<table>
<tr><td>Project:</td><td colspan="2">Employ the UNIX utility lex to write a scanner for Subset Pascal whose grammar is given in Table 1. Tokens are defined in Table 2 and Figures 1, 2, 3, and 4. An example program is given in Figure 5 and the corresponding output produced by the Subset Pascal scanner is shown in Figure 6. You are not required to produce an exact replica of the output shown in Figure 6, but you are required to print one line for every token recognized. For each token, you must print its tokencode, its location, the string of characters recognized, and its symbolic name.

On the last page of this document, you can find a sample score sheet used to grade this project.</td></tr>
<tr><td>Project Files:</td><td colspan="2">Project 1 consists of source files that are identified below. You must name your source files exactly as shown below.</td></tr>
<tr><td>Project Location:</td><td colspan="2">Project files must be stored in the root directory of your student account. Failure to store project files in the root directory of your student account will result in a score of zero (0) for this project.</td></tr>
<tr><td></td><td>File</td><td>Description</td></tr>
<tr><td></td><td>p01.cpp</td><td>File p01.cpp contains functions which process command line arguments and invoke the scanner, repeatedly to find all the tokens in the input source file.</td></tr>
<tr><td></td><td>p01lex.h</td><td>File p01lex.h defines the interface to the Subset Pascal scanner.</td></tr>
<tr><td></td><td>p01tkn.h</td><td>File p01tkn.h contains the list of preprocessor definitions that define token codes. For example,
#define PLUS 1
#define MINUS 2
…</td></tr>
<tr><td></td><td>p01lex.l</td><td>File p01lex.l contains functions that accept the tokens of a Subset Pascal program.</td></tr>
<tr><td></td><td>p01make</td><td>File p01make contains instructions for program p01. Instructions are written for the UNIX utility make. File p01make creates the executable file p01 from source files identified above.</td></tr>
<tr><td>Command Line:</td><td colspan="2">Project 1 can be invoked with zero or one program parameters. The first program parameter is the source file to be analyzed. Sample command lines together with corresponding actions by program p01 are shown below. Boldfaced type indicates data entered at the keyboard by the user.
$ p01
Enter the source file name: gcd.pas

$ p01 gcd.pas</td></tr>
<tr><td>Input File Name:</td><td colspan="2">The input file name must have the suffix .pas.</td></tr>
<tr><td>Input File:</td><td colspan="2">The input file can contain any sequence of characters. The scanner recognizes valid Subset Pascal Tokens and rejects invalid tokens. Sample test files are stored in the class directory. You can obtain copies of these files by issuing the following command when signed on to your student account on the department computer.
$ cp ~tt/cs4023/*.pas .</td></tr>
</table>

Output File Name:	The output file name has the same prefix as the input file but substitutes the suffix **.trc** for input file name suffix **.pas**. For example, if the input file name is **gcd.pas**, the output file name is **gcd.trc**.
Output File:	The output file should contain a list of the Subset Pascal tokens. Tokens and associated information is presented as shown in Figure 6.

	Rule		
No.	**LHS**		**RHS**
1	*program*	→	*program-head program-declarations program-body*
2	*program-head*		**program id** *program-parameters* **;**
3	*program-declarations*		*declarations subprogram-declarations*
4	*program-body*		*compound-statement* **.**
5	*program-parameters*		∈
6	*program-parameters*		**(** *program-parameter-list* **)**
7	*program-parameter-list*		*identifier-list*
8	*identifier-list*	→	**id**
9	*identifier-list*	→	*identifier-list* **, id**
10	*declarations*	→	∈
11	*declarations*	→	*declarations* **var** *identifier-list* **:** *type* ***;***
12	*type*	→	*standard-type*
13	*type*	→	**array [intlit .. intlit] of** *standard-type*
14	*standard-type*	→	**id**
15	*subprogram-declarations*	→	∈
16	*subprogram-declarations*	→	*subprogram-declarations subprogram-declaration* **;**
17	*subprogram-declaration*	→	*subprogram-head declarations compound-statement*
18	*subprogram-head*	→	**function id** *subprogram-parameters* **:** *standard-type* **;**
19	*subprogram-head*	→	**procedure id** *subprogram-parameters***;**
20	*subprogram-parameters*	→	∈
21	*subprogram-parameters*	→	**(** *parameter-list* **)**
22	*parameter-list*	→	*identifier-list* **:** *type*
23	*parameter-list*	→	*parameter-list* **;** *identifier-list* **:** *type*
24	*compound-statement*	→	**begin** *optional-statements* **end**
25	*optional-statements*	→	∈
26	*optional-statements*	→	*statement-list*
27	*statement-list*	→	*statement*
28	*statement-list*	→	*statement-list* **;** *statement*
29	*statement*	→	*variable* **:=** *expression*
30	*statement*	→	*procedure-statement*
31	*statement*	→	*alternative-statement*
32	*statement*	→	*iterative-statement*
33	*statement*	→	*compound-statement*

Table 1. Subset Pascal Grammar.

	Rule		
Id	**LHS**		**RHS**
34	*alternative-statement*	→	*if-statement*
35	iterative-statement	→	*while-statement*
36	*iterative-statement*	→	*repeat-statement*
37	*iterative-statement*	→	*for-statement*
38	*if-statement*	→	**if** *expression* **then** *statement* **else** *statement*
39	*while-statement*	→	**while** *expression* **do** *statement*
40	*repeat-statement*	→	**repeat** *statement_list* **until** *expression*
41	*for-statement*	→	**for** *variable* **:=** *expression* **to** *expression* **do** *statement*
42	*for-statement*	→	**for** *variable* **:=** *expression* **downto** *expression* **do** *statement*
43	*variable*	→	**id**
44	*variable*	→	**id [** *expression* **]**
45	*procedure-statement*	→	**id**
46	*procedure-statement*	→	**id (** *expression-list* **)**
47	*expression-list*	→	*expression*
48	*expression-list*	→	*expression-list* **,** *expression*
49	*expression*	→	*simple-expression*
50	*expression*	→	*simple-expression relop simple-expression*
51	*relop*	→	**=**
52	*relop*	→	**<>**
53	*relop*	→	**<**
54	*relop*	→	**<=**
55	*relop*	→	**>**
56	*relop*	→	**>=**
57	*simple-expression*	→	*term*
58	*simple-expression*	→	*sign term*
59	*sign*	→	**+**
60	*sign*	→	**-**
61	*simple-expression*	→	*simple-expression addop term*
62	*addop*	→	**+**
63	*addop*	→	**-**
64	*addop*	→	**or**
65	*term*	→	*factor*
66	*term*	→	*term mulop factor*
67	*mulop*	→	*****
68	*mulop*	→	**/**
69	*mulop*	→	**div**
70	*mulop*	→	**mod**
71	*mulop*	→	**and**

Table 1. Subset Pascal Grammar (continued).

	Rule

Id	LHS		RHS
72	*factor*	→	**id**
73	*factor*	→	**id [** *expression-list* **]**
74	*factor*	→	**id (** *expression-list* **)**
75	*factor*	→	**(** *expression* **)**
76	*factor*	→	**not** *factor*
77	*factor*	→	**intlit**
78	*factor*	→	**realit**
79	*factor*	→	**chrlit**

Table 1. Subset Pascal Grammar (continued).

Comments	Comments are enclosed between **{** and **}**. They may not contain a **{**. Comments may appear after any token.		
Blanks	Blanks between tokens are optional, with the exception that reserve words and identifiers must be surrounded by blanks, newline characters, the beginning of the program, or punctuation.		
Reserve words and identifiers	Reserve words and identifiers are case-insensitive. For example, Subset Pascal recognizes **begin** and **BeGin** as the same token. Capitalization of letters in reserve words and identifiers is ignored.		
Tokencodes	Assign a unique positive integer to each tokencode.		
Token	**Specification**	**Token**	**Specification**
AND	**and**	EQU	**=**
ARRAY	**array**	NEQ	**<>**
BEGAN	**begin**	LES	**<**
DIV	**div**	LEQ	**<=**
DO	**do**	GRT	**>**
DOWNTO	**downto**	GEQ	**>=**
ELSE	**else**	PLUS	**+**
END	**end**	MINUS	**-**
FOR	**for**	STAR	*****
FUNCTION	**function**	SLASH	**/**
IF	**if**	ASSIGN	**:=**
MOD	**mod**	LPAREN	**(**
NOT	**not**	RPAREN	**)**
OF	**of**	LBRACKET	**[**
OR	**or**	RBRACKET	**]**
PROCEDURE	**procedure**	COLON	**:**
PROGRAM	**program**	SEMICOLON	**;**
REPEAT	**repeat**	COMMA	**,**
THEN	**then**	PERIOD	**.**
TO	**to**	RANGE	**..**
UNTIL	**until**		
VAR	**var**		
WHILE	**while**		

Table 2. Subset Pascal Token Definitions

id (identifier)

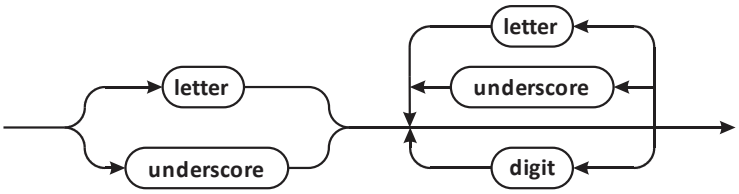

Figure 1. **id** (identifier)

Examples

- ice_cream
- _myclass
- Principal

intlit (integer literal)

Figure 2. **intlit** (integer literal)

Examples:

- 3
- 03
- 6272844

chrlit (character literal)

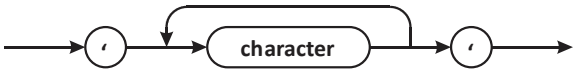

Figure 3. **chrlit** (character literal)

Note: A single apostrophe is represented by two apostrophes is succession.
Example character literals:

- 'a'
- ';'
- '3'
- 'begin'
- 'don''t'
- ''''

realit (real literal)

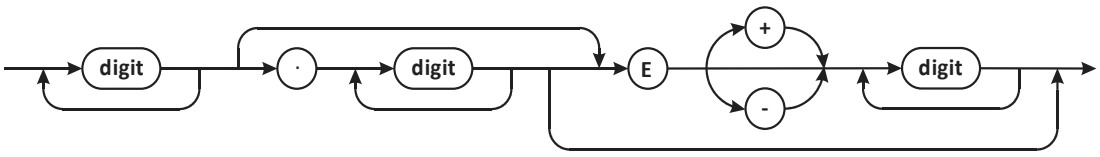

Figure 4. **realit** (real literal)

Examples:

- 0.6
- 5E-8
- 49.22E+08
- 1E10

```
program example(input,output);
    var x,y: integer;
    function gcd(a,b:integer):integer;
    begin{gcd}
        if b=0 then gcd:=a
        else gcd:=gcd(b,a mod b)
    end{gcd};
begin{example}
    readln(x,y);
    write(gcd(x,y))
end{example}.
```

Figure 5. File **gcd.pas**

```
token "program"    pos( 1, 1) code(17) symbol:PROGRAM
token "example"    pos( 1, 9) code(44) symbol:ID
token "("          pos( 1,16) code(35) symbol:LPAREN
token "input"      pos( 1,16) code(44) symbol:ID
token ","          pos( 1,21) code(41) symbol:COMMA
token "output"     pos( 1,21) code(44) symbol:ID
token ")"          pos( 1,27) code(36) symbol:RPAREN
token ";"          pos( 1,27) code(40) symbol:SEMICOLON
token "var"        pos( 2, 3) code(22) symbol:VAR
token "x"          pos( 2, 7) code(44) symbol:ID
token ","          pos( 2, 8) code(41) symbol:COMMA
token "y"          pos( 2, 8) code(44) symbol:ID
token ","          pos( 2, 9) code(41) symbol:COMMA
token "z"          pos( 2, 9) code(44) symbol:ID
token ":"          pos( 2,10) code(39) symbol:COLON
token "integer"    pos( 2,10) code(44) symbol:ID
token ";"          pos( 2,17) code(40) symbol:SEMICOLON
token "function"   pos( 3, 3) code(10) symbol:FUNCTION
token "gcd"        pos( 3,12) code(44) symbol:ID
token "("          pos( 3,15) code(35) symbol:LPAREN
token "a"          pos( 3,15) code(44) symbol:ID
token ","          pos( 3,16) code(41) symbol:COMMA
token "b"          pos( 3,16) code(44) symbol:ID
token ":"          pos( 3,17) code(39) symbol:COLON
token "integer"    pos( 3,17) code(44) symbol:ID
token ")"          pos( 3,24) code(36) symbol:RPAREN
token ":"          pos( 3,24) code(39) symbol:COLON
token "integer"    pos( 3,24) code(44) symbol:ID
token ";"          pos( 3,31) code(40) symbol:SEMICOLON
token "begin"      pos( 4, 3) code( 3) symbol:BEGIN
token "if"         pos( 5, 5) code(11) symbol:IF
token "b"          pos( 5, 8) code(44) symbol:ID
token "="          pos( 5, 9) code(24) symbol:EQU
token "0"          pos( 5, 9) code(45) symbol:INTLIT
token "then"       pos( 5,10) code(19) symbol:THEN
token "gcd"        pos( 5,15) code(44) symbol:ID
token ":="         pos( 5,18) code(34) symbol:ASSIGN
```

Figure 6. Subset Pascal Scanner output

```
token "a"          pos( 5,18) code(44) symbol:ID
token "else"       pos( 6, 5) code( 7) symbol:ELSE
token "gcd"        pos( 6,10) code(44) symbol:ID
token ":="         pos( 6,13) code(34) symbol:ASSIGN
token "gcd"        pos( 6,13) code(44) symbol:ID
token "("          pos( 6,16) code(35) symbol:LPAREN
token "b"          pos( 6,16) code(44) symbol:ID
token ","          pos( 6,17) code(41) symbol:COMMA
token "a"          pos( 6,17) code(44) symbol:ID
token "mod"        pos( 6,19) code(12) symbol:MOD
token "b"          pos( 6,23) code(44) symbol:ID
token ")"          pos( 6,24) code(36) symbol:RPAREN
token "end"        pos( 7, 3) code( 8) symbol:END
token ";"          pos( 7,11) code(40) symbol:SEMICOLON
token "begin"      pos( 8, 1) code( 3) symbol:BEGIN
token "readln"     pos( 9, 3) code(44) symbol:ID
token "("          pos( 9, 9) code(35) symbol:LPAREN
token "x"          pos( 9, 9) code(44) symbol:ID
token ","          pos( 9,10) code(41) symbol:COMMA
token "y"          pos( 9,10) code(44) symbol:ID
token ")"          pos( 9,11) code(36) symbol:RPAREN
token ";"          pos( 9,11) code(40) symbol:SEMICOLON
token "write"      pos(10, 3) code(44) symbol:ID
token "("          pos(10, 8) code(35) symbol:LPAREN
token "gcd"        pos(10, 8) code(44) symbol:ID
token "("          pos(10,11) code(35) symbol:LPAREN
token "x"          pos(10,11) code(44) symbol:ID
token ","          pos(10,12) code(41) symbol:COMMA
token "y"          pos(10,12) code(44) symbol:ID
token ")"          pos(10,13) code(36) symbol:RPAREN
token ")"          pos(10,13) code(36) symbol:RPAREN
token "end"        pos(11, 1) code( 8) symbol:END
token "."          pos(11,13) code(42) symbol:PERIOD
```

Figure 6. Subset Pascal Scanner output (continued)

Scoring Block			
Component	**Available**	**Earned**	**Explanation**
Compilation			A zero (0) is recorded for the entire project if the project fails to compile without errors or warnings.
Submission Instructions	10	10	• A zero (0) is recorded for this component if the project is stored in a folder other than the root directory of the project account. • A zero (0) is recorded for this component if the project makefile fails to function correctly. • A zero (0) is recorded for this component if any file name differs from specifications. • A zero (0) is recorded if source files are not recorded in separately titled sections of this document. • A zero (0) is recorded for this component if the author identification block is copied or completed incorrectly. • A zero (0) is recorded for the component if the Scoring block is copied incorrectly.
Author Identification	5	5	A zero (0) is recorded for this component if any source file including the makefile does not have a complete author identification block for both team members.
Command Line	5	5	A zero (0) is recorded for this component if command line arguments are not processed according to project specifications.
Output file	5	5	A zero (0) is recorded for this component if the output file is not created or not named according to project specifications.
Execution	25	25	• Five (5) points are subtracted from this component if comments are not properly managed. • Up to five (5) points are subtracted from this component if all legal forms of character literals are not recognized and all illegal forms rejected. • Up to five (5) points are subtracted from this component if all legal forms of real literals are not recognized and all illegal forms rejected. • Up to five (5) points are subtracted from this component if all reserve words are not recognized. • Up to five (5) points are subtracted from this component if all punctuation is not recognized.
Total	**50**	**50**	