```
# $Id: README,v 1.2 2006-10-16 12:17:39-07 - - $

This is a simple calculator with some arithmetic operations
having the usual priorities and a symbol table of one-letter
variable names.

It is an example of the use of bison and flex used to
generate an output file from test data.  It also illustrates
how to use a Makefile with bison, flex, and gcc.

This example is definitely overkill on the user of many
files.  However, each of the files present illustrate a part
of a compiler and their counterpart in your project will be
much larger.

```

```
// $Id: extern.h,v 1.4 2014-10-24 16:32:04-07 - - $

#ifndef __EXTERN_H__
#define __EXTERN_H__

//
// Include some things from STL.
//
#include <map>
#include <string>
using namespace std;

//
// A more compact representation should be used for efficiency.
// No token ever has both a symbol and a value.
//
struct yystype {
   string sym;
   double val;
};
extern map<string,double> symtab;

//
// External symbols.
//
extern int yy_flex_debug;
extern int yydebug;
void error (const string& message, const string& data);
void yyerror (const string& message);
int yylex (void);
int yyparse (void);
double sym_get (const string& symbol);
double sym_put (const string& symbol, double value);

//
// Include parser-generated symbols.
//
#define YYSTYPE yystype
#include "parser.h"

#endif

```

```

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

/* Skeleton interface 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.  */


/* Tokens.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
   /* Put the tokens into the symbol table, so that GDB and other debuggers
      know about them.  */
   enum yytokentype {
     IDENT = 258,
     NUMBER = 259,
     UNARY = 260
   };
#endif



#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif
```

```

extern YYSTYPE yylval;


```

```
/* $Id: main.cc,v 1.7 2014-10-24 16:32:04-07 - - $ */


#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "extern.h"

map<string,double> symtab;
const double NAN = 0.0 / 0.0;

double sym_get (const string& symbol) {
   auto entry = symtab.find (symbol);
   if (entry != symtab.end()) return entry->second;
   error ("variable is uninitialized", symbol);
   return NAN;
}

double sym_put (const string& symbol, double value) {
   symtab[symbol] = value;
   return value;
}

void sym_dump() {
   for (auto iter = symtab.cbegin(); iter != symtab.cend(); ++iter) {
      printf ("symtab[%s] = %.10g\n",
              iter->first.c_str(), iter->second);
   }
}

void scan_options (int argc, char** argv) {
   yy_flex_debug = yydebug = 0;
   for(;;) {
      int option = getopt (argc, argv, "ly");
      switch (option) {
         case EOF: return;
         case 'l': yy_flex_debug = 1; break;
         case 'y': yydebug       = 1; break;
      }
   }
}

void error (const string& message, const string& data) {
   yyerror (message + " (" + data + ")");
}

void yyerror (const string& message) {
   printf (" ... [[%s]]\n", message.c_str());
}

int main (int argc, char** argv) {
   scan_options (argc, argv);
   int parse_rc = yyparse ();
   sym_dump();
   return parse_rc;
}

```

```

%{
// $Id: scanner.l,v 1.5 2014-10-24 16:37:47-07 - - $

#include <ctype.h>
#include <stdlib.h>

#include "extern.h"

%}

%option 8bit
%option debug
%option ecs
%option nodefault
%option noinput
%option nounput
%option noyywrap
%option warn

DIGIT       [0-9]
FRACTION    ({DIGIT}+\.?{DIGIT}*|\.{DIGIT}+)
EXPONENT    ([Ee][+-]?{DIGIT}+)
NUMBER      {FRACTION}({EXPONENT})?
ERRORNUM    {FRACTION}[Ee][+-]?
LETTER      [a-zA-Z_]
IDENT       {LETTER}({LETTER}|{DIGIT})*

%%

{IDENT}       { ECHO; yylval.sym = yytext; return IDENT; }
{NUMBER}      { ECHO; yylval.val = atof (yytext); return NUMBER; }
"("           { ECHO; return '('; }
")"           { ECHO; return ')'; }
"+"           { ECHO; return '+'; }
"-"           { ECHO; return '-'; }
"/"           { ECHO; return '/'; }
"*"           { ECHO; return '*'; }
"="           { ECHO; return '='; }
\n            { ECHO; return '\n'; }
[\t ]+        { ECHO; }
"#".*         { ECHO; }
{ERRORNUM}    { ECHO; error ("invalid numeric value", yytext); }
.             { ECHO; error ("invalid input character", yytext); }

%%

```

```
%{
// $Id: parser.y,v 1.3 2014-10-24 16:32:04-07 - - $

#include "extern.h"

#define YYDEBUG 1
#define YYERROR_VERBOSE 1

%}

%debug
%defines
%token-table
%verbose

%token  IDENT NUMBER
%right  '='
%left   '+' '-'
%left   '*' '/'
%right  UNARY

%start  stmts

%%

stmts : stmts stmt           { }
      |                      { }
      ;

stmt  : expr '\n'            { printf ("****EXPR==%.10g\n", $1.val); }
      | error '\n'           { printf ("****ERROR #%d\n", yynerrs); }
      | '\n'                 { printf ("\n"); }
      ;

expr  : IDENT '=' expr       { $$.val = sym_put ($1.sym, $3.val); }
      | expr '+' expr        { $$.val = $1.val + $3.val; }
      | expr '-' expr        { $$.val = $1.val - $3.val; }
      | expr '*' expr        { $$.val = $1.val * $3.val; }
      | expr '/' expr        { $$.val = $1.val / $3.val; }
      | '+' expr %prec UNARY { $$.val = + $2.val; }
      | '-' expr %prec UNARY { $$.val = - $2.val; }
      | '(' expr ')'         { $$.val = $2.val; }
      | NUMBER               { $$.val = $1.val; }
      | IDENT                { $$.val = sym_get ($1.sym); }
      ;

%%

```

```
# $Id: Makefile,v 1.12 2014-10-24 16:28:32-07 - - $

GPP      = g++ -g -O0 -Wall -Wextra -std=gnu++0x
GPPNW    = g++ -g -O0 -std=gnu++0x
GPPDEP   = g++ -MM

TXTS    = README
HDRS    = extern.h
SRCS    = main.cc scanner.l parser.y
GENS    = parser.h scanner.cc parser.cc
OBJS    = main.o scanner.o parser.o
BINS    = calculator
LOGS    = scanner.log parser.log

OUT1    = test1.out test1.err
OUT2    = test2.out test2.err
OUTS    = ${OUT1} ${OUT2}
IN1     = test1.in ${OUT1}
IN2     = test2.in ${OUT2}
INS     = ${IN1} ${IN2}
OUTPUT  = test1.lis test2.lis
LISTS   = Listing.pdf Listing.ps
MAKES   = Makefile Makefile.deps
DEPS    = ${filter %.cc, ${SRCS} ${GENS}}
NOINCL  = ${filter ci clean spotless, ${MAKECMDGOALS}}

LISTING = ${TXTS} ${HDRS} parser.h ${SRCS} ${MAKES} ${OUTPUT}
CLEAN   = core ${OBJS} ${GENS} ${LOGS} ${OUTS} ${LISTS}
RCS     = ${TXTS} ${HDRS} ${SRCS} Makefile test*.in

define MORE
MORE() { \
   LIS=$$1; shift; \
   for i in $$*; do \
      echo ::::::::::::::::; \
      echo $$i; \
      echo ::::::::::::::::; \
      cat -nv $$i; \
   done >$$LIS; \
}
endef

```

```

all: ${BINS}

clean:
        - rm ${CLEAN}

spotless: clean
        - rm ${GENS} ${BINS} Makefile.deps ${OUTPUT}

deps:
        - rm Makefile.deps
        ${MAKE} --no-print-directory Makefile.deps

out: ${OUTS}

lis: out
        mkpspdf Listing.ps ${LISTING}

calculator: ${OBJS}
        ${GPP} -o calculator ${OBJS}

%.o: %.cc
        ${GPP} ${CFLAGS} $< -c

scanner.cc: scanner.l
        flex -oscanner.cc scanner.l >scanner.log 2>&1
        - cat lex.backup >>scanner.log
        - rm lex.backup

parser.cc parser.h: parser.y
        bison -o parser.cc parser.y
        - mv parser.hh parser.h
        - mv parser.output parser.log

test1.lis test1.out test1.err: ${BINS} test1.in
        ./calculator <test1.in >test1.out 2>test1.err
        ${MORE}; MORE test1.lis test1.in test1.out test1.err

test2.lis test2.out test2.err: ${BINS} test2.in
        ./calculator -ly <test2.in >test2.out 2>test2.err
        ${MORE}; MORE test2.lis test2.in test2.out test2.err

ci : ${RCS}
        cid + ${RCS}

Makefile.deps: Makefile ${GENS}
        ${GPP} -MM ${DEPS} >Makefile.deps

again :
        ${MAKE} --no-print-directory spotless ci all lis

ifeq (${NOINCL},)
include Makefile.deps
endif

```

```
main.o: main.cc extern.h parser.h
scanner.o: scanner.cc extern.h parser.h
parser.o: parser.cc extern.h parser.h
```

```
::::::::::::::::
test1.in
::::::::::::::::
     1  # $Id: test1.in,v 1.2 2013-09-05 20:24:24-07 - - $
     2  alpha=3+4*5-6
     3  crap out
     4  beta=alpha/3
     5  gamma=beta*10-alpha*33
     6  64e *98
     7  infinity=1e1000*1e1000
     8  nan=infinity/infinity
     9  foo*bar
::::::::::::::::
test1.out
::::::::::::::::
     1  # $Id: test1.in,v 1.2 2013-09-05 20:24:24-07 - - $
     2  
     3  alpha=3+4*5-6
     4  ****EXPR==17
     5  crap out ... [[variable is uninitialized (crap)]]
     6   ... [[syntax error, unexpected IDENT]]
     7  
     8  ****ERROR #1
     9  beta=alpha/3
    10  ****EXPR==5.666666667
    11  gamma=beta*10-alpha*33
    12  ****EXPR==-504.3333333
    13  64e ... [[invalid numeric value (64e)]]
    14   * ... [[syntax error, unexpected '*']]
    15  98
    16  ****ERROR #2
    17  infinity=1e1000*1e1000
    18  ****EXPR==inf
    19  nan=infinity/infinity
    20  ****EXPR==-nan
    21  foo* ... [[variable is uninitialized (foo)]]
    22  bar
    23   ... [[variable is uninitialized (bar)]]
    24  ****EXPR==-nan
    25  symtab[alpha] = 17
    26  symtab[beta] = 5.666666667
    27  symtab[gamma] = -504.3333333
    28  symtab[infinity] = inf
    29  symtab[nan] = -nan
::::::::::::::::
test1.err
::::::::::::::::
```

```
::::::::::::::::
test2.in
::::::::::::::::
     1  # $Id: test2.in,v 1.1 2013-09-05 19:21:46-07 - - $
     2  a=3+4*5-6
     3  b=a/3
::::::::::::::::
test2.out
::::::::::::::::
     1  # $Id: test2.in,v 1.1 2013-09-05 19:21:46-07 - - $
     2  
     3  a=3+4*5-6
     4  ****EXPR==17
     5  b=a/3
     6  ****EXPR==5.666666667
     7  symtab[a] = 17
     8  symtab[b] = 5.666666667
::::::::::::::::
test2.err
::::::::::::::::
     1  Starting parse
     2  Entering state 0
     3  Reducing stack by rule 2 (line 27):
     4  -> $$ = nterm stmts ()
     5  Stack now 0
     6  Entering state 1
     7  Reading a token: --(end of buffer or a NUL)
     8  --accepting rule at line 42 ("# $Id: test2.in,v 1.1 2013-09-05 19:21:46-07 - - $")
     9  --accepting rule at line 40 ("
    10  ")
    11  Next token is token '\n' ()
    12  Shifting token '\n' ()
    13  Entering state 8
    14  Reducing stack by rule 5 (line 32):
    15     $1 = token '\n' ()
    16  -> $$ = nterm stmt ()
    17  Stack now 0 1
    18  Entering state 10
    19  Reducing stack by rule 1 (line 26):
    20     $1 = nterm stmts ()
    21     $2 = nterm stmt ()
    22  -> $$ = nterm stmts ()
    23  Stack now 0
    24  Entering state 1
    25  Reading a token: --accepting rule at line 31 ("a")
    26  Next token is token IDENT ()
    27  Shifting token IDENT ()
    28  Entering state 4
    29  Reading a token: --accepting rule at line 39 ("=")
    30  Next token is token '=' ()
    31  Shifting token '=' ()
    32  Entering state 13
    33  Reading a token: --accepting rule at line 32 ("3")
    34  Next token is token NUMBER ()
    35  Shifting token NUMBER ()
    36  Entering state 5
    37  Reducing stack by rule 14 (line 43):
```

```
   38     $1 = token NUMBER ()
   39  -> $$ = nterm expr ()
   40  Stack now 0 1 4 13
   41  Entering state 22
   42  Reading a token: --accepting rule at line 35 ("+")
   43  Next token is token '+' ()
   44  Shifting token '+' ()
   45  Entering state 17
   46  Reading a token: --accepting rule at line 32 ("4")
   47  Next token is token NUMBER ()
   48  Shifting token NUMBER ()
   49  Entering state 5
   50  Reducing stack by rule 14 (line 43):
   51     $1 = token NUMBER ()
   52  -> $$ = nterm expr ()
   53  Stack now 0 1 4 13 22 17
   54  Entering state 24
   55  Reading a token: --accepting rule at line 38 ("*")
   56  Next token is token '*' ()
   57  Shifting token '*' ()
   58  Entering state 19
   59  Reading a token: --accepting rule at line 32 ("5")
   60  Next token is token NUMBER ()
   61  Shifting token NUMBER ()
   62  Entering state 5
   63  Reducing stack by rule 14 (line 43):
   64     $1 = token NUMBER ()
   65  -> $$ = nterm expr ()
   66  Stack now 0 1 4 13 22 17 24 19
   67  Entering state 26
   68  Reducing stack by rule 9 (line 38):
   69     $1 = nterm expr ()
   70     $2 = token '*' ()
   71     $3 = nterm expr ()
   72  -> $$ = nterm expr ()
   73  Stack now 0 1 4 13 22 17
   74  Entering state 24
   75  Reading a token: --accepting rule at line 36 ("-")
   76  Next token is token '-' ()
   77  Reducing stack by rule 7 (line 36):
   78     $1 = nterm expr ()
   79     $2 = token '+' ()
   80     $3 = nterm expr ()
   81  -> $$ = nterm expr ()
   82  Stack now 0 1 4 13
   83  Entering state 22
   84  Next token is token '-' ()
   85  Shifting token '-' ()
   86  Entering state 18
   87  Reading a token: --accepting rule at line 32 ("6")
   88  Next token is token NUMBER ()
   89  Shifting token NUMBER ()
   90  Entering state 5
   91  Reducing stack by rule 14 (line 43):
   92     $1 = token NUMBER ()
   93  -> $$ = nterm expr ()
   94  Stack now 0 1 4 13 22 18
   95  Entering state 25
```

```
 96  Reading a token: --accepting rule at line 40 ("
 97  ")
 98  Next token is token '\n' ()
 99  Reducing stack by rule 8 (line 37):
100     $1 = nterm expr ()
101     $2 = token '-' ()
102     $3 = nterm expr ()
103  -> $$ = nterm expr ()
104  Stack now 0 1 4 13
105  Entering state 22
106  Next token is token '\n' ()
107  Reducing stack by rule 6 (line 35):
108     $1 = token IDENT ()
109     $2 = token '=' ()
110     $3 = nterm expr ()
111  -> $$ = nterm expr ()
112  Stack now 0 1
113  Entering state 11
114  Next token is token '\n' ()
115  Shifting token '\n' ()
116  Entering state 21
117  Reducing stack by rule 3 (line 30):
118     $1 = nterm expr ()
119     $2 = token '\n' ()
120  -> $$ = nterm stmt ()
121  Stack now 0 1
122  Entering state 10
123  Reducing stack by rule 1 (line 26):
124     $1 = nterm stmts ()
125     $2 = nterm stmt ()
126  -> $$ = nterm stmts ()
127  Stack now 0
128  Entering state 1
129  Reading a token: --accepting rule at line 31 ("b")
130  Next token is token IDENT ()
131  Shifting token IDENT ()
132  Entering state 4
133  Reading a token: --accepting rule at line 39 ("=")
134  Next token is token '=' ()
135  Shifting token '=' ()
136  Entering state 13
137  Reading a token: --accepting rule at line 31 ("a")
138  Next token is token IDENT ()
139  Shifting token IDENT ()
140  Entering state 4
141  Reading a token: --accepting rule at line 37 ("/")
142  Next token is token '/' ()
143  Reducing stack by rule 15 (line 44):
144     $1 = token IDENT ()
145  -> $$ = nterm expr ()
146  Stack now 0 1 4 13
147  Entering state 22
148  Next token is token '/' ()
149  Shifting token '/' ()
150  Entering state 20
151  Reading a token: --accepting rule at line 32 ("3")
152  Next token is token NUMBER ()
153  Shifting token NUMBER ()
```

```
154  Entering state 5
155  Reducing stack by rule 14 (line 43):
156     $1 = token NUMBER ()
157  -> $$ = nterm expr ()
158  Stack now 0 1 4 13 22 20
159  Entering state 27
160  Reducing stack by rule 10 (line 39):
161     $1 = nterm expr ()
162     $2 = token '/' ()
163     $3 = nterm expr ()
164  -> $$ = nterm expr ()
165  Stack now 0 1 4 13
166  Entering state 22
167  Reading a token: --accepting rule at line 40 ("
168  ")
169  Next token is token '\n' ()
170  Reducing stack by rule 6 (line 35):
171     $1 = token IDENT ()
172     $2 = token '=' ()
173     $3 = nterm expr ()
174  -> $$ = nterm expr ()
175  Stack now 0 1
176  Entering state 11
177  Next token is token '\n' ()
178  Shifting token '\n' ()
179  Entering state 21
180  Reducing stack by rule 3 (line 30):
181     $1 = nterm expr ()
182     $2 = token '\n' ()
183  -> $$ = nterm stmt ()
184  Stack now 0 1
185  Entering state 10
186  Reducing stack by rule 1 (line 26):
187     $1 = nterm stmts ()
188     $2 = nterm stmt ()
189  -> $$ = nterm stmts ()
190  Stack now 0
191  Entering state 1
192  Reading a token: --(end of buffer or a NUL)
193  --EOF (start condition 0)
194  Now at end of input.
195  Shifting token $end ()
196  Entering state 2
197  Stack now 0 1 2
198  Cleanup: popping token $end ()
199  Cleanup: popping nterm stmts ()
```