CSC340
Programming Languages & Compilation
Homework #2

LEX/YACC Stack Calculator Implementation

Ammar Alamri
Std#: 438104833
Section: 57120
Serial number: 7

Q3) LEX code:

```
%{
        void yyerror (const char *s);
        #include "calc.tab.h"
        int yylex();
%}

%%
"print"             {return print;}
"exit"              {return exit_command;}
[0-9]+              {yylval.num = atoi(yytext); return number;}
[ \t\n\r\v\f]+      {;}
;                   {;}
push                {return push;}
pop                 {return pop;}
add                 {return add;}
mul                 {return mul;}
sub                 {return sub;}
divide              {return divide;}
.                   {ECHO; yyerror (" unexpected character");}
%%

int yywrap (void) {return 1;}
```

Q4 + Q5) LEX Examples:
I recorded the lexical analyzer output using the -d option when compiling the lex file (flex -d calc.l). The input is in a text file, I ran the file using Windows cmd (calc < inputs.txt).

First input file contains commands: push 5 > push 6 > print > add > print > push 3 > divide > print > mul > print > push 5 > mul > print > push 3 > sub > print

Output:
--(end of buffer or a NUL)
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("5")

--accepting rule at line 11 ("
")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("6")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 6
--accepting rule at line 11 ("
")
--accepting rule at line 15 ("add")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 11
--accepting rule at line 11 ("
")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("3")
--accepting rule at line 11 ("
")
--accepting rule at line 18 ("divide")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 3
--accepting rule at line 11 ("
")
--accepting rule at line 16 ("mul")
Stack does not contain enough elements
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 3
--accepting rule at line 11 ("
")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("5")
--accepting rule at line 11 ("
")
--accepting rule at line 16 ("mul")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 15
--accepting rule at line 11 ("

")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("3")
--accepting rule at line 11 ("
")
--accepting rule at line 17 ("sub")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: 12
--(end of buffer or a NUL)
--EOF (start condition 0)

Second input file contains: pop > push 3 > sub > push 4 > sup > sub > print

--(end of buffer or a NUL)
--accepting rule at line 14 ("pop")
Stack is empty
--accepting rule at line 11 ("
")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("3")
--accepting rule at line 11 ("
")
--accepting rule at line 17 ("sub")
Stack does not contain enough elements
--accepting rule at line 11 ("
")
--accepting rule at line 13 ("push")
--accepting rule at line 11 (" ")
--accepting rule at line 10 ("4")
--accepting rule at line 11 ("
")
--accepting rule at line 19 ("s")
s unexpected character
--accepting rule at line 19 ("u")
u unexpected character
--accepting rule at line 19 ("p")
p unexpected character
--accepting rule at line 11 ("
")
--accepting rule at line 17 ("sub")
--accepting rule at line 11 ("
")
--accepting rule at line 8 ("print")
Top of stack is: -1

--(end of buffer or a NUL)
--EOF (start condition 0)

Q6) Bison code:

```
%{
void yyerror(const char *s);
#include <stdio.h>
#include <stdlib.h>
int yylex();


int stack[60];
int size = -1;
int fpop();
void fpush(int val);
void fadd();
void fmul();
void fsub();
void fdiv();
void fprint();
%}

%union {int num; char *string;}
%start line
%token print
%token exit_command
%token <num> number
%token pop push add mul sub divide

/* %type <num> line exp */
```

```
%%


line    : exp                   {;}
        | exit_command                  {exit(EXIT_SUCCESS);}
        | line exp              {;}
        | line exit_command     {exit(EXIT_SUCCESS);}
        ;

exp     : print         {fprint();}
        | push number           {fpush($2);}
        | pop                   {fpop();}
        | add                   {fadd();}
        | mul                   {fmul();}
        | sub                   {fsub();}
        | divide            {fdiv();}
        ;

%%

void fprint() {
        printf("Top of stack is: %d\n", stack[size]);
}

void fpush(int val)
{
        if(size < 60) {
                stack[++size] = val;
                /* printf("Added"); */
                return;
```

```
	}
	/* printf("Stack is full\n"); */
}

int fpop()
{
	if(size > -1) {
		return stack[size--];
	}
	printf("Stack is empty\n");
	return -1;
}

void fadd()
{	if(size > 0) {
		fpush(fpop() + fpop());
	} else {
	printf("Stack does not contain enough elements\n");
	}
}

void fmul()
{
	if(size > 0) {
		fpush(fpop() * fpop());
	} else {
	printf("Stack does not contain enough elements\n");
	}
}
```

```
void fsub()
{
	if(size > 0) {
		int ope2 = fpop();
		fpush(fpop() - ope2);
	} else {
	printf("Stack does not contain enough elements\n");
	}
}

void fdiv()
{
	if(size > 0) {
		int ope2 = fpop();
		fpush(fpop() / ope2);
	} else {
	printf("Stack does not contain enough elements\n");
	}

}



int main (void) {
	printf("Commands are: push, pop, add, mul, sub, divide and print.");
	int i;
	for(i=0; i<60; i++) {
		stack[i] = 0;
	}
	return yyparse();
}
```

```
void yyerror (const char *s) {fprintf(stderr, "%s\n", s);}
```

Q7) Addition action:

```
Action: {fadd();}
void fadd()
{       if(size > 0) {
                fpush(fpop() + fpop());
        } else {
        printf("Stack does not contain enough elements\n");
        }
}
```

Q8 + Q9) Parsing examples:

Example 1:

```
        Commands are: push, pop, add, mul, sub, divide and print.
        push 5
        push 6
        print
        Top of stack is: 6
        add
        print
        Top of stack is: 11
        push 3
        divide
        print
        Top of stack is: 3
        mul
        Stack does not contain enough elements
        print
```

Top of stack is: 3

push 5

mul

print

Top of stack is: 15

push 3

sub

print

Top of stack is: 12

Example 2:

Commands are: push, pop, add, mul, sub, divide and print.

pop

Stack is empty

push 3

sub

Stack does not contain enough elements

push 4

sup

s unexpected character

u unexpected character

p unexpected character

sub

print

Top of stack is: -1

Q10) Produced bison code:

/* A Bison parser, made by GNU Bison 2.4.1. */

/* Skeleton implementation for Bison's Yacc-like parsers in C

Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004, 2005, 2006
Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>. */

/* As a special exception, you may create a larger work that contains
part or all of the Bison parser skeleton and distribute that work
under terms of your choice, so long as that work isn't itself a
parser generator using the skeleton or a modified version thereof
as a parser skeleton. Alternatively, if you modify or redistribute
the parser skeleton itself, you may (at your option) remove this
special exception, which will cause the skeleton and the resulting
Bison output files to be licensed under the GNU General Public

```
   License without this special exception.

   This special exception was added by the Free Software Foundation in
   version 2.2 of Bison.  */

/* C LALR(1) parser skeleton written by Richard Stallman, by
   simplifying the original so-called "semantic" parser.  */

/* All symbols defined below should begin with yy or YY, to avoid
   infringing on user name space.  This should be done even for local
   variables, as they might otherwise be expanded by user macros.
   There are some unavoidable exceptions within include files to
   define necessary library symbols; they are noted "INFRINGES ON
   USER NAME SPACE" below.  */

/* Identify Bison output.  */
#define YYBISON 1

/* Bison version.  */
#define YYBISON_VERSION "2.4.1"

/* Skeleton name.  */
#define YYSKELETON_NAME "yacc.c"

/* Pure parsers.  */
#define YYPURE 0

/* Push parsers.  */
#define YYPUSH 0
```

```
/* Pull parsers.  */
#define YYPULL 1

/* Using locations.  */
#define YYLSP_NEEDED 0



/* Copy the first part of user declarations.  */

/* Line 189 of yacc.c  */
#line 1 "calc.y"

void yyerror(const char *s);
#include <stdio.h>
#include <stdlib.h>
int yylex();


int stack[60];
int size = -1;
int fpop();
void fpush(int val);
void fadd();
void fmul();
void fsub();
void fdiv();
void fprint();
```

```
/* Line 189 of yacc.c  */
#line 92 "calc.tab.c"

/* Enabling traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif

/* Enabling verbose error messages.  */
#ifdef YYERROR_VERBOSE
# undef YYERROR_VERBOSE
# define YYERROR_VERBOSE 1
#else
# define YYERROR_VERBOSE 0
#endif

/* Enabling the token table.  */
#ifndef YYTOKEN_TABLE
# define YYTOKEN_TABLE 0
#endif


/* Tokens.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
   /* Put the tokens into the symbol table, so that GDB and other debuggers
      know about them.  */
   enum yytokentype {
     print = 258,
     exit_command = 259,
```

```
    number = 260,
    pop = 261,
    push = 262,
    add = 263,
    mul = 264,
    sub = 265,
    divide = 266
  };
#endif



#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef union YYSTYPE
{

/* Line 214 of yacc.c  */
#line 19 "calc.y"
int num; char *string;



/* Line 214 of yacc.c  */
#line 143 "calc.tab.c"
} YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif
```

```
/* Copy the second part of user declarations.  */


/* Line 264 of yacc.c  */
#line 155 "calc.tab.c"

#ifdef short
# undef short
#endif

#ifdef YYTYPE_UINT8
typedef YYTYPE_UINT8 yytype_uint8;
#else
typedef unsigned char yytype_uint8;
#endif

#ifdef YYTYPE_INT8
typedef YYTYPE_INT8 yytype_int8;
#elif (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
typedef signed char yytype_int8;
#else
typedef short int yytype_int8;
#endif

#ifdef YYTYPE_UINT16
typedef YYTYPE_UINT16 yytype_uint16;
#else
typedef unsigned short int yytype_uint16;
#endif
```

```
#ifdef YYTYPE_INT16
typedef YYTYPE_INT16 yytype_int16;
#else
typedef short int yytype_int16;
#endif

#ifndef YYSIZE_T
# ifdef __SIZE_TYPE__
#  define YYSIZE_T __SIZE_TYPE__
# elif defined size_t
#  define YYSIZE_T size_t
# elif ! defined YYSIZE_T && (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
#  include <stddef.h> /* INFRINGES ON USER NAME SPACE */
#  define YYSIZE_T size_t
# else
#  define YYSIZE_T unsigned int
# endif
#endif

#define YYSIZE_MAXIMUM ((YYSIZE_T) -1)

#ifndef YY_
# if YYENABLE_NLS
#  if ENABLE_NLS
#   include <libintl.h> /* INFRINGES ON USER NAME SPACE */
#   define YY_(msgid) dgettext ("bison-runtime", msgid)
#  endif
# endif
```

```
# ifndef YY_
#  define YY_(msgid) msgid
# endif
#endif

/* Suppress unused-variable warnings by "using" E.  */
#if ! defined lint || defined __GNUC__
# define YYUSE(e) ((void) (e))
#else
# define YYUSE(e) /* empty */
#endif

/* Identity function, used to suppress warnings about constant conditions.  */
#ifndef lint
# define YYID(n) (n)
#else
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static int
YYID (int yyi)
#else
static int
YYID (yyi)
    int yyi;
#endif
{
  return yyi;
}
#endif
```

```
#if ! defined yyoverflow || YYERROR_VERBOSE

/* The parser invokes alloca or malloc; define the necessary symbols.  */

# ifdef YYSTACK_USE_ALLOCA
#  if YYSTACK_USE_ALLOCA
#   ifdef __GNUC__
#    define YYSTACK_ALLOC __builtin_alloca
#   elif defined __BUILTIN_VA_ARG_INCR
#    include <alloca.h> /* INFRINGES ON USER NAME SPACE */
#   elif defined _AIX
#    define YYSTACK_ALLOC __alloca
#   elif defined _MSC_VER
#    include <malloc.h> /* INFRINGES ON USER NAME SPACE */
#    define alloca _alloca
#   else
#    define YYSTACK_ALLOC alloca
#    if ! defined _ALLOCA_H && ! defined _STDLIB_H && (defined __STDC__ || defined
__C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
#     include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
#     ifndef _STDLIB_H
#      define _STDLIB_H 1
#     endif
#    endif
#   endif
#  endif
# endif

# ifdef YYSTACK_ALLOC
```

```
   /* Pacify GCC's `empty if-body' warning.  */
#  define YYSTACK_FREE(Ptr) do { /* empty */; } while (YYID (0))
#  ifndef YYSTACK_ALLOC_MAXIMUM
    /* The OS might guarantee only one guard page at the bottom of the stack,
       and a page size can be as small as 4096 bytes.  So we cannot safely
       invoke alloca (N) if N exceeds 4096.  Use a slightly smaller number
       to allow for a few compiler-allocated temporary stack slots.  */
#   define YYSTACK_ALLOC_MAXIMUM 4032 /* reasonable circa 2006 */
#  endif
# else
#  define YYSTACK_ALLOC YYMALLOC
#  define YYSTACK_FREE YYFREE
#  ifndef YYSTACK_ALLOC_MAXIMUM
#   define YYSTACK_ALLOC_MAXIMUM YYSIZE_MAXIMUM
#  endif
#  if (defined __cplusplus && ! defined _STDLIB_H \
       && ! ((defined YYMALLOC || defined malloc) \
             && (defined YYFREE || defined free)))
#   include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
#   ifndef _STDLIB_H
#    define _STDLIB_H 1
#   endif
#  endif
#  ifndef YYMALLOC
#   define YYMALLOC malloc
#   if ! defined malloc && ! defined _STDLIB_H && (defined __STDC__ || defined __C99__FUNC__
\
     || defined __cplusplus || defined _MSC_VER)
void *malloc (YYSIZE_T); /* INFRINGES ON USER NAME SPACE */
#   endif
```

```
#  endif
#  ifndef YYFREE
#   define YYFREE free
#   if ! defined free && ! defined _STDLIB_H && (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
void free (void *); /* INFRINGES ON USER NAME SPACE */
#   endif
#  endif
# endif
#endif /* ! defined yyoverflow || YYERROR_VERBOSE */


#if (! defined yyoverflow \
     && (! defined __cplusplus \
           || (defined YYSTYPE_IS_TRIVIAL && YYSTYPE_IS_TRIVIAL)))

/* A type that is properly aligned for any stack member.  */
union yyalloc
{
  yytype_int16 yyss_alloc;
  YYSTYPE yyvs_alloc;
};

/* The size of the maximum gap between one aligned stack and the next.  */
# define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)

/* The size of an array large to enough to hold all stacks, each with
   N elements.  */
# define YYSTACK_BYTES(N) \
     ((N) * (sizeof (yytype_int16) + sizeof (YYSTYPE)) \
```

```
      + YYSTACK_GAP_MAXIMUM)

/* Copy COUNT objects from FROM to TO.  The source and destination do
   not overlap.  */
# ifndef YYCOPY
#  if defined __GNUC__ && 1 < __GNUC__
#   define YYCOPY(To, From, Count) \
      __builtin_memcpy (To, From, (Count) * sizeof (*(From)))
#  else
#   define YYCOPY(To, From, Count)                  \
      do                                            \
        {                                           \
          YYSIZE_T yyi;                             \
          for (yyi = 0; yyi < (Count); yyi++)       \
            (To)[yyi] = (From)[yyi];                \
        }                                           \
      while (YYID (0))
#  endif
# endif

/* Relocate STACK from its old location to the new one.  The
   local variables YYSIZE and YYSTACKSIZE give the old and new number of
   elements in the stack, and YYPTR gives the new location of the
   stack.  Advance YYPTR to a properly aligned location for the next
   stack.  */
# define YYSTACK_RELOCATE(Stack_alloc, Stack)                                \
    do                                                                       \
      {                                                                      \
        YYSIZE_T yynewbytes;                                                 \
        YYCOPY (&yyptr->Stack_alloc, Stack, yysize);                         \
```

```
        Stack = &yyptr->Stack_alloc;                                    \
        yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \
        yyptr += yynewbytes / sizeof (*yyptr);                          \
      }                                                                 \
    while (YYID (0))

#endif

/* YYFINAL -- State number of the termination state.  */
#define YYFINAL  12
/* YYLAST -- Last index in YYTABLE.  */
#define YYLAST   20

/* YYNTOKENS -- Number of terminals.  */
#define YYNTOKENS  12
/* YYNNTS -- Number of nonterminals.  */
#define YYNNTS  3
/* YYNRULES -- Number of rules.  */
#define YYNRULES  12
/* YYNRULES -- Number of states.  */
#define YYNSTATES  15

/* YYTRANSLATE(YYLEX) -- Bison symbol number corresponding to YYLEX.  */
#define YYUNDEFTOK  2
#define YYMAXUTOK   266

#define YYTRANSLATE(YYX)                                                \
  ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)

/* YYTRANSLATE[YYLEX] -- Bison symbol number corresponding to YYLEX.  */
```

```
static const yytype_uint8 yytranslate[] =
{
       0,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     1,     2,     3,     4,
       5,     6,     7,     8,     9,    10,    11
};
```

```
#if YYDEBUG
/* YYPRHS[YYN] -- Index of the first RHS symbol of rule number YYN in
   YYRHS.  */
static const yytype_uint8 yyprhs[] =
{
       0,     0,     3,     5,     7,    10,    13,    15,    18,    20,
      22,    24,    26
};

/* YYRHS -- A `-1'-separated list of the rules' RHS.  */
static const yytype_int8 yyrhs[] =
{
      13,     0,    -1,    14,    -1,     4,    -1,    13,    14,    -1,
      13,     4,    -1,     3,    -1,     7,     5,    -1,     6,    -1,
       8,    -1,     9,    -1,    10,    -1,    11,    -1
};

/* YYRLINE[YYN] -- source line where rule number YYN was defined.  */
static const yytype_uint8 yyrline[] =
{
       0,    31,    31,    32,    33,    34,    37,    38,    39,    40,
      41,    42,    43
};
#endif

#if YYDEBUG || YYERROR_VERBOSE || YYTOKEN_TABLE
/* YYTNAME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.
   First, the terminals, then, starting at YYNTOKENS, nonterminals.  */
static const char *const yytname[] =
```

```
{
  "$end", "error", "$undefined", "print", "exit_command", "number", "pop",
  "push", "add", "mul", "sub", "divide", "$accept", "line", "exp", 0
};
#endif

# ifdef YYPRINT
/* YYTOKNUM[YYLEX-NUM] -- Internal token number corresponding to
   token YYLEX-NUM.  */
static const yytype_uint16 yytoknum[] =
{
       0,   256,   257,   258,   259,   260,   261,   262,   263,   264,
     265,   266
};
# endif

/* YYR1[YYN] -- Symbol number of symbol that rule YYN derives.  */
static const yytype_uint8 yyr1[] =
{
       0,    12,    13,    13,    13,    13,    14,    14,    14,    14,
      14,    14,    14
};

/* YYR2[YYN] -- Number of symbols composing right hand side of rule YYN.  */
static const yytype_uint8 yyr2[] =
{
       0,     2,     1,     1,     2,     2,     1,     2,     1,     1,
       1,     1,     1
};
```

```
/* YYDEFACT[STATE-NAME] -- Default rule to reduce with in state
   STATE-NUM when YYTABLE doesn't specify something else to do.  Zero
   means the default is an error.  */
static const yytype_uint8 yydefact[] =
{
       0,     6,     3,     8,     0,     9,    10,    11,    12,     0,
       2,     7,     1,     5,     4
};

/* YYDEFGOTO[NTERM-NUM].  */
static const yytype_int8 yydefgoto[] =
{
      -1,     9,    10
};

/* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing
   STATE-NUM.  */
#define YYPACT_NINF -8
static const yytype_int8 yypact[] =
{
       9,    -8,    -8,    -8,    -4,    -8,    -8,    -8,    -8,     0,
      -8,    -8,    -8,    -8,    -8
};

/* YYPGOTO[NTERM-NUM].  */
static const yytype_int8 yypgoto[] =
{
      -8,    -8,    -7
};
```

```
/* YYTABLE[YYPACT[STATE-NUM]].  What to do in state STATE-NUM.  If
   positive, shift that token.  If negative, reduce the rule which
   number is the opposite.  If zero, do what YYDEFACT says.
   If YYTABLE_NINF, syntax error.  */
#define YYTABLE_NINF -1
static const yytype_uint8 yytable[] =
{
      12,    11,    14,     1,    13,     0,     3,     4,     5,     6,
       7,     8,     1,     2,     0,     3,     4,     5,     6,     7,
       8
};

static const yytype_int8 yycheck[] =
{
       0,     5,     9,     3,     4,    -1,     6,     7,     8,     9,
      10,    11,     3,     4,    -1,     6,     7,     8,     9,    10,
      11
};

/* YYSTOS[STATE-NUM] -- The (internal number of the) accessing
   symbol of state STATE-NUM.  */
static const yytype_uint8 yystos[] =
{
       0,     3,     4,     6,     7,     8,     9,    10,    11,    13,
      14,     5,     0,     4,    14
};

#define yyerrok                  (yyerrstatus = 0)
#define yyclearin        (yychar = YYEMPTY)
#define YYEMPTY                  (-2)
```

```
#define YYEOF                   0

#define YYACCEPT  goto yyacceptlab
#define YYABORT             goto yyabortlab
#define YYERROR             goto yyerrorlab


/* Like YYERROR except do call yyerror.  This remains here temporarily
   to ease the transition to the new meaning of YYERROR, for GCC.
   Once GCC version 2 has supplanted version 1, this can go.  */

#define YYFAIL              goto yyerrlab

#define YYRECOVERING()  (!!yyerrstatus)

#define YYBACKUP(Token, Value)                                  \
do                                                              \
  if (yychar == YYEMPTY && yylen == 1)                          \
    {                                                           \
      yychar = (Token);                                         \
      yylval = (Value);                                         \
      yytoken = YYTRANSLATE (yychar);                           \
      YYPOPSTACK (1);                                           \
      goto yybackup;                                            \
    }                                                           \
  else                                                          \
    {                                                           \
      yyerror (YY_("syntax error: cannot back up")); \
      YYERROR;                                                  \
    }                                                           \
```

```
while (YYID (0))


#define YYTERROR 1
#define YYERRCODE         256


/* YYLLOC_DEFAULT -- Set CURRENT to span from RHS[1] to RHS[N].
   If N is 0, then set CURRENT to the empty location which ends
   the previous symbol: RHS[0] (always defined).  */

#define YYRHSLOC(Rhs, K) ((Rhs)[K])
#ifndef YYLLOC_DEFAULT
# define YYLLOC_DEFAULT(Current, Rhs, N)                                \
    do                                                                  \
      if (YYID (N))                                                     \
        {                                                               \
          (Current).first_line   = YYRHSLOC (Rhs, 1).first_line;    \
          (Current).first_column = YYRHSLOC (Rhs, 1).first_column;      \
          (Current).last_line    = YYRHSLOC (Rhs, N).last_line;         \
          (Current).last_column  = YYRHSLOC (Rhs, N).last_column;      \
        }                                                               \
      else                                                              \
        {                                                               \
          (Current).first_line   = (Current).last_line   =              \
            YYRHSLOC (Rhs, 0).last_line;                                \
          (Current).first_column = (Current).last_column =              \
            YYRHSLOC (Rhs, 0).last_column;                              \
        }                                                               \
    while (YYID (0))
```

```
#endif


/* YY_LOCATION_PRINT -- Print the location on the stream.
   This macro was not mandated originally: define only if we know
   we won't break user code: when these are the locations we know.  */

#ifndef YY_LOCATION_PRINT
# if YYLTYPE_IS_TRIVIAL
#  define YY_LOCATION_PRINT(File, Loc)                          \
     fprintf (File, "%d.%d-%d.%d",                              \
              (Loc).first_line, (Loc).first_column,             \
              (Loc).last_line,  (Loc).last_column)
# else
#  define YY_LOCATION_PRINT(File, Loc) ((void) 0)
# endif
#endif


/* YYLEX -- calling `yylex' with the right arguments.  */

#ifdef YYLEX_PARAM
# define YYLEX yylex (YYLEX_PARAM)
#else
# define YYLEX yylex ()
#endif

/* Enable debugging if requested.  */
#if YYDEBUG
```

```
# ifndef YYFPRINTF
#  include <stdio.h> /* INFRINGES ON USER NAME SPACE */
#  define YYFPRINTF fprintf
# endif

# define YYDPRINTF(Args)                        \
do {                                            \
  if (yydebug)                                  \
    YYFPRINTF Args;                             \
} while (YYID (0))

# define YY_SYMBOL_PRINT(Title, Type, Value, Location)                          \
do {                                                                            \
  if (yydebug)                                                                  \
    {                                                                           \
      YYFPRINTF (stderr, "%s ", Title);                                         \
      yy_symbol_print (stderr,                                                  \
                       Type, Value); \
      YYFPRINTF (stderr, "\n");                                                 \
    }                                                                           \
} while (YYID (0))


/*--------------------------------.
| Print this symbol on YYOUTPUT.  |
`--------------------------------*/

/*ARGSUSED*/
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
```

```
static void
yy_symbol_value_print (FILE *yyoutput, int yytype, YYSTYPE const * const yyvaluep)
#else
static void
yy_symbol_value_print (yyoutput, yytype, yyvaluep)
    FILE *yyoutput;
    int yytype;
    YYSTYPE const * const yyvaluep;
#endif
{
  if (!yyvaluep)
    return;
# ifdef YYPRINT
  if (yytype < YYNTOKENS)
    YYPRINT (yyoutput, yytoknum[yytype], *yyvaluep);
# else
  YYUSE (yyoutput);
# endif
  switch (yytype)
    {
      default:
            break;
    }
}


/*---------------------------------.
| Print this symbol on YYOUTPUT.  |
`---------------------------------*/
```

```
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static void
yy_symbol_print (FILE *yyoutput, int yytype, YYSTYPE const * const yyvaluep)
#else
static void
yy_symbol_print (yyoutput, yytype, yyvaluep)
    FILE *yyoutput;
    int yytype;
    YYSTYPE const * const yyvaluep;
#endif
{
  if (yytype < YYNTOKENS)
    YYFPRINTF (yyoutput, "token %s (", yytname[yytype]);
  else
    YYFPRINTF (yyoutput, "nterm %s (", yytname[yytype]);


  yy_symbol_value_print (yyoutput, yytype, yyvaluep);
  YYFPRINTF (yyoutput, ")");
}


/*------------------------------------------------------------------.
| yy_stack_print -- Print the state stack from its BOTTOM up to its |
| TOP (included).                                                   |
`------------------------------------------------------------------*/


#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static void
yy_stack_print (yytype_int16 *yybottom, yytype_int16 *yytop)
```

```
#else
static void
yy_stack_print (yybottom, yytop)
    yytype_int16 *yybottom;
    yytype_int16 *yytop;
#endif
{
  YYFPRINTF (stderr, "Stack now");
  for (; yybottom <= yytop; yybottom++)
    {
      int yybot = *yybottom;
      YYFPRINTF (stderr, " %d", yybot);
    }
  YYFPRINTF (stderr, "\n");
}

# define YY_STACK_PRINT(Bottom, Top)                            \
do {                                                            \
  if (yydebug)                                                  \
    yy_stack_print ((Bottom), (Top));                           \
} while (YYID (0))


/*------------------------------------------------.
| Report that the YYRULE is going to be reduced.  |
`------------------------------------------------*/

#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static void
```

```
yy_reduce_print (YYSTYPE *yyvsp, int yyrule)
#else
static void
yy_reduce_print (yyvsp, yyrule)
    YYSTYPE *yyvsp;
    int yyrule;
#endif
{
  int yynrhs = yyr2[yyrule];
  int yyi;
  unsigned long int yylno = yyrline[yyrule];
  YYFPRINTF (stderr, "Reducing stack by rule %d (line %lu):\n",
             yyrule - 1, yylno);
  /* The symbols being reduced.  */
  for (yyi = 0; yyi < yynrhs; yyi++)
    {
      YYFPRINTF (stderr, "   $%d = ", yyi + 1);
      yy_symbol_print (stderr, yyrhs[yyprhs[yyrule] + yyi],
                       &(yyvsp[(yyi + 1) - (yynrhs)])
                                              );
      YYFPRINTF (stderr, "\n");
    }
}

# define YY_REDUCE_PRINT(Rule)                  \
do {                                            \
  if (yydebug)                                  \
    yy_reduce_print (yyvsp, Rule); \
} while (YYID (0))
```

```
/* Nonzero means print parse trace.  It is left uninitialized so that
   multiple parsers can coexist.  */
int yydebug;
#else /* !YYDEBUG */
# define YYDPRINTF(Args)
# define YY_SYMBOL_PRINT(Title, Type, Value, Location)
# define YY_STACK_PRINT(Bottom, Top)
# define YY_REDUCE_PRINT(Rule)
#endif /* !YYDEBUG */



/* YYINITDEPTH -- initial size of the parser's stacks.  */
#ifndef YYINITDEPTH
# define YYINITDEPTH 200
#endif

/* YYMAXDEPTH -- maximum size the stacks can grow to (effective only
   if the built-in stack extension method is used).

   Do not make this value too large; the results are undefined if
   YYSTACK_ALLOC_MAXIMUM < YYSTACK_BYTES (YYMAXDEPTH)
   evaluated with infinite-precision integer arithmetic.  */

#ifndef YYMAXDEPTH
# define YYMAXDEPTH 10000
#endif
```

```
#if YYERROR_VERBOSE

# ifndef yystrlen
#  if defined __GLIBC__ && defined _STRING_H
#   define yystrlen strlen
#  else
/* Return the length of YYSTR.  */
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static YYSIZE_T
yystrlen (const char *yystr)
#else
static YYSIZE_T
yystrlen (yystr)
    const char *yystr;
#endif
{
  YYSIZE_T yylen;
  for (yylen = 0; yystr[yylen]; yylen++)
    continue;
  return yylen;
}
#  endif
# endif

# ifndef yystpcpy
#  if defined __GLIBC__ && defined _STRING_H && defined _GNU_SOURCE
#   define yystpcpy stpcpy
```

```
#  else
/* Copy YYSRC to YYDEST, returning the address of the terminating '\0' in
   YYDEST.  */
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static char *
yystpcpy (char *yydest, const char *yysrc)
#else
static char *
yystpcpy (yydest, yysrc)
    char *yydest;
    const char *yysrc;
#endif
{
  char *yyd = yydest;
  const char *yys = yysrc;

  while ((*yyd++ = *yys++) != '\0')
    continue;

  return yyd - 1;
}
#  endif
# endif

# ifndef yytnamerr
/* Copy to YYRES the contents of YYSTR after stripping away unnecessary
   quotes and backslashes, so that it's suitable for yyerror.  The
   heuristic is that double-quoting is unnecessary unless the string
   contains an apostrophe, a comma, or backslash (other than
```

```
   backslash-backslash).  YYSTR is taken from yytname.  If YYRES is
   null, do not copy; instead, return the length of what the result
   would have been.  */
static YYSIZE_T
yytnamerr (char *yyres, const char *yystr)
{
  if (*yystr == "")
    {
      YYSIZE_T yyn = 0;
      char const *yyp = yystr;

      for (;;)
        switch (*++yyp)
          {
          case '\'':
          case ',':
            goto do_not_strip_quotes;

          case '\\':
            if (*++yyp != '\\')
              goto do_not_strip_quotes;
            /* Fall through.  */
          default:
            if (yyres)
              yyres[yyn] = *yyp;
            yyn++;
            break;

          case "":
            if (yyres)
```

```
                yyres[yyn] = '\0';
              return yyn;
            }
    do_not_strip_quotes: ;
    }

  if (! yyres)
    return yystrlen (yystr);

  return yystpcpy (yyres, yystr) - yyres;
}
# endif

/* Copy into YYRESULT an error message about the unexpected token
   YYCHAR while in state YYSTATE.  Return the number of bytes copied,
   including the terminating null byte.  If YYRESULT is null, do not
   copy anything; just return the number of bytes that would be
   copied.  As a special case, return 0 if an ordinary "syntax error"
   message will do.  Return YYSIZE_MAXIMUM if overflow occurs during
   size calculation.  */
static YYSIZE_T
yysyntax_error (char *yyresult, int yystate, int yychar)
{
  int yyn = yypact[yystate];

  if (! (YYPACT_NINF < yyn && yyn <= YYLAST))
    return 0;
  else
    {
      int yytype = YYTRANSLATE (yychar);
```

```
      YYSIZE_T yysize0 = yytnamerr (0, yytname[yytype]);
      YYSIZE_T yysize = yysize0;
      YYSIZE_T yysize1;
      int yysize_overflow = 0;
      enum { YYERROR_VERBOSE_ARGS_MAXIMUM = 5 };
      char const *yyarg[YYERROR_VERBOSE_ARGS_MAXIMUM];
      int yyx;

# if 0
      /* This is so xgettext sees the translatable formats that are
            constructed on the fly.  */
      YY_("syntax error, unexpected %s");
      YY_("syntax error, unexpected %s, expecting %s");
      YY_("syntax error, unexpected %s, expecting %s or %s");
      YY_("syntax error, unexpected %s, expecting %s or %s or %s");
      YY_("syntax error, unexpected %s, expecting %s or %s or %s or %s");
# endif
      char *yyfmt;
      char const *yyf;
      static char const yyunexpected[] = "syntax error, unexpected %s";
      static char const yyexpecting[] = ", expecting %s";
      static char const yyor[] = " or %s";
      char yyformat[sizeof yyunexpected
                    + sizeof yyexpecting - 1
                    + ((YYERROR_VERBOSE_ARGS_MAXIMUM - 2)
                       * (sizeof yyor - 1))];
      char const *yyprefix = yyexpecting;

      /* Start YYX at -YYN if negative to avoid negative indexes in
            YYCHECK.  */
```

```
int yyxbegin = yyn < 0 ? -yyn : 0;

/* Stay within bounds of both yycheck and yytname.  */
int yychecklim = YYLAST - yyn + 1;
int yyxend = yychecklim < YYNTOKENS ? yychecklim : YYNTOKENS;
int yycount = 1;

yyarg[0] = yytname[yytype];
yyfmt = yystpcpy (yyformat, yyunexpected);

for (yyx = yyxbegin; yyx < yyxend; ++yyx)
     if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR)
       {
         if (yycount == YYERROR_VERBOSE_ARGS_MAXIMUM)
           {
               yycount = 1;
               yysize = yysize0;
               yyformat[sizeof yyunexpected - 1] = '\0';
               break;
           }
         yyarg[yycount++] = yytname[yyx];
         yysize1 = yysize + yytnamerr (0, yytname[yyx]);
         yysize_overflow |= (yysize1 < yysize);
         yysize = yysize1;
         yyfmt = yystpcpy (yyfmt, yyprefix);
         yyprefix = yyor;
       }

yyf = YY_(yyformat);
yysize1 = yysize + yystrlen (yyf);
```

```
    yysize_overflow |= (yysize1 < yysize);
    yysize = yysize1;

    if (yysize_overflow)
        return YYSIZE_MAXIMUM;

    if (yyresult)
        {
         /* Avoid sprintf, as that infringes on the user's name space.
            Don't have undefined behavior even if the translation
            produced a string with the wrong number of "%s"s.  */
         char *yyp = yyresult;
         int yyi = 0;
         while ((*yyp = *yyf) != '\0')
           {
             if (*yyp == '%' && yyf[1] == 's' && yyi < yycount)
                 {
                  yyp += yytnamerr (yyp, yyarg[yyi++]);
                  yyf += 2;
                 }
             else
                 {
                  yyp++;
                  yyf++;
                 }
           }
        }
    return yysize;
   }
}
```

```
#endif /* YYERROR_VERBOSE */
```

```
/*-----------------------------------------------.
| Release the memory associated to this symbol.  |
`-----------------------------------------------*/

/*ARGSUSED*/
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
static void
yydestruct (const char *yymsg, int yytype, YYSTYPE *yyvaluep)
#else
static void
yydestruct (yymsg, yytype, yyvaluep)
    const char *yymsg;
    int yytype;
    YYSTYPE *yyvaluep;
#endif
{
  YYUSE (yyvaluep);

  if (!yymsg)
    yymsg = "Deleting";
  YY_SYMBOL_PRINT (yymsg, yytype, yyvaluep, yylocationp);

  switch (yytype)
    {

      default:
          break;
```

```
    }
}

/* Prevent warnings from -Wmissing-prototypes.  */
#ifdef YYPARSE_PARAM
#if defined __STDC__ || defined __cplusplus
int yyparse (void *YYPARSE_PARAM);
#else
int yyparse ();
#endif
#else /* ! YYPARSE_PARAM */
#if defined __STDC__ || defined __cplusplus
int yyparse (void);
#else
int yyparse ();
#endif
#endif /* ! YYPARSE_PARAM */


/* The lookahead symbol.  */
int yychar;

/* The semantic value of the lookahead symbol.  */
YYSTYPE yylval;

/* Number of syntax errors so far.  */
int yynerrs;
```

```
/*----------.
| yyparse or yypush_parse.  |
`----------*/

#ifdef YYPARSE_PARAM
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
int
yyparse (void *YYPARSE_PARAM)
#else
int
yyparse (YYPARSE_PARAM)
    void *YYPARSE_PARAM;
#endif
#else /* ! YYPARSE_PARAM */
#if (defined __STDC__ || defined __C99__FUNC__ \
     || defined __cplusplus || defined _MSC_VER)
int
yyparse (void)
#else
int
yyparse ()

#endif
#endif
{



    int yystate;
    /* Number of tokens to shift before error messages enabled.  */
```

```
    int yyerrstatus;

    /* The stacks and their tools:
       `yyss': related to states.
       `yyvs': related to semantic values.

       Refer to the stacks thru separate pointers, to allow yyoverflow
       to reallocate them elsewhere.  */

    /* The state stack.  */
    yytype_int16 yyssa[YYINITDEPTH];
    yytype_int16 *yyss;
    yytype_int16 *yyssp;

    /* The semantic value stack.  */
    YYSTYPE yyvsa[YYINITDEPTH];
    YYSTYPE *yyvs;
    YYSTYPE *yyvsp;

    YYSIZE_T yystacksize;

  int yyn;
  int yyresult;
  /* Lookahead token as an internal (translated) token number.  */
  int yytoken;
  /* The variables used to return semantic value and location from the
     action routines.  */
  YYSTYPE yyval;

#if YYERROR_VERBOSE
```

```
  /* Buffer for error messages, and its allocated size.  */
  char yymsgbuf[128];
  char *yymsg = yymsgbuf;
  YYSIZE_T yymsg_alloc = sizeof yymsgbuf;
#endif

#define YYPOPSTACK(N)   (yyvsp -= (N), yyssp -= (N))

  /* The number of symbols on the RHS of the reduced rule.
     Keep to zero when no symbol should be popped.  */
  int yylen = 0;

  yytoken = 0;
  yyss = yyssa;
  yyvs = yyvsa;
  yystacksize = YYINITDEPTH;

  YYDPRINTF ((stderr, "Starting parse\n"));

  yystate = 0;
  yyerrstatus = 0;
  yynerrs = 0;
  yychar = YYEMPTY; /* Cause a token to be read.  */

  /* Initialize stack pointers.
     Waste one element of value and location stack
     so that they stay on the same level as the state stack.
     The wasted elements are never initialized.  */
  yyssp = yyss;
  yyvsp = yyvs;
```

```
  goto yysetstate;


/*------------------------------------------------------------.
| yynewstate -- Push a new state, which is found in yystate.  |
`------------------------------------------------------------*/
 yynewstate:
  /* In all cases, when you get here, the value and location stacks
     have just been pushed.  So pushing a state here evens the stacks.  */
  yyssp++;


 yysetstate:
  *yyssp = yystate;

  if (yyss + yystacksize - 1 <= yyssp)
    {
      /* Get the current used size of the three stacks, in elements.  */
      YYSIZE_T yysize = yyssp - yyss + 1;

#ifdef yyoverflow
      {
          /* Give user a chance to reallocate the stack.  Use copies of
             these so that the &'s don't force the real ones into
             memory.  */
          YYSTYPE *yyvs1 = yyvs;
          yytype_int16 *yyss1 = yyss;

          /* Each stack pointer address is followed by the size of the
             data in use in that stack, in bytes.  This used to be a
             conditional around just the two extra args, but that might
```

```
           be undefined if yyoverflow is a macro.  */
        yyoverflow (YY_("memory exhausted"),
                    &yyss1, yysize * sizeof (*yyssp),
                    &yyvs1, yysize * sizeof (*yyvsp),
                    &yystacksize);

        yyss = yyss1;
        yyvs = yyvs1;
    }
#else /* no yyoverflow */
# ifndef YYSTACK_RELOCATE
    goto yyexhaustedlab;
# else
    /* Extend the stack our own way.  */
    if (YYMAXDEPTH <= yystacksize)
        goto yyexhaustedlab;
    yystacksize *= 2;
    if (YYMAXDEPTH < yystacksize)
        yystacksize = YYMAXDEPTH;

    {
        yytype_int16 *yyss1 = yyss;
        union yyalloc *yyptr =
          (union yyalloc *) YYSTACK_ALLOC (YYSTACK_BYTES (yystacksize));
        if (! yyptr)
          goto yyexhaustedlab;
        YYSTACK_RELOCATE (yyss_alloc, yyss);
        YYSTACK_RELOCATE (yyvs_alloc, yyvs);
#  undef YYSTACK_RELOCATE
        if (yyss1 != yyssa)
```

```
            YYSTACK_FREE (yyss1);
      }
# endif
#endif /* no yyoverflow */

      yyssp = yyss + yysize - 1;
      yyvsp = yyvs + yysize - 1;

      YYDPRINTF ((stderr, "Stack size increased to %lu\n",
                  (unsigned long int) yystacksize));

      if (yyss + yystacksize - 1 <= yyssp)
        YYABORT;
    }

  YYDPRINTF ((stderr, "Entering state %d\n", yystate));

  if (yystate == YYFINAL)
    YYACCEPT;

  goto yybackup;

/*-----------.
| yybackup.  |
`-----------*/
yybackup:

  /* Do appropriate processing given the current state.  Read a
     lookahead token if we need one and don't already have one.  */
```

```
/* First try to decide what to do without reference to lookahead token.  */
yyn = yypact[yystate];
if (yyn == YYPACT_NINF)
  goto yydefault;

/* Not known => get a lookahead token if don't already have one.  */

/* YYCHAR is either YYEMPTY or YYEOF or a valid lookahead symbol.  */
if (yychar == YYEMPTY)
  {
    YYDPRINTF ((stderr, "Reading a token: "));
    yychar = YYLEX;
  }

if (yychar <= YYEOF)
  {
    yychar = yytoken = YYEOF;
    YYDPRINTF ((stderr, "Now at end of input.\n"));
  }
else
  {
    yytoken = YYTRANSLATE (yychar);
    YY_SYMBOL_PRINT ("Next token is", yytoken, &yylval, &yylloc);
  }

/* If the proper action on seeing token YYTOKEN is to reduce or to
   detect an error, take that action.  */
yyn += yytoken;
if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)
  goto yydefault;
```

```
  yyn = yytable[yyn];
  if (yyn <= 0)
    {
      if (yyn == 0 || yyn == YYTABLE_NINF)
          goto yyerrlab;
      yyn = -yyn;
      goto yyreduce;
    }

  /* Count tokens shifted since error; after three, turn off error
     status.  */
  if (yyerrstatus)
    yyerrstatus--;

  /* Shift the lookahead token.  */
  YY_SYMBOL_PRINT ("Shifting", yytoken, &yylval, &yylloc);

  /* Discard the shifted token.  */
  yychar = YYEMPTY;

  yystate = yyn;
  *++yyvsp = yylval;

  goto yynewstate;


/*-----------------------------------------------------------.
| yydefault -- do the default action for the current state.  |
`-----------------------------------------------------------*/
yydefault:
```

```
  yyn = yydefact[yystate];
  if (yyn == 0)
    goto yyerrlab;
  goto yyreduce;


/*----------------------------.
| yyreduce -- Do a reduction.  |
`----------------------------*/
yyreduce:
  /* yyn is the number of a rule to reduce with.  */
  yylen = yyr2[yyn];

  /* If YYLEN is nonzero, implement the default value of the action:
     `$$ = $1'.

     Otherwise, the following line sets YYVAL to garbage.
     This behavior is undocumented and Bison
     users should not rely upon it.  Assigning to YYVAL
     unconditionally makes the parser a bit smaller, and it avoids a
     GCC warning that YYVAL may be used uninitialized.  */
  yyval = yyvsp[1-yylen];


  YY_REDUCE_PRINT (yyn);
  switch (yyn)
    {
        case 2:

/* Line 1455 of yacc.c  */
```

```
#line 31 "calc.y"
    {;;}
    break;

  case 3:

/* Line 1455 of yacc.c  */
#line 32 "calc.y"
    {exit(EXIT_SUCCESS);;}
    break;

  case 4:

/* Line 1455 of yacc.c  */
#line 33 "calc.y"
    {;;}
    break;

  case 5:

/* Line 1455 of yacc.c  */
#line 34 "calc.y"
    {exit(EXIT_SUCCESS);;}
    break;

  case 6:

/* Line 1455 of yacc.c  */
#line 37 "calc.y"
    {fprint();;}
```

```
    break;

  case 7:

/* Line 1455 of yacc.c  */
#line 38 "calc.y"
    {fpush((yyvsp[(2) - (2)].num));;}
    break;

  case 8:

/* Line 1455 of yacc.c  */
#line 39 "calc.y"
    {fpop();;}
    break;

  case 9:

/* Line 1455 of yacc.c  */
#line 40 "calc.y"
    {fadd();;}
    break;

  case 10:

/* Line 1455 of yacc.c  */
#line 41 "calc.y"
    {fmul();;}
    break;
```

```
  case 11:

/* Line 1455 of yacc.c  */
#line 42 "calc.y"
    {fsub();;}
    break;

  case 12:

/* Line 1455 of yacc.c  */
#line 43 "calc.y"
    {fdiv();;}
    break;



/* Line 1455 of yacc.c  */
#line 1424 "calc.tab.c"
      default: break;
    }
  YY_SYMBOL_PRINT ("-> $$ =", yyr1[yyn], &yyval, &yyloc);

  YYPOPSTACK (yylen);
  yylen = 0;
  YY_STACK_PRINT (yyss, yyssp);

  *++yyvsp = yyval;

  /* Now `shift' the result of the reduction.  Determine what state
     that goes to, based on the state we popped back to and the rule
```

```
     number reduced by.  */

  yyn = yyr1[yyn];

  yystate = yypgoto[yyn - YYNTOKENS] + *yyssp;
  if (0 <= yystate && yystate <= YYLAST && yycheck[yystate] == *yyssp)
    yystate = yytable[yystate];
  else
    yystate = yydefgoto[yyn - YYNTOKENS];

  goto yynewstate;


/*------------------------------------.
| yyerrlab -- here on detecting error |
`------------------------------------*/
yyerrlab:
  /* If not already recovering from an error, report this error.  */
  if (!yyerrstatus)
    {
      ++yynerrs;
#if ! YYERROR_VERBOSE
      yyerror (YY_("syntax error"));
#else
      {
          YYSIZE_T yysize = yysyntax_error (0, yystate, yychar);
          if (yymsg_alloc < yysize && yymsg_alloc < YYSTACK_ALLOC_MAXIMUM)
            {
              YYSIZE_T yyalloc = 2 * yysize;
              if (! (yysize <= yyalloc && yyalloc <= YYSTACK_ALLOC_MAXIMUM))
```

```
            yyalloc = YYSTACK_ALLOC_MAXIMUM;
          if (yymsg != yymsgbuf)
            YYSTACK_FREE (yymsg);
          yymsg = (char *) YYSTACK_ALLOC (yyalloc);
          if (yymsg)
            yymsg_alloc = yyalloc;
          else
            {
                yymsg = yymsgbuf;
                yymsg_alloc = sizeof yymsgbuf;
            }
        }

      if (0 < yysize && yysize <= yymsg_alloc)
        {
          (void) yysyntax_error (yymsg, yystate, yychar);
          yyerror (yymsg);
        }
      else
        {
          yyerror (YY_("syntax error"));
          if (yysize != 0)
            goto yyexhaustedlab;
        }
    }
#endif
  }
```

```
  if (yyerrstatus == 3)
    {
      /* If just tried and failed to reuse lookahead token after an
           error, discard it.  */

      if (yychar <= YYEOF)
          {
            /* Return failure if at end of input.  */
            if (yychar == YYEOF)
              YYABORT;
          }
      else
          {
            yydestruct ("Error: discarding",
                        yytoken, &yylval);
            yychar = YYEMPTY;
          }
    }

  /* Else will try to reuse lookahead token after shifting the error
     token.  */
  goto yyerrlab1;


/*---------------------------------------------------.
| yyerrorlab -- error raised explicitly by YYERROR.  |
`---------------------------------------------------*/
yyerrorlab:

  /* Pacify compilers like GCC when the user code never invokes
```

```
     YYERROR and the label yyerrorlab therefore never appears in user
     code.  */
  if (/*CONSTCOND*/ 0)
     goto yyerrorlab;

  /* Do not reclaim the symbols of the rule which action triggered
     this YYERROR.  */
  YYPOPSTACK (yylen);
  yylen = 0;
  YY_STACK_PRINT (yyss, yyssp);
  yystate = *yyssp;
  goto yyerrlab1;


/*-------------------------------------------------------------.
| yyerrlab1 -- common code for both syntax error and YYERROR.  |
`-------------------------------------------------------------*/
yyerrlab1:
  yyerrstatus = 3;        /* Each real token shifted decrements this.  */

  for (;;)
    {
      yyn = yypact[yystate];
      if (yyn != YYPACT_NINF)
           {
             yyn += YYTERROR;
             if (0 <= yyn && yyn <= YYLAST && yycheck[yyn] == YYTERROR)
               {
                 yyn = yytable[yyn];
                 if (0 < yyn)
```

```
                    break;
                }
            }

      /* Pop the current state because it cannot handle the error token.  */
      if (yyssp == yyss)
          YYABORT;


      yydestruct ("Error: popping",
                  yystos[yystate], yyvsp);
      YYPOPSTACK (1);
      yystate = *yyssp;
      YY_STACK_PRINT (yyss, yyssp);
    }

  *++yyvsp = yylval;


  /* Shift the error token.  */
  YY_SYMBOL_PRINT ("Shifting", yystos[yyn], yyvsp, yylsp);

  yystate = yyn;
  goto yynewstate;


/*-------------------------------------.
| yyacceptlab -- YYACCEPT comes here.  |
`-------------------------------------*/
yyacceptlab:
```

```
  yyresult = 0;
  goto yyreturn;


/*-----------------------------------.
| yyabortlab -- YYABORT comes here.  |
`-----------------------------------*/
yyabortlab:
  yyresult = 1;
  goto yyreturn;


#if !defined(yyoverflow) || YYERROR_VERBOSE
/*-------------------------------------------------.
| yyexhaustedlab -- memory exhaustion comes here.  |
`-------------------------------------------------*/
yyexhaustedlab:
  yyerror (YY_("memory exhausted"));
  yyresult = 2;
  /* Fall through.  */
#endif


yyreturn:
  if (yychar != YYEMPTY)
     yydestruct ("Cleanup: discarding lookahead",
                 yytoken, &yylval);
  /* Do not reclaim the symbols of the rule which action triggered
     this YYABORT or YYACCEPT.  */
  YYPOPSTACK (yylen);
  YY_STACK_PRINT (yyss, yyssp);
  while (yyssp != yyss)
    {
```

```
      yydestruct ("Cleanup: popping",
                  yystos[*yyssp], yyvsp);
      YYPOPSTACK (1);
    }
#ifndef yyoverflow
  if (yyss != yyssa)
    YYSTACK_FREE (yyss);
#endif
#if YYERROR_VERBOSE
  if (yymsg != yymsgbuf)
    YYSTACK_FREE (yymsg);
#endif
  /* Make sure YYID is used.  */
  return YYID (yyresult);
}



/* Line 1675 of yacc.c  */
#line 46 "calc.y"


void fprint() {
        printf("Top of stack is: %d\n", stack[size]);
}

void fpush(int val)
{
        if(size < 60) {
                stack[++size] = val;
```

```
                /* printf("Added"); */
                return;
        }
        /* printf("Stack is full\n"); */
}

int fpop()
{
        if(size > -1) {
                return stack[size--];
        }
        printf("Stack is empty\n");
        return -1;
}

void fadd()
{       if(size > 0) {
                fpush(fpop() + fpop());
        } else {
        printf("Stack does not contain enough elements\n");
        }
}

void fmul()
{
        if(size > 0) {
                fpush(fpop() * fpop());
        } else {
        printf("Stack does not contain enough elements\n");
        }
```

```
}

void fsub()
{
	if(size > 0) {
		int ope2 = fpop();
		fpush(fpop() - ope2);
	} else {
	printf("Stack does not contain enough elements\n");
	}
}

void fdiv()
{
	if(size > 0) {
		int ope2 = fpop();
		fpush(fpop() / ope2);
	} else {
	printf("Stack does not contain enough elements\n");
	}

}


int main (void) {
	printf("Commands are: push, pop, add, mul, sub, divide and print.\n");
	int i;
	for(i=0; i<60; i++) {
		stack[i] = 0;
	}
```

```
        return yyparse();
}

void yyerror (const char *s) {fprintf(stderr, "%s\n", s);}
```