.LP
The reason for breaking these functions into two lists is that only those
functions in \fBlibl.a\fR can be reliably redefined by a portable application.
.sp
.LP
Except for \fBinput\fR, \fBunput\fR and \fBmain\fR, all external and static
names generated by \fBlex\fR begin with the prefix \fByy\fR or \fBYY\fR.
```

```
.SH USAGE
.sp
.LP
Portable applications are warned that in the \fBRules in lex\fR section, an
\fBERE\fR without an action is not acceptable, but need not be detected as
erroneous by \fBlex\fR. This may result in compilation or run-time errors.
.sp
.LP
The purpose of \fBinput\fR is to take characters off the input stream and
discard them as far as the lexical analysis is concerned. A common use is to
discard the body of a comment once the beginning of a comment is recognized.
.sp
.LP
The \fBlex\fR utility is not fully internationalized in its treatment of
regular expressions in the \fBlex\fR source code or generated lexical analyzer.
It would seem desirable to have the lexical analyzer interpret the regular
expressions given in the \fBlex\fR source according to the environment
specified when the lexical analyzer is executed, but this is not possible with
the current \fBlex\fR technology. Furthermore, the very nature of the lexical
analyzers produced by \fBlex\fR must be closely tied to the lexical
requirements of the input language being described, which will frequently be
locale-specific anyway. (For example, writing an analyzer that is used for
French text will not automatically be useful for processing other languages.)
.SH EXAMPLES
.LP
\fBExample 1 \fRUsing lex
.sp
.LP
The following is an example of a \fBlex\fR program that implements a
rudimentary scanner for a Pascal-like syntax:

.sp
.in +2
.nf
%{
/* need this for the call to atof() below */
#include <math.h>
/* need this for printf(), fopen() and stdin below */
#include <stdio.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*
%%

{DIGIT}+        {
                        printf("An integer: %s (%d)\en", yytext,
                        atoi(yytext));
                        }

{DIGIT}+"."{DIGIT}*     {
                        printf("A float: %s (%g)\en", yytext,
                        atof(yytext));
                        }

if|then|begin|end|procedure|function        {
                        printf("A keyword: %s\en", yytext);
                        }

{ID}                    printf("An identifier: %s\en", yytext);

"+"|"-"|"*"|"/"         printf("An operator: %s\en", yytext);

"{"[^}\en]*"}"          /* eat up one-line comments */

[ \et\en]+                /* eat up white space */
```

```
\&.                       printf("Unrecognized character: %s\en", yytext);

%%

int main(int argc, char *argv[\|])
{
                      ++argv, --argc;  /* skip over program name */
                      if (argc > 0)
                                  yyin = fopen(argv[0], "r");
                      else
                      yyin = stdin;

                      yylex();
}
```

______***unchanged_portion_omitted_***