Jessie George
HW3
Section 1

Question 1

Rule 1 <program> ::= **program** <block> **.**
Rule 2 <block> ::= **begin** <stmtlist> **end**
Rule 3 <stmtlist> ::= <stmt> <morestmts>
Rule 4 <morestmts> ::= **;** <stmtlist>
Rule 5 <morestmts> ::= Ɛ
Rule 6 <stmt> ::= <assign>
Rule 7 <stmt> ::= <ifstmt>
Rule 8 <stmt> ::= <whilestmt>
Rule 9 <stmt> ::= <block>
Rule 10 <assign> ::= <variable> = <expr>
Rule 11 <ifstmt> ::= **if** <testexpr> **then** <stmt> **else** <stmt>
Rule 12 <whilestmt> ::= **while** <testexpr> **do** <stmt>
Rule 13 <testexpr> ::= <variable> <= <expr>
Rule 14 <expr> ::= +<expr><expr>
Rule 15 <expr> ::= *<expr><expr>
Rule 16 <expr> ::= <variable>
Rule 17 <expr> ::= <digit>
Rule 18 <variable> ::= a|b|c
Rule 19 <digit> ::= 0|1|2

FIRST(<program>) = {**program**} by Rule 1
FOLLOW(<program>) = {eof, <block>**.**} by Rule 1

FIRST(<block>) = {**begin**} by Rule 2
FOLLOW(<block>) = {<stmtlist>, **end**} by Rule 2

FIRST(<stmtlist>) = {<stmt>} by Rule 3
FOLLOW(<stmtlist>) = {<morestmts>} by Rule 3

FIRST(<morestmts>) = {**;**} by Rule 4
FIRST(Ɛ) = {Ɛ} by Rule 5
FOLLOW(<morestmts>) = {<stmtlist>, **end**} by Rules 4 and 2

FIRST(<stmt>) = {<assign>, <ifstmt>, <whilestmt>, <block>} by Rules 6,7,8,9
FOLLOW(<stmt>) = {<morestmts>} by Rule 3

FIRST(<assign>) = {<variable> =} by Rule 10
FOLLOW(<assign>) = {<expr>} by Rule 10

FIRST(<ifstmt>) = {**if**} by Rule 11
FOLLOW(<ifstmt>) = {<testexpr> **then** <stmt> **else** <stmt>} by Rule 11

FIRST(<whilestmt>) = {**while**} by Rule 12
FOLLOW(<whilestmt>) = {<testexpr> **do** <stmt>} by Rule 12

FIRST(<testexpr>) = {<variable> <=} by Rule 13
FOLLOW(<testexpr>) = {<expr>} by Rule 13

FIRST(<expr>) = {+<expr><expr>, *<expr><expr>, <variable>, <digit>} by Rules 14,15,16,17
FOLLOW(<expr>) = {<morestmts>, **then**, **do**} by Rules 3,6,7,8,10,11,12,13

FIRST(<variable>) = {a,b,c} by Rule 18
FOLLOW(<variable>) = {=, <=, <morestmts>, **then**, **do**} by Rules 10,13,16 and the logic for follow set of expr which is given above

FIRST(<digit>) = {0,1,2} by Rule 19
FOLLOW(<digit>) = {<morestmts>, **then**, **do**} by Rule 17 and the logic for follow set of expr which is given above

FIRST$^+$(<program>) = {**program**}
FIRST$^+$(<block>) = {**begin**}
FIRST$^+$(<stmtlist>) = {<stmt>}
FIRST$^+$(<morestmts>) = {**;**}
FIRST$^+$(Ɛ) = (FIRST(Ɛ) – {Ɛ}) ∪FOLLOW(<morestmts>) = {<stmtlist>, **end**}
FIRST$^+$(<stmt>) = {<assign>, <ifstmt>, <whilestmt>, <block>}
FIRST$^+$(<assign>) = {<variable> =}
FIRST$^+$(<ifstmt>) = {**if**}
FIRST$^+$(<whilestmt>) = {**while**}
FIRST$^+$(<testexpr>) = {<variable> <=}
FIRST$^+$(<expr>) = {+<expr><expr>, *<expr><expr>, <variable>, <digit>}
FIRST$^+$(<variable>) = {a,b,c}
FIRST$^+$(<digit>) = {0,1,2}

FIRST$^+$(<program>) ∩FIRST$^+$(<block>) ∩FIRST$^+$(<stmtlist>) ∩FIRST$^+$(<morestmts>) ∩FIRST$^+$(Ɛ) ∩FIRST$^+$(<stmt>) ∩FIRST$^+$(<assign>) ∩ FIRST$^+$(<ifstmt>) ∩FIRST$^+$(<whilestmt>) ∩FIRST$^+$(<testexpr>) ∩FIRST$^+$(<expr>) ∩FIRST$^+$(<variable>) ∩FIRST$^+$(<digit>) = Ø

Therefore, the given grammar is LL(1).

See next page for Question 2.

Question 2: Parse Table (the numbers in the table are the rule numbers on page 1)

	Tokens	program	.	begin	end	if	then	else	while	do	=	<=	+	*	a	b	c	0	1	2	;	eof	other
Rules																							
<program>		1	1																			Ɛ	Error
<block>				2	2																		
<stmtlist>																							
<morestmts>																					4		
Ɛ																							
<stmt>																							
<assign>											10												
<ifstmt>						11	11	11															
<whilestmt>									12	12													
<testexpr>												13											
<expr>													14	15									
<variable>															18	18	18						
<digit>																		19	19	19			

Question 3

```
main:   {
        token := next_token();
        if (S() and token == eof) print “accept” else print “error”;
}

bool S:
{
        if(token != program)
                return false;

        token := next_token();
        if(not block())
                return false;

        token := next_token();
        if(token != .)
                return false;

        token := next_token();
        return true;
}

bool block:
{
        if (token!=begin)
                return false;

        token := next_token();
        if(not stmtlist())
```

```
                return false;

        if(token != end)
                return false;

        return true;
}

bool stmtlist:
{

        if(not stmt())
                return false;

        token := next_token();
        if(not morestmts())
                return false;

        return true;
}

bool stmt:
{
        if(variable())
        {       if(not assign())
                        return false;

                return true;
        }

        else if(token==if)
```

```
	{
		if(not ifstmt())
			return false;

		return true;
	}

	else if(token==while)
	{
		if(not whilestmt())
			return false;

		return true;
	}

	else if(token==begin)
	{
		if(not block())
			return false;

		return true;
	}

	else
		return false;
}

bool morestmts:
{
	if(token == ;)
	{
```

```
                token := next_token();
                if(not stmtlist())
                        return false;

                return true;
        }

        else if(token == end)
                return true;

        else
                return false;

}

bool assign:
{
        if(not variable())
                return false;

        token := next_token();
        if(token != =)
                return false;

        token := next_token();
        if(not expr())
                return false;

        return true;
}
```

```
bool ifstmt:
{
	if(token != if)
		return false;

	token := next_token();
	if(not testexpr())
		return false;

	token := next_token();
	if(token != then)
		return false;

	token := next_token();
	if(not stmt())
		return false;

	token := next_token();
	if(token != else)
		return false;

	token := next_token();
	if(not stmt())
		return false;

	return true;
}

bool whilestmt:
{
	if(token != while)
```

```
                return false;

        token := next_token();
        if(not testexpr())
                return false;

        token := next_token();
        if(token != do)
                return false;

        token := next_token();
        if(not stmt())
                return false;

        return true;
}

bool testexpr:
{
        if(not variable())
                return false;

        token := next_token();
        if(token != <=)
                return false;

        token := next_token();
        if(not expr())
                return false;
```

```
        return true;
}

bool expr:
{
        if(token == + || token == *)
        {
                token := next_token();
                if(not expr())
                        return false;

                token := next_token();
                if(not expr())
                        return false;

                return true;
        }

        else if(variable())
                return true;

        else if(digit())
                return true;

        else
                return false;
}

bool variable:
{
        if(token == a || token == b || token == c)
```

```
                return true;

        return false;
}

bool digit:
{
        if(token == 0 || token == 1 || token == 2)
                return true;

        return false;
}
```

Question 4

```
main:   {

        int numAssignments := 0;
        int numBinaryOps := 0;
        token := next_token();
        if (S() and token == eof)
        {
                print “accept”;
                print numAssignments;
                print numBinaryOps;
        }

        else
                print “error”;
}

bool S:
{
        if(token != program)
                return false;

        token := next_token();
        if(not block())
                return false;

        token := next_token();
        if(token != .)
                return false;
```

```
        token := next_token();
        return true;
}

bool block:
{
        if (token!=begin)
                return false;

        token := next_token();
        if(not stmtlist())
                return false;

        if(token != end)
                return false;

        return true;
}

bool stmtlist:
{

        if(not stmt())
                return false;

        token := next_token();
        if(not morestmts())
                return false;

        return true;
}
```

```
bool stmt:
{
        if(variable())
        {       if(not assign())
                        return false;

                return true;
        }

        else if(token==if)
        {
                if(not ifstmt())
                        return false;

                return true;
        }

        else if(token==while)
        {
                if(not whilestmt())
                        return false;

                return true;
        }

        else if(token==begin)
        {
                if(not block())
                        return false;
```

```
                return true;
        }

        else
                return false;
}

bool morestmts:
{
        if(token == ;)
        {
                token := next_token();
                if(not stmtlist())
                        return false;

                return true;
        }

        else if(token == end)
                return true;

        else
                return false;

}

bool assign:
{
        if(not variable())
                return false;
```

```
        token := next_token();
        if(token != =)
                return false;

        token := next_token();
        if(not expr())
                return false;

        numAssignments++;
        return true;
}

bool ifstmt:
{
        if(token != if)
                return false;

        token := next_token();
        if(not testexpr())
                return false;

        token := next_token();
        if(token != then)
                return false;

        token := next_token();
        if(not stmt())
                return false;

        token := next_token();
        if(token != else)
```

```
                return false;

        token := next_token();
        if(not stmt())
                return false;

        return true;
}

bool whilestmt:
{
        if(token != while)
                return false;

        token := next_token();
        if(not testexpr())
                return false;

        token := next_token();
        if(token != do)
                return false;

        token := next_token();
        if(not stmt())
                return false;

        return true;
}

bool testexpr:
{
```

```
        if(not variable())
                return false;

        token := next_token();
        if(token != <=)
                return false;

        token := next_token();
        if(not expr())
                return false;

        numBinaryOps++;
        return true;
}

bool expr:
{
        if(token == + || token == *)
        {
                token := next_token();
                if(not expr())
                        return false;

                token := next_token();
                if(not expr())
                        return false;

                numBinaryOps++;
                return true;
        }
```

```
        else if(variable())
                return true;

        else if(digit())
                return true;

        else
                return false;
}

bool variable:
{
        if(token == a || token == b || token == c)
                return true;

        return false;
}

bool digit:
{
        if(token == 0 || token == 1 || token == 2)
                return true;

        return false;
}
```