Project: Generate P-Code instructions for *simple_expressions*. Implement semantics for the shaded rules in the table below. Implementation notes are written immediately below the rule. Write your p-code instructions to a file having the same prefix as the file containing the Subset Pascal Program but having the suffix "**.pcd**"

Compile test programs in Table 2 and inspect the P-Code produced to validate that your Subset Pascal Compiler functions properly.

PCode Generation:

File	Description
PCode.h	Defines class PCode for creating p-code instructions
PCode.cpp	Implements class PCode
Node.h	Defines class Node for creating expression trees
Node.cpp	Implements class Node
Exp.h	Defines class Exp for printing expression trees
Exp.cpp	Implements class Exp

No.	LHS		Rule RHS
	program	→	*program_head program_declarations program_body*
	program_head	→	**program id** *program_parameters* **;**
	program_declarations	→	*variable_declarations subprogram_definitions*
	program_body	→	*compound_statement* **.**
	program_parameters	→	∈
	program_parameters	→	**(** *program_parameter_list* **)**
	program_parameter_list	→	*identifier_list*
	identifier_list	→	**id**
	identifier_list	→	*identifier_list* **, id**
	variable_declarations	→	∈
	variable_declarations	→	*variable_declarations* **var** *identifier_list* **:** *type* **;**
	type	→	*standard_type*
	type	→	**array [intlit .. intlit] of** *standard_type*
	standard_type	→	**id**
	subprogram_definitions	→	∈
	subprogram_definitions	→	*subprogram_definitions subprogram_definition* **;**
	subprogram_definition	→	*subprogram_head variable_declarations subprogram_body*
	subprogram_head	→	*function_prolog subprogram_parameters* **:** *standard_type* **;**
	subprogram_head	→	*procedure_prolog subprogram_parameters* **;**
	function_prolog	→	**function id**
	procedure_prolog	→	**procedure id**
	subprogram_body	→	*compound_statement*
	subprogram_parameters	→	∈
	subprogram_parameters	→	**(** *parameter_list* **)**

Table 1. Subset Pascal Grammar

No.	LHS		Rule RHS
	parameter_list	→	*identifier_list* **:** *type*
	parameter_list	→	*parameter_list* **;** *identifier_list* **:** *type*
	compound_statement	→	**begin** *optional_statements* **end**
	optional_statements	→	∈
	optional_statements	→	*statement_list*
	statement_list	→	*statement*
	statement_list	→	*statement_list* **;** *statement*
	statement	→	*variable* **:=** *expression*
	statement	→	*procedure_statement*
28	*statement*	→	*compound_statement*
29	*statement*	→	**if** *expression* **then** *statement* **else** *statement*
30	*statement*	→	**while** *expression* **do** *statement*
31	*variable*	→	**id**
32	*variable*	→	**id [** *expression* **]**
33	*procedure_statement*	→	**id**
34	*procedure_statement*	→	**id (** *expression_list* **)**
35	*expression_list*	→	*expression*
36	*expression_list*	→	*expression_list* **,** *expression*
37	*expression*	→	*simple_expression*
38	*expression*	→	*simple_expression* = *simple_expression*
39	*expression*	→	*simple_expression* <> *simple_expression*
40	*expression*	→	*simple_expression* < *simple_expression*
41	*expression*	→	*simple_expression* <= *simple_expression*
42	*expression*	→	*simple_expression* > *simple_expression*
43	*expression*	→	*simple_expression* >= *simple_expression*
44	*simple_expression*	→	*term*
45	*simple_expression*	→	*+ term*
46	*simple_expression*	→	*- term*
47	*simple_expression*	→	*simple_expression + term*
48	*simple_expression*	→	*simple_expression - term*
49	*simple_expression*	→	*simple_expression* **or** *term*
50	*term*	→	*factor*
51	*term*	→	*term* ***** *factor*
52	*term*	→	*term* / *factor*
53	*term*	→	*term* **div** *factor*
54	*term*	→	*term* **mod** *factor*
55	*term*	→	*term* **and** *factor*

Table 1. Subset Pascal Grammar (continued)

		Rule	
No.	**LHS**		**RHS**
56	*factor*	→	**id**
57	*factor*	→	**id [** *expression* **]**
58	*factor*	→	**id (** *expression_list* **)**
59	*factor*	→	**(** *expression* **)**
60	*factor*	→	**not** *factor*
61	*factor*	→	**intlit**
62	*factor*	→	**realit**
63	*factor*	→	**chrlit**

Table 1. Subset Pascal Grammar (continued)

Test Program	**P-Code**
44 *simple_expression* → *term*	

`program se00;`
`  var r:real;`
`begin{se00}`
`  r:=9.8`
`end{se00}.`
	`ldc  r  9.8`

45 *simple_expression* → *+ term*	

`program se01;`
`  var r:real;`
`begin{se01}`
`  r:=+9.8`
`end{se01}.`
	`ldc  r  9.8`

45 *simple_expression* → *+ term*	

`program se02;`
`  var i:integer;`
`begin{se02}`
`  i:=+4`
`end{se02}.`
	`ldc  i  4`

46 *simple_expression* → *- term*	

`program se03;`
`  var r:real;`
`begin{se03}`
`  r:=-9.8`
`end{se03}.`
	`ldc  r  9.8`
	`ngr`

46 *simple_expression* → *- term*	

`program se04;`
`  var i:integer;`
`begin{se04}`
`  i:=-4`
`end{se04}.`
	`ldc  i  4`
	`ngi`

47 *simple_expression* → *simple_expression + term*	

`program se05;`
`  var se:integer; var l:integer;`
`  var r:integer;`
`begin{se05}`
`  se:=l+r`
`end{se05}.`
	`lvi  0  6`
	`lvi  0  7`
	`adi`

Table 2. Sample programs that exercise *simple_expression* productions and corresponding P-Code fragments

Test Program	**P-Code**
47 *simple_expression* → *simple_expression* + *term*	

`program se06;`
`  var se:real;`
`  var l:real; var r:integer;`
`begin{se06}`
`  se:=l+r`
`end{se06}.`
	`lvr 0 6`
	`lvi 0 7`
	`flt`
	`adr`

47 *simple_expression* → *simple_expression* + *term*	

`program se07;`
`  var se:real;`
`  var l:integer; var r:real;`
`begin{se07}`
`  se:=l+r`
`end{se07}.`
	`lvi 0 6`
	`flt`
	`lvr 0 7`
	`adr`

47 *simple_expression* → *simple_expression* + *term*	

`program se08;`
`  var se:real;`
`  var l:real; var r:real;`
`begin{se08}`
`  se:=l+r`
`end{se08}.`
	`lvr 0 6`
	`lvr 0 7`
	`adr`

48 *simple_expression* → *simple_expression* - *term*	

`program se09;`
`  var se:integer; var l:integer;`
`  var r:integer;`
`begin{se09}`
`  se:=l-r`
`end{se09}.`
	`lvi 0 6`
	`lvi 0 7`
	`sbi`

48 *simple_expression* → *simple_expression* - *term*	

`program se10;`
`  var se:real;`
`  var l:real; var r:integer;`
`begin{se10}`
`  se:=l-r`
`end{se10}.`
	`lvr 0 6`
	`lvi 0 7`
	`flt`
	`sbr`

48 *simple_expression* → *simple_expression* - *term*	

`program se11;`
`  var se:real;`
`  var l:integer; var r:real;`
`begin{se11}`
`  se:=l-r`
`end{se11}.`
	`lvi 0 6`
	`flt`
	`lvr 0 7`
	`sbr`

48 *simple_expression* → *simple_expression* - *term*	

`program se12;`
`  var se:real;`
`  var l:real; var r:real;`
`begin{se12}`
`  se:=l-r`
`end{se12}.`
	`lvr 0 6`
	`lvr 0 7`
	`sbr`

Table 2. Sample programs that exercise *simple_expression* productions and corresponding P-Code fragments (continued)

Test Program	**P-Code**
49 *simple_expression* → *simple_expression* **or** *term*	

`program se13;`
`  var se:boolean;`
`  var l:boolean; var r:boolean;`
`begin{se13}`
`  se:=l or r`
`end{se13}.`
	`lvb  0  6`
	`lvb  0  7`
	`ior`

Table 2. Sample programs that exercise *simple_expression* productions and corresponding P-Code fragments (continued)