

Jessie George

HW3

Section 1

### Question 1

- Rule 1       $\langle \text{program} \rangle ::= \text{program } \langle \text{block} \rangle .$   
Rule 2       $\langle \text{block} \rangle ::= \text{begin } \langle \text{stmtlist} \rangle \text{ end}$   
Rule 3       $\langle \text{stmtlist} \rangle ::= \langle \text{stmt} \rangle \langle \text{morestmts} \rangle$   
Rule 4       $\langle \text{morestmts} \rangle ::= ; \langle \text{stmtlist} \rangle$   
Rule 5       $\langle \text{morestmts} \rangle ::= \epsilon$   
Rule 6       $\langle \text{stmt} \rangle ::= \langle \text{assign} \rangle$   
Rule 7       $\langle \text{stmt} \rangle ::= \langle \text{ifstmt} \rangle$   
Rule 8       $\langle \text{stmt} \rangle ::= \langle \text{whilestmt} \rangle$   
Rule 9       $\langle \text{stmt} \rangle ::= \langle \text{block} \rangle$   
Rule 10      $\langle \text{assign} \rangle ::= \langle \text{variable} \rangle = \langle \text{expr} \rangle$   
Rule 11      $\langle \text{ifstmt} \rangle ::= \text{if } \langle \text{testexpr} \rangle \text{ then } \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle$   
Rule 12      $\langle \text{whilestmt} \rangle ::= \text{while } \langle \text{testexpr} \rangle \text{ do } \langle \text{stmt} \rangle$   
Rule 13      $\langle \text{testexpr} \rangle ::= \langle \text{variable} \rangle \leq \langle \text{expr} \rangle$   
Rule 14      $\langle \text{expr} \rangle ::= + \langle \text{expr} \rangle \langle \text{expr} \rangle$   
Rule 15      $\langle \text{expr} \rangle ::= * \langle \text{expr} \rangle \langle \text{expr} \rangle$   
Rule 16      $\langle \text{expr} \rangle ::= \langle \text{variable} \rangle$   
Rule 17      $\langle \text{expr} \rangle ::= \langle \text{digit} \rangle$   
Rule 18      $\langle \text{variable} \rangle ::= a | b | c$   
Rule 19      $\langle \text{digit} \rangle ::= 0 | 1 | 2$

$\text{FIRST}(\langle \text{program} \rangle) = \{\text{program}\}$       by Rule 1

$\text{FOLLOW}(\langle \text{program} \rangle) = \{\text{eof}, \langle \text{block} \rangle.\}$       by Rule 1

$\text{FIRST}(\langle \text{block} \rangle) = \{\text{begin}\}$       by Rule 2

$\text{FOLLOW}(\langle \text{block} \rangle) = \{\langle \text{stmtlist} \rangle, \text{end}\}$       by Rule 2

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

FIRST(<morestmts>) = {;}                              by Rule 4  
FIRST(ε) = {ε}                                          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

$\text{FIRST}^+(\langle \text{program} \rangle) = \{\mathbf{program}\}$   
 $\text{FIRST}^+(\langle \text{block} \rangle) = \{\mathbf{begin}\}$   
 $\text{FIRST}^+(\langle \text{stmtlist} \rangle) = \{\langle \text{stmt} \rangle\}$   
 $\text{FIRST}^+(\langle \text{morestmts} \rangle) = \{;\}$   
 $\text{FIRST}^+(\epsilon) = (\text{FIRST}(\epsilon) - \{\epsilon\}) \cup \text{FOLLOW}(\langle \text{morestmts} \rangle) = \{\langle \text{stmtlist} \rangle, \mathbf{end}\}$   
 $\text{FIRST}^+(\langle \text{stmt} \rangle) = \{\langle \text{assign} \rangle, \langle \text{ifstmt} \rangle, \langle \text{whilestmt} \rangle, \langle \text{block} \rangle\}$   
 $\text{FIRST}^+(\langle \text{assign} \rangle) = \{\langle \text{variable} \rangle =\}$   
 $\text{FIRST}^+(\langle \text{ifstmt} \rangle) = \{\mathbf{if}\}$   
 $\text{FIRST}^+(\langle \text{whilestmt} \rangle) = \{\mathbf{while}\}$   
 $\text{FIRST}^+(\langle \text{testexpr} \rangle) = \{\langle \text{variable} \rangle <=\}$   
 $\text{FIRST}^+(\langle \text{expr} \rangle) = \{+\langle \text{expr} \rangle \langle \text{expr} \rangle, * \langle \text{expr} \rangle \langle \text{expr} \rangle, \langle \text{variable} \rangle, \langle \text{digit} \rangle\}$   
 $\text{FIRST}^+(\langle \text{variable} \rangle) = \{a, b, c\}$   
 $\text{FIRST}^+(\langle \text{digit} \rangle) = \{0, 1, 2\}$

$\text{FIRST}^+(\langle \text{program} \rangle) \cap \text{FIRST}^+(\langle \text{block} \rangle) \cap \text{FIRST}^+(\langle \text{stmtlist} \rangle) \cap \text{FIRST}^+(\langle \text{morestmts} \rangle) \cap \text{FIRST}^+(\epsilon) \cap \text{FIRST}^+(\langle \text{stmt} \rangle) \cap \text{FIRST}^+(\langle \text{assign} \rangle) \cap$   
 $\text{FIRST}^+(\langle \text{ifstmt} \rangle) \cap \text{FIRST}^+(\langle \text{whilestmt} \rangle) \cap \text{FIRST}^+(\langle \text{testexpr} \rangle) \cap \text{FIRST}^+(\langle \text{expr} \rangle) \cap \text{FIRST}^+(\langle \text{variable} \rangle) \cap \text{FIRST}^+(\langle \text{digit} \rangle) = \emptyset$

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)

	T o k e n s	p r o g r a m	.	b e g i n	e n d	if	then	else	while	do	=	<=	+	*	a	b	c	0	1	2	;	e o f	other
Rules																							
<program>		1	1																			ε	Error
<block>				2	2																		
<stmtlist>																							
<morestmts >																					4		
ε																							
<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;
    }
```