Computer Science 330 Language Implementation Test Solution

Question 1 — 17 Marks

Write JFlex rules to match the following tokens, and where appropriate return a value, or perform some appropriate action.

(a) A binary integer, composed of the binary indicator "0b", followed by one or more binary digits. For example `0b101011`, but not `0b1234` or `0B101011` or `101011`.

```
0b[01]+
```

(2 marks)

(b) A string composed of a double quote ("), zero or more component characters, then another double quote ("). A component character can be: any character except a control character (\0 to \037, \177) or backslash (\) or double quote ("); or can be a pair of double quotes ("", representing a single "). For example `"He said ""hello"""`, but not `"He said \"hello\""` or `"he said "hello""`.

```
\"([^\0-\037\177\\\"]|\"\")*\"
```

(3 marks)

(c) A decimal integer, with ","s to separate the digits into groups of three, as is normally done when writing numbers in English. For example `12,345,678` but not `12345678` or `123,456,78`.

```
[0-9]{1,3}(\,[0-9]{3})*
```

or a better solution

```
0|[1-9][0-9]{0,2}(\,[0-9]{3})*
```

(7 marks)

(d) C style `/* ... */` comments, but allowing nesting of comments. For example,

```
/*
A comment.
/* A nested comment */
And more of the same comment.
*/
```

Add appropriate actions to increment the line count, when a line break occurs. Do not return a value.

```
<NORMAL> {
      "/*"  { yybegin( COMMENT ); nest++; }
      }
<COMMENT> {
      "/*"  { nest++; }
      "*/"  { --nest; if ( nest == 0 ) yybegin( NORMAL ); }
      \r|\r\n|\n  { lineCount++; }
      .     { }
      }
```

(5 marks)

Question 2 — 63 marks

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

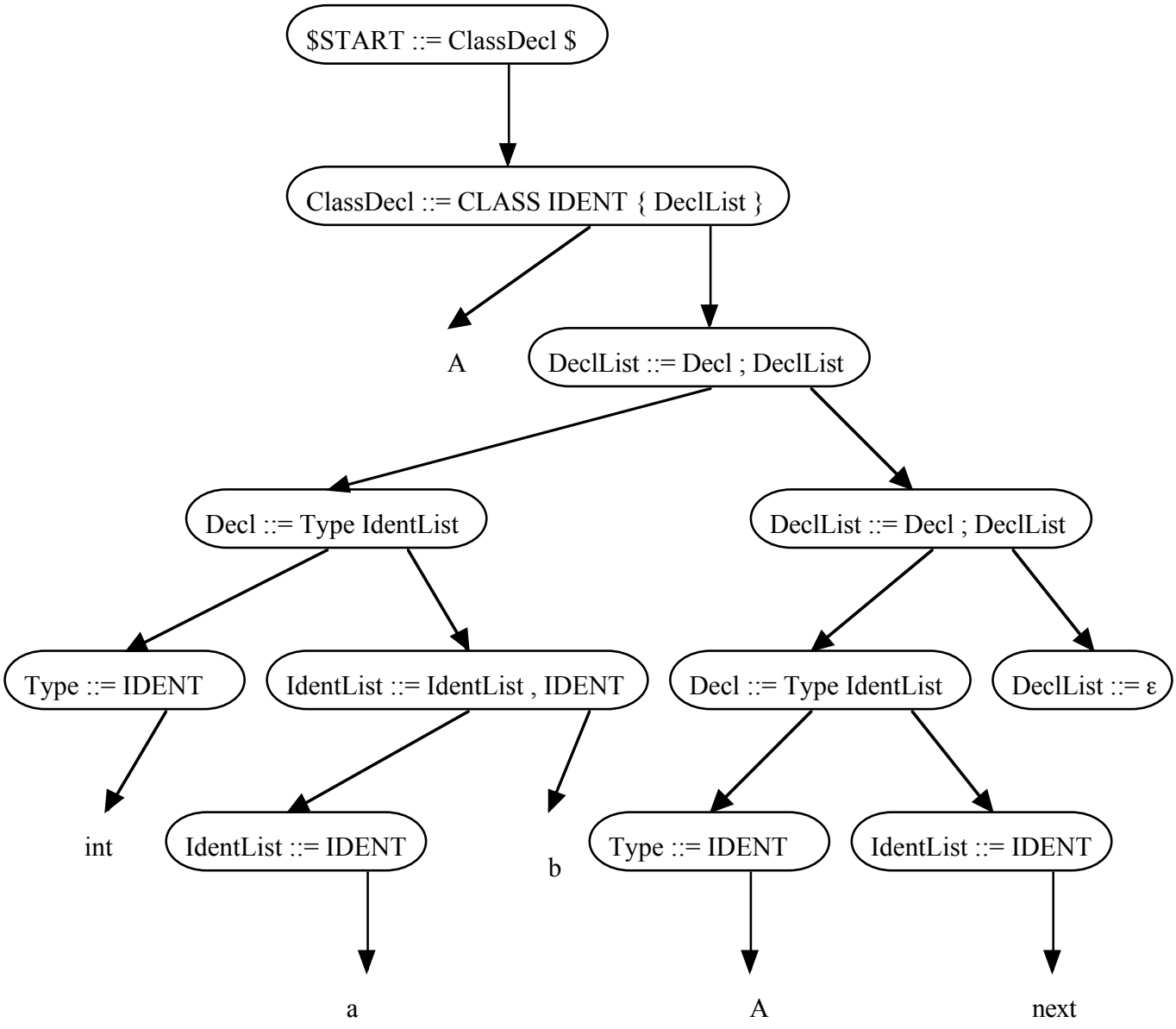
```
class A { int a, b; A next; }
```

Show both the symbols and states on the stack, the current token, and the action performed at each stage. (Consider "int" to be an IDENT). (22 marks)

Stack									Input	Action	
$0									class	Shift	class 2
$0	class 2								ID A	Shift	ID 3
$0	class 2	ID 3							{	Shift	{ 4
$0	class 2	ID 3	{ 4						ID int	Shift	ID 8
$0	class 2	ID 3	{ 4	ID 8					ID a	Reduce	Type ::= ID
$0	class 2	ID 3	{ 4	Type 6						Shift	ID 12
$0	class 2	ID 3	{ 4	Type 6	ID 12				,	Reduce	IdList ::= ID
$0	class 2	ID 3	{ 4	Type 6	IdL 13					Shift	, 14
$0	class 2	ID 3	{ 4	Type 6	IdL 13	, 14			ID b	Shift	ID 15
$0	class 2	ID 3	{ 4	Type 6	IdL 13	, 14	ID 15		;	Reduce	IdList ::= IDList , ID
$0	class 2	ID 3	{ 4	Type 6	IdL 13					Reduce	Decl ::= Type IdList
$0	class 2	ID 3	{ 4	Decl 9						Shift	; 10
$0	class 2	ID 3	{ 4	Decl 9	; 10				ID A	Shift	ID 8
$0	class 2	ID 3	{ 4	Decl 9	; 10	ID 8			ID next	Reduce	Type ::= ID
$0	class 2	ID 3	{ 4	Decl 9	; 10	Type 6				Shift	ID 12
$0	class 2	ID 3	{ 4	Decl 9	; 10	Type 6	ID 12		;	Reduce	IdList ::= ID
$0	class 2	ID 3	{ 4	Decl 9	; 10	Type 6	IdL 13			Reduce	Decl ::= Type IdList
$0	class 2	ID 3	{ 4	Decl 9	; 10	Decl 9				Shift	; 10
$0	class 2	ID 3	{ 4	Decl 9	; 10	Decl 9	; 10		}	Reduce	DeclL ::= empty
$0	class 2	ID 3	{ 4	Decl 9	; 10	Decl 9	; 10	DeclL 11		Reduce	DeclL ::= Decl ; DeclL
$0	class 2	ID 3	{ 4	Decl 9	; 10	DeclL 11				Reduce	DeclL ::= Decl ; DeclL
$0	class 2	ID 3	{ 4	DeclL 5						Shift	} 16
$0	class 2	ID 3	{ 4	DeclL 5	} 16				$	Reduce	CDecl ::= class ID { DeclL }
$0	CDecl 1									Shift	$17
$0	CDecl 1	$17								Reduce	$start ::= ClassDecl $
$0	$Start -1									Accept	

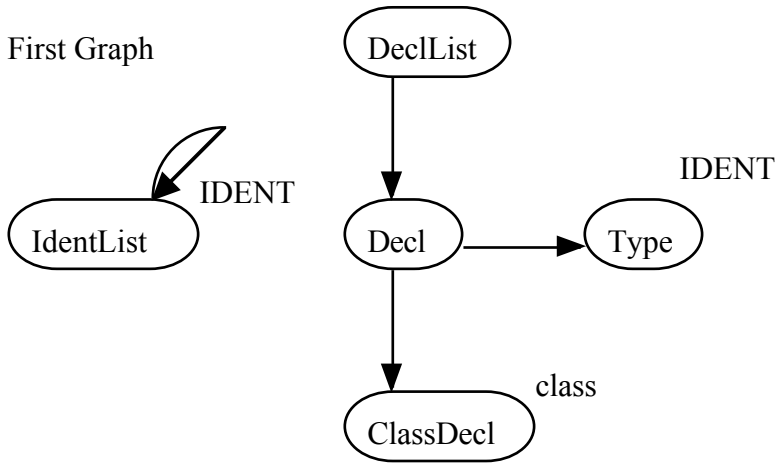
(b) Draw the full parse tree, showing all rules used in the above shift-reduce LALR(1) parse.

(8 marks)

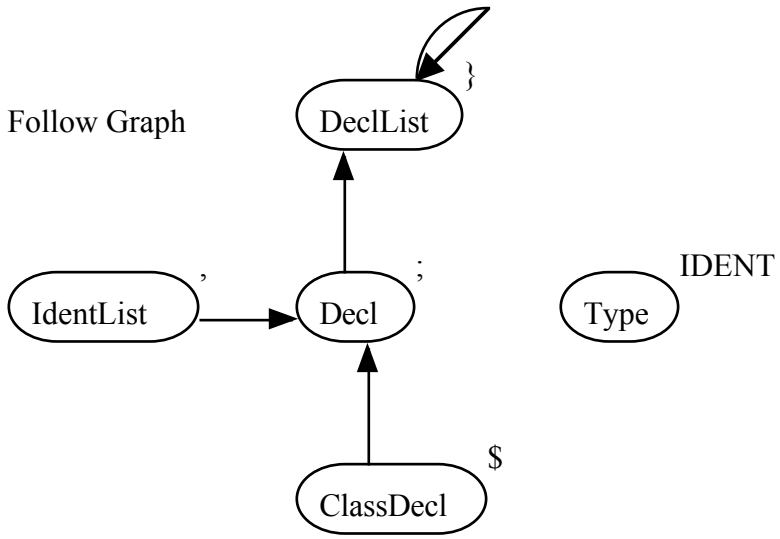


(c) (20 marks)

Draw the first graph for this grammar.



Draw the follow graph for this grammar.



Indicate the first and follow sets for the grammar

Symbol	First Set	Follow Set
DeclList	Class IDENT	}
Decl	Class IDENT	; }
ClassDecl	class	$; }
Type	IDENT	IDENT
IdentList	IDENT	, ; }

(d) State 10 is ...

(i) Write down the set of items for goto(state 10, Decl) (state 9).

```
lalr_state [9]: {
  [DeclList ::= Decl (*) SEMICOLON DeclList , {RIGHTBRACE }]
  [DeclList ::= Decl (*) , {RIGHTBRACE }]
}
```

(5 marks)

(ii) Write down the set of items for goto(state 10, Type) (state 6). Make sure you take the closure.

```
lalr_state [6]: {
  [Decl ::= Type (*) IdentList , {RIGHTBRACE SEMICOLON }]
  [IdentList ::= (*) IDENT , {RIGHTBRACE COMMA SEMICOLON }]
  [IdentList ::= (*) IdentList COMMA IDENT , {RIGHTBRACE COMMA SEMICOLON }]
}
```

(8 marks)

Question 3 — 20 marks

```
State ::=
            "lalr_state" "[" INTCONST "]" "{" ItemList "}" TransitionList
      ;
ItemList::=
            Item
      |
            ItemList Item
      ;
Item::=
            "[" Rule "," FollowSet "]"
      ;
Rule::=
            IDENT "::=" SymbolList "(*)" SymbolList
      ;
SymbolList::=
            /* empty */
      |
            SymbolList IDENT
      ;
FollowSet::=
            "{" ElementList "}"
      ;
ElementList::=
            IDENT
      |
            ElementList IDENT
      ;
TransitionList::=
            /* Empty */
      |
            TransitionList TransitionItem
      ;
TransitionItem::=
            "transition" "on" IDENT "to" "state" "[" INTCONST "]"
      ;
```