Project:	Employ the U*nix* utility ***lex*** to create a lexical analyzer for the Subset Pascal programming language. A block diagram of the entire Subset Pascal Compiler, P-Code Assembler, and P-Code Interpreter is given in Figure 1.	
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**.
	pas.cpp	File **pas.cpp** contains function main and processes command line arguments.
	makepascal	File **makepascal** contains instructions for program **paslex**. Instructions are written for the *Unix* utility ***make***. Program **paslex** is contained in file **paslex**.
	makepas	File **makepas** is a Unix script file that removes old file created in the last creation of executable file **pas** and invokes file **makepascal** to create a new executable file **pas**. File makepas is given below. rm *.o rm paslex.cpp rm pas make -f makepascal
Command Line:	Project **1** can be invoked with zero or one program parameters. The first program parameter is the input file name. Sample command lines together with corresponding actions by program **pas** are shown below. Boldfaced type indicates data entered at the keyboard by the user. $ **pas** Enter the input file name: **p00.pas** $ **pas p00.pas**	
Input File:	The input file contains a Subset Pascal program. The input file name must have the suffix "**.pas**" Please refer to Figure 2 for an example of the format of an input file.	

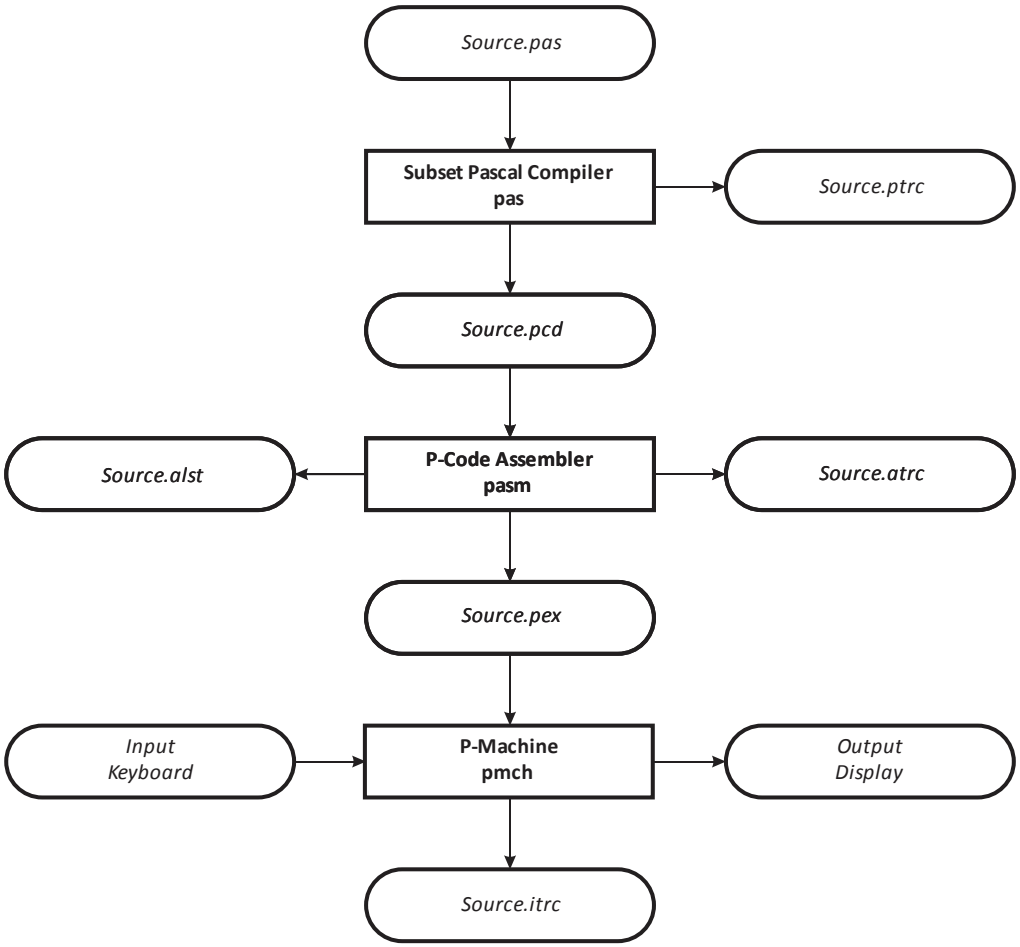

Figure 1. Subset Pascal Compiler, P-Code Assembler, and P-Code Interpreter.

Output File:	The name of the output file has the same prefix as the input file and the four-character suffix, "**.pas**" is replaced by the four-character suffix "**.trc**" For example, if the name of the input file was **p00.pas**, the name of the output file is **p00.trc**. Each line of the output file contains information about a single token. A token is a pair consisting of a unique integer identifying the pattern recognized by the lexical analyzer and a string, called the spelling, that is a specific instance of the pattern. For the purpose of this project, five (5) items are required for each token. They are: 1. **TokenCode**: The TokenCode is the first part of the quintuple that is a token. It is the unique integer identifying the pattern recognized by the lexical analyzer. 2. **TokenName**: The TokenName is a symbolic name for the TokenCode. For example, the range-token, consisting of two periods in sequence, might be assigned the unique integer code 300. The name for the unique integer code is **RANGE**. The names of the tokens are given in Table 1. The integer codes assigned to the tokens will be assigned by the parser generator, yacc, so their values are arbitrary with the following exception. All integer codes assigned to tokens must be positive counting numbers. 3. **Line**: The line is the line number on which the token appears. 4. **Column**: The column is the column number on which the first character of the token appears. 5. **TokenSpelling**: The TokenSpelling is the actual string recognized by the lexical analyzer. The TokenSpelling is the specific instance of a general pattern recognized by the lexical analyzer. For example, the string of characters "**247**" is recognized by the lexical analyzer as an integer literal abbreviated to **INTLIT**. Please refer to Figure 3 for an example of the output file format.

```
program p00;
  var a:integer;
begin
  a:=1;
  a:=a+1
end.
```

Figure 2. Example input file **p00.pas.**

```
Token:Code=293 Name=   PROGRAM line=  1 col=  1 Spelling="program"
Token:Code=300 Name=        ID line=  1 col=  9 Spelling="p00"
Token:Code=265 Name= SEMICOLON line=  1 col= 12 Spelling=";"
Token:Code=297 Name=       VAR line=  2 col=  3 Spelling="var"
Token:Code=300 Name=        ID line=  2 col=  7 Spelling="a"
Token:Code=266 Name=     COLON line=  2 col=  8 Spelling=":"
Token:Code=300 Name=        ID line=  2 col=  9 Spelling="integer"
Token:Code=265 Name= SEMICOLON line=  2 col= 16 Spelling=";"
Token:Code=281 Name=     BEGIN line=  3 col=  1 Spelling="begin"
Token:Code=300 Name=        ID line=  4 col=  3 Spelling="a"
Token:Code=262 Name=    ASSIGN line=  4 col=  4 Spelling=":="
Token:Code=301 Name=    INTLIT line=  4 col=  6 Spelling="1"
Token:Code=265 Name= SEMICOLON line=  4 col=  7 Spelling=";"
Token:Code=300 Name=        ID line=  5 col=  3 Spelling="a"
Token:Code=262 Name=    ASSIGN line=  5 col=  4 Spelling=":="
Token:Code=300 Name=        ID line=  5 col=  6 Spelling="a"
Token:Code=258 Name=      PLUS line=  5 col=  7 Spelling="+"
Token:Code=301 Name=    INTLIT line=  5 col=  8 Spelling="1"
Token:Code=285 Name=       END line=  6 col=  1 Spelling="end"
Token:Code=263 Name=    PERIOD line=  6 col=  4 Spelling="."
```

Figure 3. Example output file **p00.trc.**

Token			Token		
TokenCode	**TokenName**	**Pattern**	**TokenCode**	**TokenName**	**Pattern**
	PLUS	**+**		AND	and
	MINUS	**-**		ARRAY	array
	STAR	*****		BEGIN	begin
	SLASH	**/**		DIV	div
	ASSIGN	**:=**		DO	do
	PERIOD	**.**		ELSE	else
	COMMA	**,**		END	end
	SEMICOLON	**;**		FUNCTION	function
	COLON	**:**		IF	if
	EQU	**=**		MOD	mod
	NEQ	**<>**		NOT	not
	LES	**<**		OF	of
	LEQ	**<=**		OR	or
	GRT	**>**		PROCEDURE	procedure
	GEQ	**>=**		PROGRAM	program
	LPAREN	**(**		THEN	then
	RPAREN	**)**		TO	to
	LBRACKET	**[**		TYPE	type
	RBRACKET	**]**		VAR	var
	RANGE	**..**		WHILE	while

Table 1. Subset Pascal Token Specifications

TokenCode	TokenName	Pattern
	ID	An identifier must have at least one character. Only the first 30 characters will be used to distinguish one identifier from another. The length of identifiers is undefined. All identifiers and reserve words are case-insensitive. An identifier can begin with a letter or the underscore character. Subsequent characters can be letters, digits, or the underscore character.
	INTLIT	An integer literal is one or more digits.
	REALIT	A real literal – a real number constant – consists of one or more integer digits, a decimal point, one or more fractional digits, and an optional exponent. The exponent, if present, is the letter e, an optional sign, and one or more digits. Remember, Pascal is case-insensitive so the letter e may be capitalized.
	REALIT	A real literal – a real number constant – consists of one or more integer digits and an exponent.
	CHRLIT	*any single character enclosed between two apostrophes or, to represent an apostrophe, a doubled apostrophe enclosed between two apostrophes*
	COMMENT	*Comments begin with an opening curly brace,* **{***, and end with a closing curly brace,* **}***. Any number, including zero, of characters can appear in a comment. Comments can include multiple lines. No tokens are produced when a comment is recognized and no action is taken. Comments are ignored.*

Table 1. Subset Pascal Token Specifications (continued)