procedure match(expectedToken)

begin

if token = expectedToken{

call ADVANCE();

}

else{

call ERROR();

}

end match;

<program> := [start] [start\_main] [identifier] <stmts> [end\_main] [end]

procedure <program>;

begin

case token of [start]:

match ([start]);

match ([start\_main]);

match ([identifier]);

call (<stmts>);

match ([end\_main]);

match ([end]);

case token of empty:

ERROR();

end <program>;

<stmts> := <stmt> <stmts\_prime>

procedure <stmts>

begin

call (<stmt>);

if token == [terminator]{

call (<stmts\_prime>);

}

end <stmts>

<stmts\_prime> := <stmts> | ε

procedure <stmts\_prime>

begin

if match (empty){

return;

}

else{

call (<stmts>);

}

end <stmts\_prime>

<stmt> := <declaration> [terminator] |

<assign> [terminator] |

[increment] [identifier] [terminator] |

[decrement] [identifier] [terminator] |

<input> [terminator] |

<output> [terminator] |

<cond\_stmt> |

<loop> |

<switch\_a>

procedure <stmt>

begin

case 1: { call (<declaration>);

if token == [terminator]:

match ([terminator]);

else

ERROR();

}

case 2: { call (<assign>);

if token == [terminator]:

match ([terminator]);

else

ERROR();

}

case 3: { if token == [increment]{

match ([increment]);

match ([identifier]);

match ([terminator]); }

case 4: { if token == [decrement]{

match ([decrement]);

match ([identifier]);

match ([terminator]); }

case 5: { call (<input>);

if token == [terminator]

match ([terminator]); }

case 6: { call (<output);

if token == [terminator]

match ([terminator]); }

case 7: { call (<cond\_stmt>); }

case 8: { call (<loop>); }

case 9: { call (<switch\_a>); }

case 10: { if token == empty

ERROR(); }

end <stmt>

<declaration> := [data\_type\_int] [identifier] <declaration\_A> | [data\_type\_dec] [identifier] <declaration\_A> | [data\_type\_bool] [identifier] <declaration\_B> | [data\_type\_string] [identifier] <declaration\_C>

procedure <declaration>

begin

case token of [data\_type\_int]:

match ([data\_type\_int]);

match ([identifier]);

call (<declaration\_A>);

case token of [data\_type\_dec]:

match ([data\_type\_dec]);

match ([identifier]);

call (<declaration\_A>);

case token of [data\_type\_bool]:

match ([data\_type\_bool]);

match ([identifier]);

call (<declaration\_B>);

case token of [data\_type\_string]:

match ([data\_type\_string]);

match ([identifier]);

call (<declaration\_C>);

case token of empty:

ERROR();

end <declaration>

<declaration\_A> := [assign\_op] <arith\_expr>

procedure <declaration\_A>

begin

case token of [assign\_op]:

match ([assign\_op]);

call (<arith\_expr>);

case token of empty:

ERROR();

end <declaration\_A>

<declaration\_B> := [assign\_op] <bool\_expr>

procedure <declaration\_B>

begin

case token of [assign\_op]:

match ([assign\_op]);

call (<bool\_expr>);

case token of empty:

ERROR();

end <declaration\_B>

<declaration\_C> := [assign\_op] [string\_const]

procedure <declaration\_C>

begin

case token of [assign\_op]:

match ([assign\_op]);

match ([string\_const]);

case token of empty:

ERROR();

end <declaration\_C>

<arith\_expr> := <a\_term> <arith\_expr\_prime>

procedure <arith\_expr>

begin

call (<a\_term>);

if token == last token from <a\_term>{

call (<arith\_expr\_prime>);

}

end <arith\_expr>

<arith\_expr\_prime> := [arithas] <a\_term> <arith\_expr\_prime> | ε

procedure <arith\_expr\_prime>

begin

case token of [arithas]:

match ([arithas]);

call (<a\_term>);

if token == empty

return;

else

while token == last token from <a\_term>

call (<arith\_expr\_prime>);

end while

end <arith\_expr\_prime>

<a\_term> := <a\_factor> <a\_term\_prime>

procedure <a\_term>

begin

call (<a\_factor>);

if token == last token from (<a\_factor>){

call (<a\_term\_prime>);

}

end <a\_term >

<a\_term\_prime> := [arithmd] <a\_factor> <a\_term\_prime> | ε

procedure <a\_term\_prime>

begin

case token of [arithmd]:

match ([arithmd]);

call (<a\_factor>);

if token == empty

return;

else

while token == last token from <a\_factor> {

call (<a\_term\_prime>);

}

end <a\_term\_prime>

<a\_factor> := [identifier] | [int\_const] | [dec\_const] | [lpar] <arith\_expr> [rpar] | <arith\_expr>

procedure <a\_factor>

begin

case 1: if token == [identifier]:

match ([identifier]);

case 2: if token == [int\_const]:

match ([int\_const]);

case 3: if token == [dec\_const]:

match ([dec\_const]);

case 4: if token == [lpar]:

match ([lpar]);

call (<arith\_expr>);

match ([rpar]);

case 5:

call (<arith\_expr>);

case 6: if token == empty

ERROR();

end <a\_factor>

<rel\_expr> := <r\_term> <rel\_expr\_prime>

procedure <rel\_expr>

begin

call (<r\_term>);

if token == last of <r\_term> {

call (<rel\_expr\_prime>);

}

end <rel\_expr>

<rel\_expr\_prime> := [relop] <r\_term\_a> <rel\_expr\_prime> | ε

procedure <rel\_expr\_prime>

begin

case token of [relop]:

match ([relop]);

call (<r\_term\_a>);

if token == empty

return;

else

while token == last token from <r\_term\_a> {

call (<a\_term\_prime>);

}

call (<rel\_expr\_prime>);

end <rel\_expr>

<r\_term> := [identifier] | [bool\_const]

procedure <r\_term>

begin

case token of [identifier]:

match ([identifier]);

case token of [bool\_const]:

match ([bool\_const]);

case token of empty:

ERROR();

end <r\_term>

<r\_term\_a> := <arith\_expr> | [int\_const] | [dec\_const] | [string\_const] | [lpar] <rel\_expr> [rpar]

procedure <r\_term\_a>

begin

case 1: token of [int\_const]:

match ([int\_const]);

case 2: token of [dec\_const]:

match ([dec\_const]);

case 3: token of [string\_const]:

match ([string\_const]);

case 4: token of [lpar]:

match ([lpar]);

call (<rel\_expr>);

match ([rpar]);

case 5: call (<r\_term>);

case 6: call (<arith\_expr>);

case 7: if token == empty:

ERROR();

end <r\_term\_a>

<bool\_expr> := [logn\_optr] <bool\_expr> | <b\_term>

procedure <bool\_expr>

begin

case 1 : if token == [logn\_optr]:

match ([logn\_optr]);

call (<bool\_expr>);

match ([dec\_const]);

case 2:

call (<b\_term>);

case 3: if token == empty

ERROR();

end <bool\_expr>

<b\_term> := <b\_factor> <b\_term\_prime>

procedure <b\_term>

begin

call (<b\_factor>);

if token == last token of <b\_factor> {

call (<b\_term\_prime>);

}

end <b\_term>

<b\_term\_prime> := [logao\_optr] <b\_factor> <b\_term\_prime> | ε

procedure <b\_term\_prime>

begin

case token of [logao\_optr]:

match ([logao\_optr]);

call (<b\_factor>);

if token == empty

return;

else

while token == last of <b\_factor>{

call (<b\_term\_prime>);

}

end <b\_term\_prime>

<b\_factor> := [identifier] | [bool\_const] | <rel\_expr> | [lpar] <bool\_expr> [rpar] | <bool\_expr>

procedure <b\_factor>

begin

case 1: if token == [identifier]:

match ([identifier]);

case 2: if token == [bool\_const]:

match ([bool\_const]);

case 3: if token == [lpar]:

match ([lpar]);

call (<bool\_expr>);

match ([rpar]);

case 4: call (<rel\_expr>);

case 5: call (<bool\_expr>);

case 6: if token == empty

ERROR();

end <b\_factor>

<assign> := [identifier] [assign\_op] <assign\_prime>

procedure <assign>

begin

case token of [identifier]:

match ([identifier]);

match ([assign\_op]);

call (<assign\_prime>);

case token of empty:

ERROR();

end <assign>

<assign\_prime> := <arith\_expr> | <bool\_expr> | [string\_const]

procedure <assign\_prime>

begin

case 1: if token == [string\_const]:

match ([string\_const]);

case 2: call (<arith\_expr>);

case 3: call (<bool\_expr>);

case 4: if token == empty

ERROR();

end <assign\_prime>

<input> := [input\_type] [identifier]

procedure <input>

begin

case token of [input\_type]:

match ([input\_type]);

match ([identifier]);

case token of empty:

ERROR();

end <input>

<output> := [output\_type] <output\_prime>

procedure <output>

begin

case token of [output\_type]:

match ([output\_type]);

call (<output\_prime>);

case token of empty:

ERROR();

end <output>

<output\_prime> := <arith\_expr> | <bool\_expr> | [string\_const]

procedure <output\_prime>

begin

case 1: if token == [string\_const]:

match ([string\_const]);

case 2: call (<arith\_expr>);

case 3: call (<bool\_expr>);

case 4: if token == empty

ERROR();

end <output\_prime>

<cond\_stmt> := <cond\_stmt\_a> <cond\_stmt\_prime>

procedure <cond\_stmt>

begin

call (<cond\_stmt\_a>);

if token == last token from <cond\_stmt\_a>{

call (<cond\_stmt\_prime>);

}

end <cond\_stmt>

<cond\_stmt\_prime> := <cond\_stmt> | EMPTY

procedure <cond\_stmt\_prime>

begin

case 1: call (<cond\_stmt>);

case 2: call (<cond\_stmt\_a>);

case 3: if token == empty

ERROR();

end <cond\_stmt\_prime>

<cond\_stmt\_a> := [cond\_if] [lpar] <bool\_expr> [rpar] [cond\_then] <if\_terms> [cond\_endif]

procedure <cond\_stmt\_a>

begin

case token of [cond\_if]:

match ([cond\_if]);

match ([lpar]);

call (<bool\_expr>);

match ([rpar]);

match ([cond\_then]);

call (<if\_terms>);

match ([cond\_endif]);

case token of empty:

ERROR();

end <cond\_stmt\_a>

<if\_terms> := <stmts> <if\_terms\_prime>

procedure <if\_terms>

begin

call (<stmts>);

if token == [terminator] {

call (<if\_terms\_prime>);

}

end <if\_terms>

<if\_terms\_prime> := ε | <optional\_elseif> | <optional\_else>

procedure <if\_terms\_prime>

begin

case 1: call (<optional\_elseif>);

case 2: call (<optional\_else>);

end <if\_terms\_prime>

<optional\_elseif> := <optional\_elseif\_a> <optional\_elseif\_Aprime>

procedure <optional\_elseif>

begin

call (<optional\_elseif\_a>);

if token == [terminator]{

call (<optional\_elseif\_Aprime>);

}

end <optional\_elseif>

<optional\_elseif\_Aprime> := <optional\_elseif> <optional\_elseif\_prime> | <optional\_else> <optional\_elseif\_prime>

procedure <optional\_elseif\_Aprime>

begin

case 1: call (<optional\_elseif>);

if token == last token of <optional\_else\_if>

call (<optional\_elseif\_prime>);

}

case 2: call (<optional\_else>)

if token == last token of <optional\_else>

call (<optional\_elseif\_prime>);

}

case 3: if token == empty

ERROR();

end <optional\_elseif\_Aprime>

<optional\_elseif\_prime> := <optional\_elseif> <optional\_elseif\_prime> | ε

procedure <optional\_elseif\_prime>

begin

call (<optional\_elseif>);

if token == last token of <optional\_elseif>{

call (<optional\_elseif\_prime>);

}

end <optional\_elseif\_prime>

<optional\_elseif\_a>:= [cond\_elseif] [lpar] <bool\_expr> [rpar] [cond\_then] <stmts>

procedure <optional\_elseif\_a>

begin

case token of [cond\_elseif]:

match ([cond\_elseif]);

match ([lpar]);

call (<bool\_expr>);

match ([rpar]);

match ([cond\_then]);

call (<stmts>);

case token of empty:

ERROR();

end <optional\_elseif\_a>

<optional\_else> := [cond\_else] <stmts>

procedure <optional\_else>

begin

case token of [cond\_else]:

match ([cond\_else]);

call (<stmts>);

case token of empty:

ERROR();

end <optional\_else >

<switch\_a> := [switcher] [lpar] [identifier] [rpar] [lcurl] <case\_a> [rcurl]

procedure <switch\_a>

begin

case token of [switcher]:

match ([switcher]);

match ([lpar]);

match ([identifier]);

match ([rpar]);

match ([lcurl]);

call (<case\_a>);

match ([rcurl]);

case token of empty:

ERROR();

end <switch\_a>

<case\_a> := [caser] <constants> [colon] [lcurl] <case\_loob> [rcurl] <case\_a\_prime>

procedure <case\_a>

begin

case token of [caser]:

match ([caser]);

call (<constants>);

match ([colon]);

match ([lcurl]);

call (<case\_loob>);

match ([rcurl]);

call (<case\_a\_prime>);

case token of empty:

ERROR();

end <case\_a>

<case\_a\_prime> := <case\_a> | [defaulter] [colon] [lcurl] <stmts> [rcurl]

procedure <case\_a\_prime>

begin

case 1: if token of [defaulter]:

match ([defaulter]);

match ([colon]);

match ([lcurl]);

call (<stmts>);

match ([rcurl]);

case 2: if token == empty:

ERROR();

case 3:

call (<case\_a>);

end <case\_a\_prime>

<case\_loob> := <stmts> [breaker] [terminator]

procedure <case\_loob>

begin

call <stmts>;

case token of [breaker]:

match ([breaker]);

match ([terminator]);

case token of empty:

ERROR();

end <case\_loob >

<constants> : = [int\_const] | [bool\_const]| [dec\_const] |[string\_const]

procedure <constants>

begin

case token of [int\_const]:

match ([int\_const]);

case token of [bool\_const]:

match ([bool\_const]);

case token of [dec\_const]:

match ([dec\_const]);

case token of [string\_const]:

match ([string\_const]);

case token of empty:

ERROR();

end <constants>

<loop> := <loop\_a> <loop\_prime> | <loop\_b> <loop\_prime>

procedure <loop>

begin

case 1: call (<loop\_a>);

if token== last token from <loop\_a>{

call (<loop\_prime>);

}

case 2: call (<loop\_b>);

if token == last token from <loop\_b>{

call (<loop\_prime>);

}

case 3: token of empty:

ERROR();

end <loop>

<loop\_prime> := <loop> | ε

procedure <loop\_prime>

begin

if match(empty){

return;

}

else

call (<loop>);

end <loop\_prime>

<loop\_a> := [loop\_do] [lcurl] <stmts> [rcurl] [loop\_while] [lpar] <bool\_expr> [rpar] [terminator]

procedure <loop\_a>

begin

case token of [loop\_do]:

match ([loop\_do]);

match ([lcurl]);

call (<stmts>);

match ([rcurl]);

match ([loop\_while]);

match ([lpar]);

call (<bool\_expr>);

match ([rpar]);

match ([terminator]);

case token of empty:

ERROR();

end <loop\_a>

<loop\_b> := [loop\_while] [lpar] <bool\_expr> [rpar] [lcurl] <stmts> [rcurl]

procedure <loop\_b>

begin

case token of [loop\_while]:

match ([loop\_while]);

match ([lpar]);

call (<bool\_expr>);

match ([rpar]);

match ([lcur]);

call (<stmts>);

match ([rcurl]);

case token of empty:

ERROR();

end <loop\_b>

<comments> := [line\_comment] | [block\_comment]

procedure <comments>

begin

case token of [line\_comment]:

match ([line\_comment]);

case token of [block\_comment]:

match ([block\_comment]);

case token of empty:

ERROR();

end <comments>