Project: Complete the assembler by writing and implementing semantic rules for the grammar given in project a02.

Program Files:

File	**Description**
PasmScanner.h	File **PasmScanner.h** contains the interface to the lexer and supporting functions defined in file **PasmScanner.l**. Modify **PasmScanner.l** to include file **y.tab.h** created by the utility ***yacc***.
PasmScanner.l	File **PasmScanner.l** contains specifications acceptable to the *Unix* utility ***lex*** for P-Code Assembler tokens.
PasmParser.h	File **PasmParser.h** contains the interface to the parser and supporting functions defined in file **PasmParser.y**.
PasmParser.y	File **PasmParser.y** contains specifications acceptable to the *unix* utility ***yacc*** for the P-Code Assembler grammar. Specifications for the grammar of the P-Code Assembler are given in Table 1.
y.tab.h	File **y.tab.h** is produced by the *unix* utility ***yacc*** as a result of processing file **PasmParser.y**. File **y.tab.h** contains instructions that define the integer codes assigned tokens.
pasm.cpp	File **pasm.cpp** contains function main and processes command line arguments.
PasmFiles.h	File **PasmFiles.h** defines the interface that makes the files required by the P-Code Assembler available throughout the components of the assembler. Files that are made available include the source file, the executable file the listing file and the trace file.
PasmFiles.cpp	File **PasmFiles.cpp** contains the implementation of member functions defined in file **PasmFiles.h** including functions that open and close the files described above.
PasmDirectory.h	File **PasmDirectory.h** defines the P-Code Assembler Directory that specifies the locations of the components in a P-Code Executable (**pex**) file including the locations of the P-Code instructions and constants.
PasmDirectory.cpp	File **PasmDirectory.cpp** contains implementations of member functions that store, read, write, and print the P-Code Assembler Directory.
PasmInstructionArray.h	File **PasmInstructionArray.h** defines the array employed to store P-Code Instructions.

Program Files:

File	Description
PasmInstructionArray.cpp	File **PasmInstructionArray.cpp** implements member functions that permit P-Code instructions to be inserted in the array and modified after they have been placed in the array. Member functions also read, write, and print the array.
PasmInstruction.h	File **PasmInstruction.h** defines a single P-Code instruction.
PasmInstruction.cpp	File **PasmInstruction.cpp** implements member functions that create, store, read, write, and print a single P-Code instruction.
PasmLabel.h	File **PasmLabel.h** defines a single label with members recording the spelling of the label, its integer value, and a list of instruction addresses where the label is referenced.
PasmLabel.cpp	File **PasmLabel.cpp** implements member functions that record a label that has yet to be defined. Other member functions record all instruction addresses where a label is referenced and assist in resolving those undefined references when the label is finally defined.
PasmLabelTable.h	File **PasmLabelTable.h** defines the class used to store, manage, and resolve labels. A label is a symbol for an integer value. Often a label is a symbol for an instruction address but a label can also be an initial value assigned to the stack pointer (**sp**) or the extreme pointer (**ep**) register.
PasmLabelTable.cpp	File **PasmLabelTable.cpp** implements member functions that define, reference, and locate a particular label.
PasmList.h	File **PasmList.h** defines a list of integers. Class PasmList is used in conjunction with class PasmLabelTabel to manage references to a label.
PasmList.cpp	File **PasmList.cpp** implements member functions that insert, remove, and traverse the list of integers.

Program Files:

File	**Description**
PasmIntegerConstants.h	File **PasmIntegerConstants.h** defines the array employed to store integer constants in the P-Code Executable file.
PasmIntegerConstants.cpp	File **PasmIntegerConstants.cpp** implements member functions that insert, print read, and write integer constants.
PasmRealConstants.h	File **PasmRealConstants.h** defines the array employed to store real constants in the P-Code Executable file.
PasmRealConstants.cpp	File **PasmRealConstants.cpp** implements member functions that insert, print read, and write real constants.
PasmSetConstants.h	File **PasmSetConstants.h** defines the array employed to store set constants in the P-Code Executable file.
PasmSetConstants.cpp	File **PasmSetConstants.cpp** implements member functions that insert, print read, and write set constants.
PasmStringConstants.h	File **PasmStringConstants.h** defines the array employed to store set constants in the P-Code Executable file.
PasmStringConstants.cpp	File **PasmStringConstants.cpp** implements member functions that insert, print read, and write string constants.
PasmPCodeFunction.h	File **PasmPCodeFunction.h** defines the relationship between P-Code functions and their integer values.
PasmPCodeOp.h	File **PasmPCodeOp.h** defines the relationship between P-Code operations and their integer values.
PasmPCodeRegister.h	File **PasmPCodeRegister.h** defines the relationship between P-Code registers and their integer values.
PasmPCodeType.h	File **PasmPCodeType.h** defines the relationship between P-Code types and their integer values.

Program Files:	**File**	**Description**
	makepasm	File **makepasm** contains instructions for program **pasm**. Instructions are written for the *Unix* utility ***make***. Program **pasm** is contained in file **pasm**.
	mkpasm	File **mkpasm** is a Unix script file that removes old file created in the last creation of executable file **pas** and invokes file **makepasm** to create a new executable file **pasm**. File makepasm is given below. rm *.o rm pasmlex.cpp rm pasm make -f makepasm

Command Line: Project **2** 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 **pasm** are shown below. Boldfaced type indicates data entered at the keyboard by the user.

```
$ pasm
Enter the input file name: p00.pasm

$ pasm p00.pasm
```

Input File: The input file contains a P-Code Assembler program. The input file name must have the suffix "**.pcd**" Please refer to figure 1 for an example of the format of an input file.

Output Files: There are three (3) output files having suffixes, "**.pex**", "**.atrc**", and "**alst**".

The "**.pex**" output file contains the binary executable form of the input "**.pcd**" file. Descriptions of the format of a P-Code Executable file can be found in http://cs2.uco.edu/~trt/cs4173/pexecutable.pdf.

The "**.atrc**" file contains a trace of the translation steps to help the designed locate and remove errors in the assembler.

The "**.alst**" file is a listing of the original "**.pcd**" file.

The "**.pex**", "**.atrc**" and "**.alst**" files have the same prefix as the input "**.pasm**" file. For example, if the command below is issued

```
$ pasm p00.pcd
```

Files **p00.pex, p00.atrc,** and **p00.alst** are produced as shown in Figure 2.

```
L00001     ent    sp     L00002
           ent    ep     L00003
           rtn    p
#define    L00002  4
#define    L00003  4
           mst    0
           cup    0      L00001
           stp
```

Figure 1. Example input file **p00.pcd.**

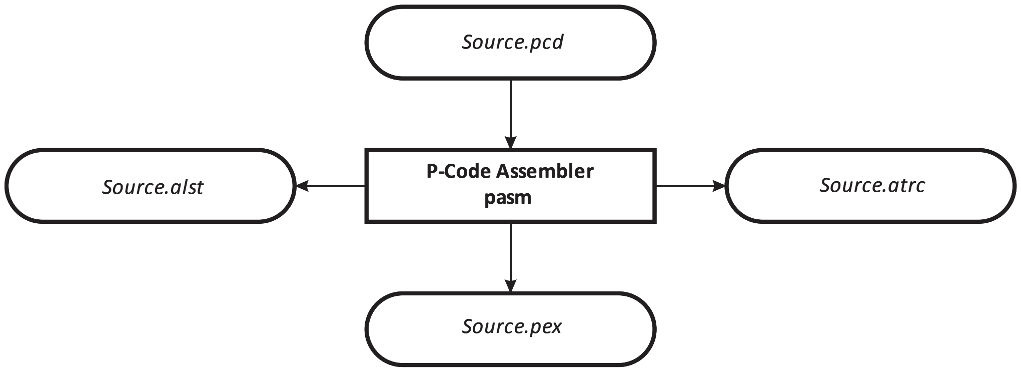

Figure 2. P-Code Assembler Block Diagram

	Rule		
	LHS		**RHS**
1	*program*	→	*list*
2	*list*	→	*item*
3	*list*	→	*list item*
4	*item*	→	*statement*
5	*item*	→	*definition*
6	*definition*	→	**define label intlit**
7	*statement*	→	*label-list operation*
8	*statement*	→	*operation*
9	*label-list*	→	**label**
10	*label-list*	→	*label-list* **label**
11	*operation*	→	*class0-operation*
12	*operation*	→	*class1-operation*
13	*operation*	→	*class2-operation*
14	*operation*	→	*class3-operation*
15	*class0-operation*	→	**adi**
16	*class0-operation*	→	**sbi**
17	*class0-operation*	→	**ngi**
18	*class0-operation*	→	**mpi**
19	*class0-operation*	→	**dvi**
20	*class0-operation*	→	**abi**
21	*class0-operation*	→	**mod**
22	*class0-operation*	→	**adr**
23	*class0-operation*	→	**sbr**
24	*class0-operation*	→	**ngr**
25	*class0-operation*	→	**mpr**
26	*class0-operation*	→	**dvr**
27	*class0-operation*	→	**abr**
28	*class0-operation*	→	**ior**
29	*class0-operation*	→	**xor**
30	*class0-operation*	→	**and**
31	*class0-operation*	→	**not**
32	*class0-operation*	→	**inn**
33	*class0-operation*	→	**uni**
34	*class0-operation*	→	**ntr**
35	*class0-operation*	→	**dif**
36	*class0-operation*	→	**cmp**
37	*class0-operation*	→	**flt**
38	*class0-operation*	→	**flo**
39	*class0-operation*	→	**trc**
40	*class0-operation*	→	**stp**

Table 1 P-Code Assembler Grammar.

	Rule		
	LHS		**RHS**
79	*stdfunction*	$\rightarrow$	**rdb**
80	*stdfunction*	$\rightarrow$	**rdc**
81	*stdfunction*	$\rightarrow$	**rdi**
82	*stdfunction*	$\rightarrow$	**rdr**
83	*stdfunction*	$\rightarrow$	**rln**
84	*stdfunction*	$\rightarrow$	**wrb**
85	*stdfunction*	$\rightarrow$	**wrc**
86	*stdfunction*	$\rightarrow$	**wri**
87	*stdfunction*	$\rightarrow$	**wrr**
88	*stdfunction*	$\rightarrow$	**wrs**
89	*stdfunction*	$\rightarrow$	**wrt**
90	*stdfunction*	$\rightarrow$	**sqt**
91	*stdfunction*	$\rightarrow$	**ln**
92	*stdfunction*	$\rightarrow$	**exp**

Table 1 P-Code Assembler Grammar. (continued)