Project p01 is the lexical analyzer or scanner.

- Project p01 is realized as program **pas**.
- Project p01 is the first of 12 projects that result in the Subset Pascal Compiler.
- Each of the subsequent projects adds functions to the Subset Pascal Compiler.

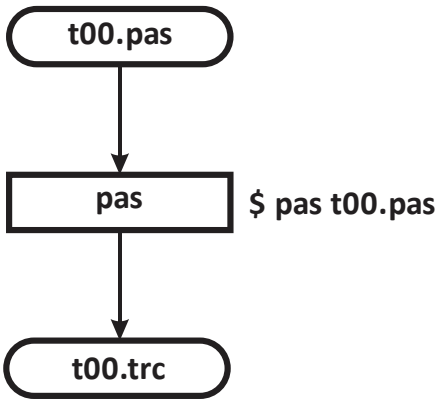

Figure 1. Input-Output-Process Model of project p01

- File **t00.pas** is the source file containing strings that are recognized by the Subset Pascal Lexical analyzer – the scanner. The strings are called tokens or lexemes.
- Each token or lexeme has two parts:
  - the ***spelling*** containing the actual string of characters that were recognized.
  - a positive integer code that uniquely identifies the token.
- File **t00.trc** is a file that contains a record of the actions of the Subset Pascal Compiler. In this case every token recognized is printed in the trace file **t00.trc**. For each token the trace file records the following information
  - the unique integer code identifying the token
  - the symbolic name of the token. For example, the reserve word **program** may be identified by the unique integer 309 but the symbol representing the code is PROGRAM.
  - the spelling of the token. For example, the source file may have the string Program but the Subset Pascal Compiler does not recognize differences in strings having the same letters and sequence but different case values. The Subset Pascal Compiler is case-insensitive. The strings “Program” and “proGram” are both recognized as the reserve word **program**. Thus, it is important to record the actual spelling including capitalization.
  - the line number of the line on which the token appeared.
  - the position of the first character on the line on which the token appeared.
  - Examples:
    Token:Code=309 Name= PROGRAM line= 5 col= 1 Spelling="program"
    Token:Code=324 Name=IDENTIFIER line= 5 col= 9 Spelling="p001_000"

Program Files:	**File**	**Description**
	pastkn.h	File **pastkn.h** contains the list of positive integer codes that uniquely identify each token. #define macro directives are used to define each token. For example, #define PROGRAM 309. Alternatively, you may construct this list using an enumerated type. However, you must ensure that every token has a ***positive*** integer code.
	paslex.h	File **paslex.h** contains the interface to the lexer and supporting functions defined in file **paslex.l**.
	paslex.l	File **paslex.l** contains specifications acceptable to the *Unix* utility ***lex*** for Subset Pascal tokens.
	pas.cpp	File **pas.cpp** contains function main and processes command line arguments.
	makepas	File **makepas** is a script file of Unix commands to remove files **paslex.cpp**, all object files, and the executable file **pas**.
	makepascal	File **makepascal** contains instructions to create program **pas**. Instructions are written for the *Unix* utility ***make***.

File t01.mcr
begin read(*x*); *x*:=*x*+2; write(*x*) end

File t01.trc

```
Token:Code=267 Name=     BEGIN line=  1 col=  1 Spelling="begin"
Token:Code=269 Name=      READ line=  1 col=  7 Spelling="read"
Token:Code=263 Name=    LPAREN line=  1 col= 11 Spelling="("
Token:Code=273 Name=IDENTIFIER line=  1 col= 12 Spelling="x"
Token:Code=264 Name=    RPAREN line=  1 col= 13 Spelling=")"
Token:Code=262 Name= SEMICOLON line=  1 col= 14 Spelling=";"
Token:Code=273 Name=IDENTIFIER line=  1 col= 16 Spelling="x"
Token:Code=265 Name=    ASSIGN line=  1 col= 17 Spelling=":="
Token:Code=273 Name=IDENTIFIER line=  1 col= 19 Spelling="x"
Token:Code=259 Name=      PLUS line=  1 col= 20 Spelling="+"
Token:Code=272 Name=    INTLIT line=  1 col= 21 Spelling="2"
Token:Code=262 Name= SEMICOLON line=  1 col= 22 Spelling=";"
Token:Code=270 Name=     WRITE line=  1 col= 24 Spelling="write"
Token:Code=263 Name=    LPAREN line=  1 col= 29 Spelling="("
Token:Code=273 Name=IDENTIFIER line=  1 col= 30 Spelling="x"
Token:Code=264 Name=    RPAREN line=  1 col= 31 Spelling=")"
Token:Code=268 Name=       END line=  1 col= 33 Spelling="end"
```

File makemcr

```
rm mcrlex.cpp
rm *.o
rm mcr
make -f makemicro
```

File makemicro

```
#---------------------------------------------------------------------
# File makemcr creates a micro language compiler
#---------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@uco.edu
# Date:   January, 2012
#---------------------------------------------------------------------
# Copyright January, 2012 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#---------------------------------------------------------------------
#---------------------------------------------------------------------
# Object files
#---------------------------------------------------------------------
obj =    mcrlex.o \
         mcr.o
#---------------------------------------------------------------------
# Bind the subset Pascal Scanneer
#---------------------------------------------------------------------
mcr:          ${obj}
              g++ -o mcr ${obj} -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
```

File mcr.cpp

```
//-----------------------------------------------------------------
//File mcr.cpp contains functions that process command line arguments
//and interface with the lex-generated scanner
//-----------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: trturner@uco.edu
//Date:   January, 2012
//-----------------------------------------------------------------
//Copyright January, 2012 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"
//-----------------------------------------------------------------
//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 arguments on the command line.";
        cout << endl;
        cout << m << " argument(s) are permitted on the command line.";
        cout << endl;
        cout << a << " argument(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)
{    Lexer L(i);
     int tc=0;
     do tc=L.Lex(); while (tc);
     //Parser P(i);
     //P.Parse();
}
//-------------------------------------------------------------------
//Function main processes command line arguments
//-------------------------------------------------------------------
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, 2012
//-----------------------------------------------------------------
// Copyright January, 2012 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 mcrner.  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 mcrtkn.h

```
#ifndef mcrtkn_h
#define mcrtkn_h 1
#define TOKEN_BEGIN 258
#define PLUS 259
#define MINUS 260
#define COMMA 261
#define SEMICOLON 262
#define LPAREN 263
#define RPAREN 264
#define ASSIGN 265
#define RESERVE_WORDS 266
#define BEGIN_ 267
#define END 268
#define READ 269
#define WRITE 270
#define REGULAR_EXPRESSIONS 271
#define INTLIT 272
#define IDENTIFIER 273
#define TOKEN_END 274
#endif
```

File mcrlex.l

```
%{
//-------------------------------------------------------------------
// File mcrlex.l defines a scanner for micro, a language defined
// in "Crafting a Compiler" by Fischer and LeBlanc.
//-------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner@uco.edu
// Date:   January, 2012
//-------------------------------------------------------------------
//Copyright January, 2012 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//-------------------------------------------------------------------
//-------------------------------------------------------------------
// Standard C and C++ Library Include Files
//-------------------------------------------------------------------
#include <cstdlib>
#include <cstring>
#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;
//-------------------------------------------------------------------
//Global Variables
//-------------------------------------------------------------------
static map<string,int> RW;
static int tokencode;
static string* TokenName;
int line=1;
int col =1;
```

```
//-----------------------------------------------------------------
//Functions
//-----------------------------------------------------------------
void ToLower(char* o,char* i,int l);              //Coerce string i to lower case
int TokenMgr(int t);                              //Token post processing
void PrintToken(ostream& o,int tc,int l,int c);   //Print the token and attributes
//-----------------------------------------------------------------
//Exceptions
//-----------------------------------------------------------------
struct StringTokenException{
    StringTokenException(char* t,int l,int c)
    {   cout << endl;
        cout << "line(" << l << ") col (" << c << ")" ;
        cout << "Lexical error: ";
        cout << "Strings cannot span lines";
        cout << endl;
        cout << "|" << t << "|";
        cout << endl;
    }
};
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;
    }
};
%}
%%
[ \t]+                  {col+=strlen(yytext);}
[\n]                    {line++;col=1;}
[a-zA-Z][a-zA-Z0-9]*    return TokenMgr(IDENTIFIER);
[0-9]+                  return TokenMgr(INTLIT);
"+"                     return TokenMgr(PLUS);
"-"                     return TokenMgr(MINUS);
","                     return TokenMgr(COMMA);
";"                     return TokenMgr(SEMICOLON);
"("                     return TokenMgr(LPAREN);
")"                     return TokenMgr(RPAREN);
":="                    return TokenMgr(ASSIGN);
.                       {throw BadCharacterException
                            (*yytext
                            ,line
                            ,col
                            );
                        }
```

```
%%
//---------------------------------------------------------------------
//Class Lexer implementation
//---------------------------------------------------------------------
//---------------------------------------------------------------------
void ToLower(char* o,char* i,int l)
{   for (int a=0;a<l&&a<1024;a++) o[a]=tolower(i[a]);  //To lower case
    o[l]=0;                                           //Null termination
}
//---------------------------------------------------------------------
//Function TokenMgr processes the token after it has been recognized
//---------------------------------------------------------------------
int TokenMgr(int t)
{   int tc=t;
    if (t==IDENTIFIER) {
        char s[1024];
        ToLower(s,yytext,strlen(yytext));
        tc=RW[s];
        if (tc==0) tc=t;
    }
    PrintToken(tfs,tc,line,col);
    col+=yyleng;
    return tc;
}
//---------------------------------------------------------------------
//Constructor Lexer is used to redirect the input file stream from the
//keyboard to input file stream i.
//---------------------------------------------------------------------
Lexer::Lexer(FILE* i)
{   yyin=i;
    const int MAXSY=17;
    static string sy[]=                 //17 Symbol Names
        {"TOKEN_BEGIN"
        ,"PLUS" ,"MINUS" ,"COMMA","SEMICOLON","LPAREN","RPAREN","ASSIGN"
        ,"RESERVE_WORDS"
        ,"BEGIN" ,"END" ,"READ" ,"WRITE"
        ,"REGULAR_EXPRESSIONS"
        ,"INTLIT","IDENTIFIER"
        , "TOKEN_END"
        };
    //---------------------------------------------------------------
    //Populate array TokenName with the names of the tokens given in array sy.
    //---------------------------------------------------------------
    TokenName=new string[MAXSY];
    for (int a=0;a<MAXSY;a++) TokenName[a]=sy[a];
```

```
    //-----------------------------------------------------------------
    //Populate the map RW with the spellings and unique integer
    //token codes for the reserve words.
    //-----------------------------------------------------------------
    static string rw[]={"begin","end","read","write" };
    static int tc[]={BEGIN_,END,READ,WRITE};
    for (int a=0;a<4;a++) RW[rw[a]]=tc[a];
}
//-----------------------------------------------------------------
//Function Lex calls yylex
//-----------------------------------------------------------------
int Lexer::Lex(void)
{   tokencode=yylex();
    return tokencode;
}
//-----------------------------------------------------------------
//Function PrintToken prints the token code name and the spelling of the
//token.
//-----------------------------------------------------------------
void PrintToken(ostream& o,int tc,int l,int c)
{   o << endl;
    o << "Token";
    o << ":Code="    << setw( 3) << tc;
    o << " Name="    << setw(10) << TokenName[tc-TOKEN_BEGIN];
    o << " line="    << setw( 3) << l;
    o << " col="     << setw( 3) << c;
    o << " Spelling=\"" << (char*)yytext << "\"";
}

//----------------------End of Lex Definition----------------------
```