Project p02 is a Subset Pascal Parser. Specifications for the scanner are on the class web page http://cs2.uco.edu/~trt/cs4023/p02.pdf. Directions for submission (http://cs2.uco.edu/~trt/cs4023/ProjectSubmission.pdf) and a project submission template are also on the class web page.

In this lecture, we have an example of a simple scanner for a language that is note quite trivial. We hope that you will be able to use this example as a basis for creating the Subset Pascal Scanner.

ID	Left-hand side		Right-hand side
	program	→	**begin** *statement-list* **end**
	statement-list	→	*statement*
	statement-list	→	*statement-list* **;** *statement*
	statement	→	**id :=** *expression*
	statement	→	**read (** *identifier-list* **)**
	statement	→	**write (** *expression-list* **)**
	identifier-list	→	**id**
	identifier-list	→	*identifier-list* **, id**
	expression-list	→	*expression*
	expression-list	→	*expression-list* **,** *expression*
	expression	→	*primary*
	expression	→	*expression add-op primary*
	primary	→	**(** *expression* **)**
	primary	→	**id**
	primary	→	**intlit**
	add-op	→	**+**
	add-op	→	**-**

Token	Specification	Token	Specification
ID	(*letter* | _)(*letter* | *digit* | _)*	ASSIGN	**:=**
BEGAN	**begin**	COMMA	**,**
END	**end**	SEMICOLON	**;**
READ	**read**	LPAREN	**(**
WRITE	**write**	RPAREN	**)**
INTLIT	*digit*+	PLUS	**+**
		MINUS	**-**

File mkmcr

```
rm mcrlex.cpp
rm mcrpar.cpp
rm *.o
rm mcr
make -f makemcr
```

File rmmcr

```
rm mcrlex.cpp
rm mcrpar.cpp
rm *.o
rm mcr
```

File makemcr

```
#----------------------------------------------------------------------
# File makemcr creates a Micro parser
#----------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@uco.edu
# Date:   January, 2015
#----------------------------------------------------------------------
# Copyright January, 2015 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#----------------------------------------------------------------------
# Object files
#----------------------------------------------------------------------
obj     =       mcrpar.o \
                mcrlex.o \
                mcr.o
#----------------------------------------------------------------------
# Bind the subset Micro Scanneer
#----------------------------------------------------------------------
mcr:            ${obj}
                g++ -o mcr ${obj} -lm -ll
#----------------------------------------------------------------------
# File mcr.cpp processes command line arguments
#----------------------------------------------------------------------
mcr.o:          mcr.cpp mcrlex.h
                g++ -c -g mcr.cpp
#----------------------------------------------------------------------
# File mcrlex.cpp is the lex-generated scanner
#----------------------------------------------------------------------
mcrlex.cpp:     mcrlex.l mcrlex.h
                lex mcrlex.l
                mv lex.yy.c mcrlex.cpp
#----------------------------------------------------------------------
#----------------------------------------------------------------------
mcrlex.o:       mcrlex.cpp mcrlex.h
                g++ -c -g mcrlex.cpp
#----------------------------------------------------------------------
# Create files mcrpar.cpp and mcrtkn.h from file mcrpar.y
#----------------------------------------------------------------------
mcrtkn.h        \
mcrpar.cpp:     mcrpar.y
                yacc -d -v mcrpar.y
                mv y.tab.c mcrpar.cpp
                cat mcrtkn_prolog.h y.tab.h mcrtkn_epilog.h > mcrtkn.h
```

```
#-------------------------------------------------------------
# Compile the parser mcrpar.y
#-------------------------------------------------------------
mcrpar.o:       mcrpar.cpp mcrpar.h
                g++ -c -g mcrpar.cpp
#-------------------------------------------------------------
```

File mcr.cpp

```
//----------------------------------------------------------------------
//File mcr.cpp contains functions that process command line file names
//and interface with the lex-generated scanner
//----------------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: trturner@uco.edu
//Date:   January, 2015
//----------------------------------------------------------------------
//Copyright January, 2015 by Thomas R. Turner
//Do not reproduce without permission from Thomas R. Turner
//----------------------------------------------------------------------
//C++ Standard include files
//----------------------------------------------------------------------
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <string>
using namespace std;
//----------------------------------------------------------------------
//Application include files
//----------------------------------------------------------------------
#include "mcrlex.h"
#include "mcrpar.h"
//----------------------------------------------------------------------
//Externals
//----------------------------------------------------------------------
ofstream tfs;                          //trace file stream
//----------------------------------------------------------------------
//BadSuffixException
//----------------------------------------------------------------------
struct BadSuffixException {
BadSuffixException(char* fn)
    {    cout << endl;
         cout << "Input file \"" << fn << "\" does not have a .mcr suffix.";
    }
};
```

```
//-----------------------------------------------------------------
//-----------------------------------------------------------------
class FileNameSuffix {
    char* prefix;
public:
    FileNameSuffix(char* fn)
    {   char* p=strstr(fn,".mcr");
        if (!p) throw BadSuffixException(fn);
        int n=p-fn;
        if (n+4!=strlen(fn)) throw BadSuffixException(fn);
        prefix=new char[strlen(fn)+1];
        strncpy(prefix,fn,n);
        prefix[n]=0;
    }
    ~FileNameSuffix(){if (prefix) delete[] prefix;}
    void Suffix(char* fn,const char* suffix)
    {   strcpy(fn,prefix);
        strcat(fn,suffix);
    }
};
//-----------------------------------------------------------------
//CommandLineException
//-----------------------------------------------------------------
struct CommandLineException {
    CommandLineException(int m,int a)
    {   cout << endl;
        cout << "Too many file names on the command line.";
        cout << endl;
        cout << m << " file name(s) are permitted on the command line.";
        cout << endl;
        cout << a << " file name(s) appeared on the command line.";
        cout << endl;
    }
};
//-----------------------------------------------------------------
//FileException
//-----------------------------------------------------------------
struct FileException {
    FileException(const char* fn)
    {   cout << endl;
        cout << "File " << fn << " could not be opened.";
        cout << endl;
    }
};
```

```
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
void CompilerMgr(FILE* i)
{    Parser P(i);
     int rc=P.Parse();
}
//-----------------------------------------------------------------------
//Function main processes command line file names
//-----------------------------------------------------------------------
int main(int argc,char* argv[])
{    try {
          char ifn[255];
          switch (argc) {
               case 1:                    //Prompt for the input file name
                    cout << "Enter the input file name. ";
                    cin >> ifn;
                    break;
               case 2:                    //Read the input file name
                    strcpy(ifn,argv[1]);
                    break;
               default:
                    throw CommandLineException(1,argc-1);
                    break;
          }
          FileNameSuffix F(ifn);          //Find the prefix of the input file name
          char tfn[255];
          F.Suffix(tfn,".trc");           //Create the trace file name
          FILE* i=fopen(ifn,"r");         //Open the input file
          if (!i) throw FileException(ifn);
          tfs.open(tfn); if (!tfs) throw FileException(tfn);
          CompilerMgr(i);
          tfs << endl;                    //Put a new line in the trace file
          tfs.close();                    //Close the trace file
          fclose(i);                      //Close the input file
     } catch (...) {
          cout << endl;
          cout << "Program terminated!";
          cout << endl;
          cout << "I won't be back!";
          cout << endl;
          exit(EXIT_FAILURE);
     }
     return 0;
}
```

File mcrlex.h

```
#ifndef mcrlex_h
#define mcrlex_h 1
//--------------------------------------------------------------------
// File mcrlex.h defines class Lexer.
//--------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   January, 2015
//--------------------------------------------------------------------
// Copyright January, 2015 by Thomas R. Turner
// Do not reproduce without permission from Thomas R. Turner.
//--------------------------------------------------------------------
//--------------------------------------------------------------------
// Standard C and C++ include files
//--------------------------------------------------------------------
#include <cstdio>
#include <fstream>
#include <iostream>
//--------------------------------------------------------------------
//Namespaces
//--------------------------------------------------------------------
using namespace std;
//--------------------------------------------------------------------
//Function: yylex
//Function yylex is the pasner.  Function yylex returns an integer
//token code as defined above or 0 if end-of-file has been
//reached.
//--------------------------------------------------------------------
#ifdef __cplusplus
extern "C"
#endif
int yylex (void);
//--------------------------------------------------------------------
//Class Lexer defines the attributes of a Scanner
//--------------------------------------------------------------------
class Lexer {
public:
  Lexer(FILE* i);             //Constructor used to redirect the keyboard
                            //(stdin) to file i.
  int Lex(void);              //Call the scanner yylex and return the code
};
#endif
```

File mcrlex.l

```
%{
//---------------------------------------------------------------------
// File mcrlex.l defines a prototype scanner for the Micro language.
// The scanner definition is a lex specification.
//---------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner@uco.edu
// Date:   January, 2015
//---------------------------------------------------------------------
//Copyright January, 2015 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//---------------------------------------------------------------------
//---------------------------------------------------------------------
// Standard C and C++ Library Include Files
//---------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <cstdio>
#include <map>
using namespace std;
//---------------------------------------------------------------------
// Application Includes
//---------------------------------------------------------------------
#include "mcrlex.h"
#include "mcrtkn.h"


//---------------------------------------------------------------------
//Externals
//---------------------------------------------------------------------
extern ofstream tfs;                //Trace file stream tfs
//---------------------------------------------------------------------
//Global Variables
//---------------------------------------------------------------------
static map<string,int> RW;          //RW - Reserve Words
                       //RW is a table of reserve words
                       //and their corresponding
                       //token codes.
static map<int,string> TokenName;
int line=1;                    //Current line number
int col=1;                     //Current column number
//---------------------------------------------------------------------
//Functions
//---------------------------------------------------------------------
```

```
int TokenMgr(int t);                //Token post processing
int ReserveWord(char* s);           //Determines if string s is
                                    //a reserve word
void TokenTrace(int t);             //Records token t in the trace file
void PopulateRWMap(void);           //Create the table of reserve
                                    //words and their token codes.
void PopulateTokenNameMap(void);    //Create a table of names for tokens
                                    //to be printed in the TokenTrace
void ToLower(char* d,char* s);      //Coerce the characters of string s
                                    //to lower case. Put the result
                                    //in string d.
//-------------------------------------------------------------------
//Exceptions
//-------------------------------------------------------------------
//An UnterminatedCommentException is thrown when an unterminated
//comment appears in the source file.
//-------------------------------------------------------------------
struct UnterminatedCommentException {
   UnterminatedCommentException()
   {  tfs << endl;
      tfs << "An unterminated comment begins on line " <<  line
         << " and column " << col;
      tfs << endl;
   }
};
//-------------------------------------------------------------------
//A BacCharacterException is thrown when a character outside the
//defined set of characters for the Micro language appears in the
//source.
//-------------------------------------------------------------------
struct BadCharacterException{
   BadCharacterException(char p,int l,int c)
   {  cout << endl;
      cout << "line(" << l << ") col (" << c << ")" ;
      cout << "Lexical error: ";
      cout << "Illegal character |" << p << "| ASCII code=" << (int)p;
      cout << endl;
   }
};
%}
%%
[{]                          {throw UnterminatedCommentException();}
[{][^}]*[}]                     {  for (int a=0;a<yyleng;a++) {
                                   col++;
                                   if (yytext[a]=='\n')
                                   {  col=1;
                                      line++;
                                   }
```

```
                              }
                            }
[\n]                          {line++; col=1;}
[ \t]+                        {col+=yyleng;}
[_a-zA-Z][_a-zA-Z0-9]*              return TokenMgr(IDENTIFIER);
[0-9]+                       return TokenMgr(INTLIT);
":="                         return TokenMgr(ASSIGN);
";"                          return TokenMgr(SEMICOLON);
","                          return TokenMgr(COMMA);
"("                          return TokenMgr(LPAREN);
")"                          return TokenMgr(RPAREN);
"+"                          return TokenMgr(PLUS);
"-"                          return TokenMgr(MINUS);
.                            {throw BadCharacterException
                                 (*yytext
                                 ,line
                                 ,col
                                 );
                             }
%%
//---------------------------------------------------------------------
//Class Lexer implementation
//---------------------------------------------------------------------
//Function ReserveWord determines if the input string is a reserve
//word and if it is the function returns the corresponding tokencode.
//Otherwise it return the tokencode IDENTIFIER.
//---------------------------------------------------------------------
int ReserveWord(char* s)
{
   int t=RW[(string)s];
   if (t>0) return t; else return IDENTIFIER;
}
//---------------------------------------------------------------------
//Function ToLower coerces the characters of string s to lower case
//and stores the result in string d.
//---------------------------------------------------------------------
void ToLower(char* d,char* s)
{  strcpy(d,s);
   for (int a=0;a<strlen(d);a++) d[a]=tolower(d[a]);
}
//---------------------------------------------------------------------
//Function TokenMgr processes the token after it has been recognized
//---------------------------------------------------------------------
int TokenMgr(int t)
{  col+=yyleng;                  //Add the length of the current
                            //Token to the column position
   if (t==IDENTIFIER)
   {  char* s=new char[yyleng];    //Create storage for a lower case
```

```
                        //version of the token
      ToLower(s,yytext);          //Coerce the token to lower case
      yylval.token= new string(s);  //yylval is the yacc variable
                        //associated with the %union
                        //directive.  Member token was
                        //declared to have the semantic
                        //value for tokens.
      t=ReserveWord(s);           //Determie if the IDENTIFIER is a reserve word
   }
   TokenTrace(t);                 //Record the current token in the trace file
   return t;
}
//---------------------------------------------------------------------
//---------------------------------------------------------------------
//Function TokenTrace records the attributes of the input token t
//in the trace file.
//---------------------------------------------------------------------
void TokenTrace(int tkn)
{
   tfs << endl;
   tfs << "Tokentrace(";
   tfs << "line(" << setw(4) << line << ")";
   tfs << ",";
   tfs << "col(" << setw(4) << col-yyleng << ")";
   tfs << ",";
   tfs << "code(" << setw(3) << tkn << ")";
   tfs << ",";
   tfs << "name(" << setw(10) << TokenName[tkn] << ")";
   tfs << ",";
   tfs << "spelling(\"" << yytext << "\")";
   tfs << ")";
}
//---------------------------------------------------------------------
//Function PopulateRWMap assigns values to the map, RW, such that each
//reserve word is given its associated tokencode.
//---------------------------------------------------------------------
void PopulateRWMap(void)
{  RW["begin"]=BEGAN;
   RW["end"  ]=END;
   RW["read" ]=READ;
   RW["write"]=WRITE;
}
void PopulateTokenNameMap(void)
{  TokenName[BEGAN     ]="BEGAN";
   TokenName[END       ]="END";
   TokenName[READ      ]="READ";
   TokenName[WRITE     ]="WRITE";
   TokenName[INTLIT    ]="INTLIT";
```

```
   TokenName[IDENTIFIER ]="IDENTIFIER";
   TokenName[ASSIGN     ]="ASSIGN";
   TokenName[SEMICOLON  ]="SEMICOLON";
   TokenName[COMMA      ]="COMMA";
   TokenName[LPAREN     ]="LPAREN";
   TokenName[RPAREN     ]="RPAREN";
   TokenName[PLUS       ]="PLUS";
   TokenName[MINUS      ]="MINUS";
   TokenName[ERROR      ]="ERROR";
}
//------------------------------------------------------------------
//Constructor Lexer is used to redirect the input file stream from the
//keyboard to input file stream i.
//------------------------------------------------------------------
Lexer::Lexer(FILE* i)
{  yyin=i;
   PopulateRWMap();
   PopulateTokenNameMap();
}
//-----------------------End of Lex Definition-------------------------
```

File mcrpar.h

```
#ifndef mcrpar_h
#define mcrpar_h 1
//-----------------------------------------------------------------------
// File mcrpar.h defines the interface to the parser generated by
// yacc.
//-----------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   January, 2015
//-----------------------------------------------------------------------
// Copyright January, 2015 by Thomas R. Turner
// Do not reproduce without permission from Thomas R. Turner.
//-----------------------------------------------------------------------
//Application include files
//-----------------------------------------------------------------------
#include "mcrlex.h"
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
//Function yyparse is the parser generated by yacc.
//-----------------------------------------------------------------------
#ifdef __cplusplus
extern "C"
#endif
int yyparse (void);
#endif
class Parser : public Lexer {
public:
  Parser(FILE* i):Lexer(i){}
  int Parse(void) { return yyparse(); }
};
```

File mcrpar.y

```
%{
//---------------------------------------------------------------------
//File mcrpar.y contains a specification for Micro
//defined by Thomas R. Turner.
//---------------------------------------------------------------------
//Author:   Thomas R. Turner
//E-Mail:   trturner@uco.edu
//Date:     January, 2015
//---------------------------------------------------------------------
//Copyright January, 2015 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner.
//---------------------------------------------------------------------
//C++ inlcude files
//---------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
//---------------------------------------------------------------------
//---------------------------------------------------------------------
//Application inlcude files
//---------------------------------------------------------------------
#include "mcrlex.h"
#include "mcrpar.h"
//---------------------------------------------------------------------
//Functions
//---------------------------------------------------------------------
void yyerror(const char* m);
//---------------------------------------------------------------------
//Externals
//---------------------------------------------------------------------
extern char* yytext;
extern ofstream tfs;
extern int line;
extern int col;
%}
%union {
  string* token;
}


%token <token> BEGAN
%token <token> END
%token <token> READ
%token <token> WRITE
```

```
%token <token> INTLIT
%token <token> IDENTIFIER
%token <token> ASSIGN
%token <token> SEMICOLON
%token <token> COMMA
%token <token> LPAREN
%token <token> RPAREN
%token <token> PLUS
%token <token> MINUS
%token <token> REALIT
%token <token> CHRLIT
%token <token> ERROR

%%
program:
  BEGAN statement_list END
  {tfs << endl << "#001 program -> BEGIN statement-list END";
  }
statement_list:
  statement
  {tfs << endl << "#002 statement_list -> statement";
  }
statement_list:
  statement_list SEMICOLON statement
  {tfs << endl << "#003 statement_list -> statement_list ; statement";
  }
statement:
  IDENTIFIER ASSIGN expression
  {tfs << endl << "#004 statement -> IDENTIFIER(" << (*$2) << ") := expression ;";
  }
statement:
  READ LPAREN identifier_list RPAREN
  {tfs << endl << "#005 statement -> READ ( identifier-list ) ;";
  }
statement:
  WRITE LPAREN expression_list RPAREN
  {tfs << endl << "#006 statement -> WRITE ( expression-list ) ;";
  }
identifier_list:
  IDENTIFIER
  {tfs << endl << "#007 identifer-list -> IDENTIFIER ";
  }
identifier_list:
  identifier_list COMMA IDENTIFIER
  {tfs << endl << "#008 identifer-list -> identifier-list , IDENTIFIER ";
  }
expression_list:
  expression
```

```
 {tfs << endl << "#009 expression-list -> expression ";
 }
expression_list:
 expression_list COMMA expression
 {tfs << endl << "#010 expression-list -> expression-list , expression ";
 }
expression:
 primary
 {tfs << endl << "#011 expression -> primary ";
 }
expression:
 primary addop primary
 {tfs << endl << "#012 expression -> primary addop primary";
 }
primary:
 LPAREN expression RPAREN
 {tfs << endl << "#013 primary -> ( expression )";
 }
primary:
 IDENTIFIER
 {tfs << endl << "#014 primary -> IDENTIFIER";
 }
primary:
 INTLIT
 {tfs << endl << "#015 primary -> INTLIT";
 }
addop:
 PLUS
 {tfs << endl << "#016 addop -> +";
 }
addop:
 MINUS
 {tfs << endl << "#017 addop -> -";
 }
%%
//-------------------------------------------------------------------
//User function section
//-------------------------------------------------------------------
void yyerror(const char* m)
{   cout << endl
       << "line(" << line << ") col(" << col << ") " << m;
   cout << endl;
}
```

File mcrtkn_prolog.h

```
#ifndef mcrtkn_h
#define mcrtkn_h 1
//-------------------------------------------------------------------------
//File mcrtkn_prolog.h must be prefixed onto mcrtkn.h
//-------------------------------------------------------------------------
```

File y.tab.h

```
/* A Bison parser, made by GNU Bison 3.0.4.  */

/* Bison interface for Yacc-like parsers in C

   Copyright (C) 1984, 1989-1990, 2000-2015 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.  */

#ifndef YY_YY_Y_TAB_H_INCLUDED
# define YY_YY_Y_TAB_H_INCLUDED
/* Debug traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif
```

```
#if YYDEBUG
extern int yydebug;
#endif

/* Token type.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  enum yytokentype
  {
    BEGAN = 258,
    END = 259,
    READ = 260,
    WRITE = 261,
    INTLIT = 262,
    IDENTIFIER = 263,
    ASSIGN = 264,
    SEMICOLON = 265,
    COMMA = 266,
    LPAREN = 267,
    RPAREN = 268,
    PLUS = 269,
    MINUS = 270,
    REALIT = 271,
    CHRLIT = 272,
    ERROR = 273
  };
#endif
/* Tokens.  */
#define BEGAN 258
#define END 259
#define READ 260
#define WRITE 261
#define INTLIT 262
#define IDENTIFIER 263
#define ASSIGN 264
#define SEMICOLON 265
#define COMMA 266
#define LPAREN 267
#define RPAREN 268
#define PLUS 269
#define MINUS 270
#define REALIT 271
#define CHRLIT 272
#define ERROR 273

/* Value type.  */
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
```

```
union YYSTYPE
{
#line 38 "mcrpar.y" /* yacc.c:1909  */

  string* token;

#line 94 "y.tab.h" /* yacc.c:1909  */
};

typedef union YYSTYPE YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define YYSTYPE_IS_DECLARED 1
#endif


extern YYSTYPE yylval;

int yyparse (void);
```

File mcrtkn_epilog.h

```
//-----------------------------------------------------------------------
//File mcrtkn_epilog.h must be appended to file mcrtkn.h
//-----------------------------------------------------------------------
#endif
```

File t00.mcr

```
begin x:=x+2 end
```

File t00.trc

```
Tokentrace(line(   1),col(   1),code(258),name(     BEGAN),spelling("begin"))
Tokentrace(line(   1),col(   7),code(263),name(IDENTIFIER),spelling("x"))
Tokentrace(line(   1),col(   8),code(264),name(    ASSIGN),spelling(":="))
Tokentrace(line(   1),col(  10),code(263),name(IDENTIFIER),spelling("x"))
#014 primary -> IDENTIFIER
Tokentrace(line(   1),col(  11),code(269),name(      PLUS),spelling("+"))
#016 addop -> +
Tokentrace(line(   1),col(  12),code(262),name(    INTLIT),spelling("2"))
#015 primary -> INTLIT
#012 expression -> primary addop primary
#004 statement -> IDENTIFIER(x) := expression ;
#002 statement_list -> statement
Tokentrace(line(   1),col(  14),code(259),name(       END),spelling("end"))
#001 program -> BEGIN statement-list END
```

File t01.mcr

```
begin read(x);

x:=x+2; { This is a one line comment}
{This is a comment that begins in column 1}


y:=x-3; write(x,y)  {This is
                     a multi-line
                     comment
                    } end
```

File t01.trc

```
Tokentrace(line(   1),col(   1),code(258),name(     BEGAN),spelling("begin"))
Tokentrace(line(   1),col(   7),code(260),name(      READ),spelling("read"))
Tokentrace(line(   1),col(  11),code(267),name(    LPAREN),spelling("("))
Tokentrace(line(   1),col(  12),code(263),name(IDENTIFIER),spelling("x"))
#007 identifer-list -> IDENTIFIER
Tokentrace(line(   1),col(  13),code(268),name(    RPAREN),spelling(")"))
#005 statement -> READ ( identifier-list ) ;
#002 statement_list -> statement
Tokentrace(line(   1),col(  14),code(265),name( SEMICOLON),spelling(";"))
Tokentrace(line(   3),col(   1),code(263),name(IDENTIFIER),spelling("x"))
Tokentrace(line(   3),col(   2),code(264),name(    ASSIGN),spelling(":="))
Tokentrace(line(   3),col(   4),code(263),name(IDENTIFIER),spelling("x"))
#014 primary -> IDENTIFIER
Tokentrace(line(   3),col(   5),code(269),name(      PLUS),spelling("+"))
#016 addop -> +
Tokentrace(line(   3),col(   6),code(262),name(    INTLIT),spelling("2"))
#015 primary -> INTLIT
#012 expression -> primary addop primary
#004 statement -> IDENTIFIER(x) := expression ;
#003 statement_list -> statement_list ; statement
Tokentrace(line(   3),col(   7),code(265),name( SEMICOLON),spelling(";"))
Tokentrace(line(   7),col(   1),code(263),name(IDENTIFIER),spelling("y"))
Tokentrace(line(   7),col(   2),code(264),name(    ASSIGN),spelling(":="))
Tokentrace(line(   7),col(   4),code(263),name(IDENTIFIER),spelling("x"))
#014 primary -> IDENTIFIER
Tokentrace(line(   7),col(   5),code(270),name(     MINUS),spelling("-"))
#017 addop -> -
Tokentrace(line(   7),col(   6),code(262),name(    INTLIT),spelling("3"))
#015 primary -> INTLIT
#012 expression -> primary addop primary
#004 statement -> IDENTIFIER(y) := expression ;
#003 statement_list -> statement_list ; statement
Tokentrace(line(   7),col(   7),code(265),name( SEMICOLON),spelling(";"))
Tokentrace(line(   7),col(   9),code(261),name(     WRITE),spelling("write"))
Tokentrace(line(   7),col(  14),code(267),name(    LPAREN),spelling("("))
Tokentrace(line(   7),col(  15),code(263),name(IDENTIFIER),spelling("x"))
#014 primary -> IDENTIFIER
Tokentrace(line(   7),col(  16),code(266),name(     COMMA),spelling(","))
#011 expression -> primary
#009 expression-list -> expression
Tokentrace(line(   7),col(  17),code(263),name(IDENTIFIER),spelling("y"))
#014 primary -> IDENTIFIER
Tokentrace(line(   7),col(  18),code(268),name(    RPAREN),spelling(")"))
#011 expression -> primary
#010 expression-list -> expression-list , expression
#006 statement -> WRITE ( expression-list ) ;
```

```
#003 statement_list -> statement_list ; statement
Tokentrace(line(  10),col(  23),code(259),name(      END),spelling("end"))
#001 program -> BEGIN statement-list END
```