ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Unit 4
BNF and Syntax Diagrams

Backus-Naur form and variants

- **Metasyntax**: a syntax used to describe the syntax of languages,
- **BNF** (Backus–Naur Form) is a metasyntax used to express context free grammars
- BNF is widely used as a notation for the grammars programming languages, instruction sets, communication protocols and parts of natural language grammars

Backus-Naur form and variants (cont)

- A set of rules is specified. These are known as **production** rules.
- Each production rule defines the pattern that represents a named structured part of the language
- The name of such a part is called a **non-terminal** symbol in the language.
- The basic elements of the language are called **terminal** symbols.

Backus-Naur form and variants (cont)

- Each rule contains the name of the non-terminal being defined, followed by the sequence or alternative sequences allowed for that symbol. A defining sequence can contain any terminal and non-terminal symbols allowed for that language.
- The definition of a rule can also contain the symbol being defined by that rules. This is called **recursive** definition.

Example: 2 grammars for natural numbers

- **Productions**

<nat> ::= <digit> | <digit><nat>

<digit> ::= “0”|”1”|”2”|”3”|”4”|”5”|”6”|”7”|”8”|”9”

- **Terminal symbols**
  - “0", “1, “2", …..“9"
- **Nonterminal symbols**
  - <nat>, <digit>
- **Start symbol**
  - <nat>

Example: Grammar for Arithmetic Expressions

- **Productions**

```
<Exp>    ::=
           "+"<Expr2>|"-"<Expr2>|<Expr2>
<Expr2>  ::= <Term><Expr3>
<Expr3>  ::= "+" <Term><Expr3>|
             "-"<Term><Expr3>|ε
<Term>   ::= <Factor> <Term2>
<Term2>  ::= "*" <Factor> <Term2> |"/"<Factor> <Term2>|ε
<Factor> ::="ident"|"number"|"("<Exp>")"
```

- **Terminal symbols**
  - simple TS: "+", "-", "*", "/", "(", ")"
  - terminal classes: "ident", "number"
- **Nonterminal symbols**
  - <Expr>, <Expr2>, <Expr3>, <Term>, <Term2>, <Factor>
- **Start symbol**
  - <Expr>

EBNF(Extended BNF)

- Terminal symbols start with lower-case letters
- Nonterminal symbols start with upper-case letters
- **Metasymbols**
  - | (...) separates alternatives groups
  - [...] alternatives optional part
  - {...} iterative part

  e.g.

KPL Grammar in BNF

```
01) <Prog> ::= KW_PROGRAM TK_IDENT SB_SEMICOLON <Block> SB_PERIOD

02) <Block>  ::= KW_CONST <ConstDecl> <ConstDecls> <Block2>
03) <Block>  ::= <Block2>

04) <Block2> ::= KW_TYPE <TypeDecl> <TypeDecls> <Block3>
05) <Block2> ::= <Block3>

06) <Block3> ::= KW_VAR <VarDecl> <VarDecls><Block4>
07) <Block3> ::= <Block4>

08) <Block4> ::= <SubDecls><Block5>
09) <Block4> ::=|<Block5>

10) <Block5> ::= KW_BEGIN <Statements> KW_END

11) <ConstDecls>::= <ConstDecl> <ConstDecls>
12) <ConstDecls>::= ε
13) <ConstDecl> ::= TK_IDENT SB_EQUAL <Constant> SB_SEMICOLON

14) <TypeDecls> ::= <TypeDecl> <TypeDecls>
15) <TypeDecls> ::= ε
16) <TypeDecl>  ::= TK_IDENT SB_EQUAL <Type> SB_SEMICOLON

17) <VarDecls>::= <VarDecl> <VarDecls>
18) <VarDecls>::= ε
19) <VarDecl>   ::= TK_IDENT SB_COLON <Type> SB_SEMICOLON
```

KPL Grammar in BNF

```
20) <SubDecls> ::= <FunDecl> <SubDecls>
21) <SubDecls> ::= <ProcDecl> <SubDecls>
22) <SubDecls> ::= ε

23) <FunDecl>   ::= KW_FUNCTION TK_IDENT <Params> SB_COLON <BasicType>
    SB_SEMICOLON
                    <Block> SB_SEMICOLON

24) <ProcDecl>  ::= KW_PROCEDURE TK_IDENT <Params> SB_SEMICOLON <Block>
    SB_SEMICOLON

25) <Params>    ::= SB_LPAR <Param> <Params2> SB_RPAR
26) <Params>    ::= ε

27) <Params2> ::= SB_SEMICOLON <Param> <Params2>
28) <Params2> ::= ε

29) <Param>   ::= TK_IDENT SB_COLON <BasicType>
30) <Param>   ::= KW_VAR TK_IDENT SB_COLON <BasicType>

31) <Type> ::= KW_INTEGER
32) <Type> ::= KW_CHAR
33) <Type> ::= TK_IDENT
34) <Type> ::= KW_ARRAY SB_LSEL TK_NUMBER SB_RSEL KW_OF <Type>
```

KPL Grammar in BNF

```
35) <BasicType> ::= KW_INTEGER
36) <BasicType> ::= KW_CHAR

37) <UnsignedConstant> ::= TK_NUMBER
38) <UnsignedConstant> ::= TK_IDENT
39) <UnsignedConstant> ::= TK_CHAR

40) <Constant> ::= SB_PLUS <Constant2>
41) <Constant> ::= SB_MINUS <Constant2>
42) <Constant> ::= <Constant2>
43) <Constant> ::= TK_CHAR

44) <Constant2>::= TK_IDENT
45) <Constant2>::= TK_NUMBER

46) <Statements> ::= <Statement> <Statements2>

47) <Statements2> ::= SB_SEMICOLON <Statement> <Statements2>
48) <Statements2> ::= ε
```

KPL Grammar in BNF

```
49) <Statement> ::= <AssignSt>
50) <Statement> ::= <CallSt>
51) <Statement> ::= <GroupSt>
52) <Statement> ::= <IfSt>
53) <Statement> ::= <WhileSt>
54) <Statement> ::= <ForSt>
55) <Statement> ::= ε

56) <AssignSt> ::= <Variable> SB_ASSIGN <Expression>
57) <AssignSt> ::= TK_IDENT SB_ASSIGN <Expression>

58) <CallSt>   ::= KW_CALL TK_IDENT <Arguments>

59) <GroupSt>  ::= KW_BEGIN <Statements> KW_END

60) <IfSt>     ::= KW_IF <Condition> KW_THEN <Statement> <ElseSt>

61) <ElseSt>   ::= KW_ELSE <Statement>
62) <ElseSt>   ::= ε

63) <WhileSt>  ::= KW_WHILE <Condition> KW_DO <Statement>
64) <ForSt>    ::= KW_FOR TK_IDENT SB_ASSIGN <Expression> KW_TO
                   <Expression> KW_DO <Statement>
```

KPL Grammar in BNF

```
65) <Arguments> ::= SB_LPAR <Expression> <Arguments2> SB_RPAR
66) <Arguments> ::= ε

67) <Arguments2>::= SB_COMMA <Expression> <Arguments2>
68) <Arguments2>::= ε

69) <Condition> ::= <Expression> <Condition2>

70) <Condition2>::= SB_EQ <Expression>
71) <Condition2>::= SB_NEQ <Expression>
72) <Condition2>::= SB_LE <Expression>
73) <Condition2>::= SB_LT <Expression>
74) <Condition2>::= SB_GE <Expression>
75) <Condition2>::= SB_GT <Expression>
```

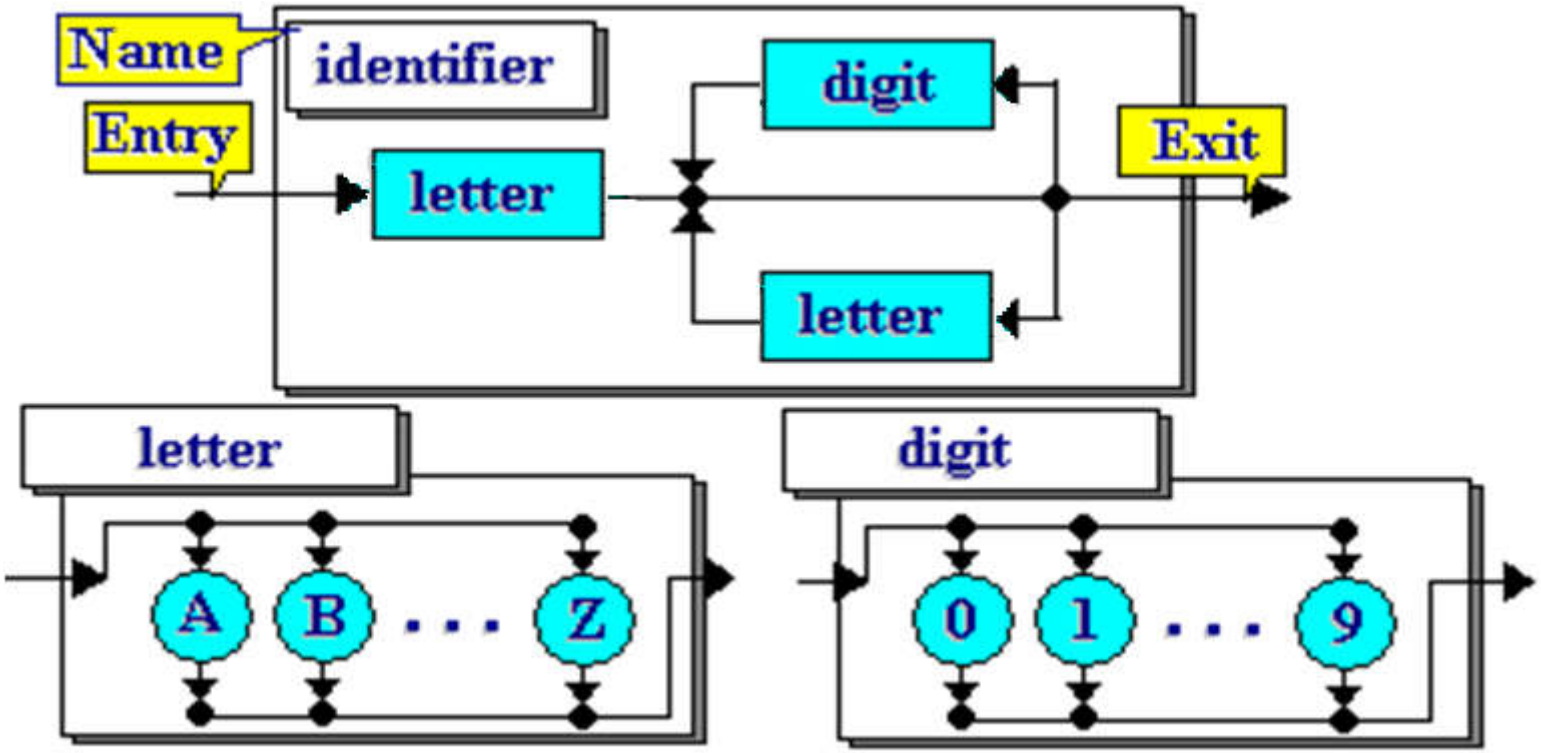
KPL Grammar in BNF

```
76) <Expression> ::= SB_PLUS <Expression2>
77) <Expression> ::= SB_MINUS <Expression2>
78) <Expression> ::= <Expression2>

79) <Expression2> ::= <Term> <Expression3>

80) <Expression3> ::= SB_PLUS <Term> <Expression3>
81) <Expression3> ::= SB_MINUS <Term> <Expression3>
82) <Expression3> ::= ε

83) <Term> ::= <Factor> <Term2>

84) <Term2> ::= SB_TIMES <Factor> <Term2>
85) <Term2> ::= SB_SLASH <Factor> <Term2>
86) <Term2> ::= ε
87) <Factor> ::= <UnsignedConstant>
88) <Factor> ::= <Variable>
89) <Factor> ::= <FunctionApptication>
90) <Factor> ::= SB_LPAR <Expression> SB_RPAR

91) <Variable> ::= TK_IDENT <Indexes>
92) <FunctionApplication> ::= TK_IDENT <Arguments>

93) <Indexes> ::= SB_LSEL <Expression> SB_RSEL <Indexes>
94) <Indexes> ::= ε
```

Syntax Diagram

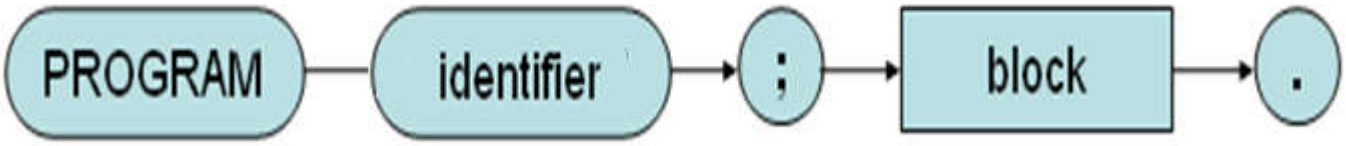
- Each diagram defines a non-terminal
- There is a main diagram which defines the language
- Each diagram has an entry point and an end point
- Terminals are represented by round boxes
- Nonterminals are represented by square boxes.

Examples of syntax diagram

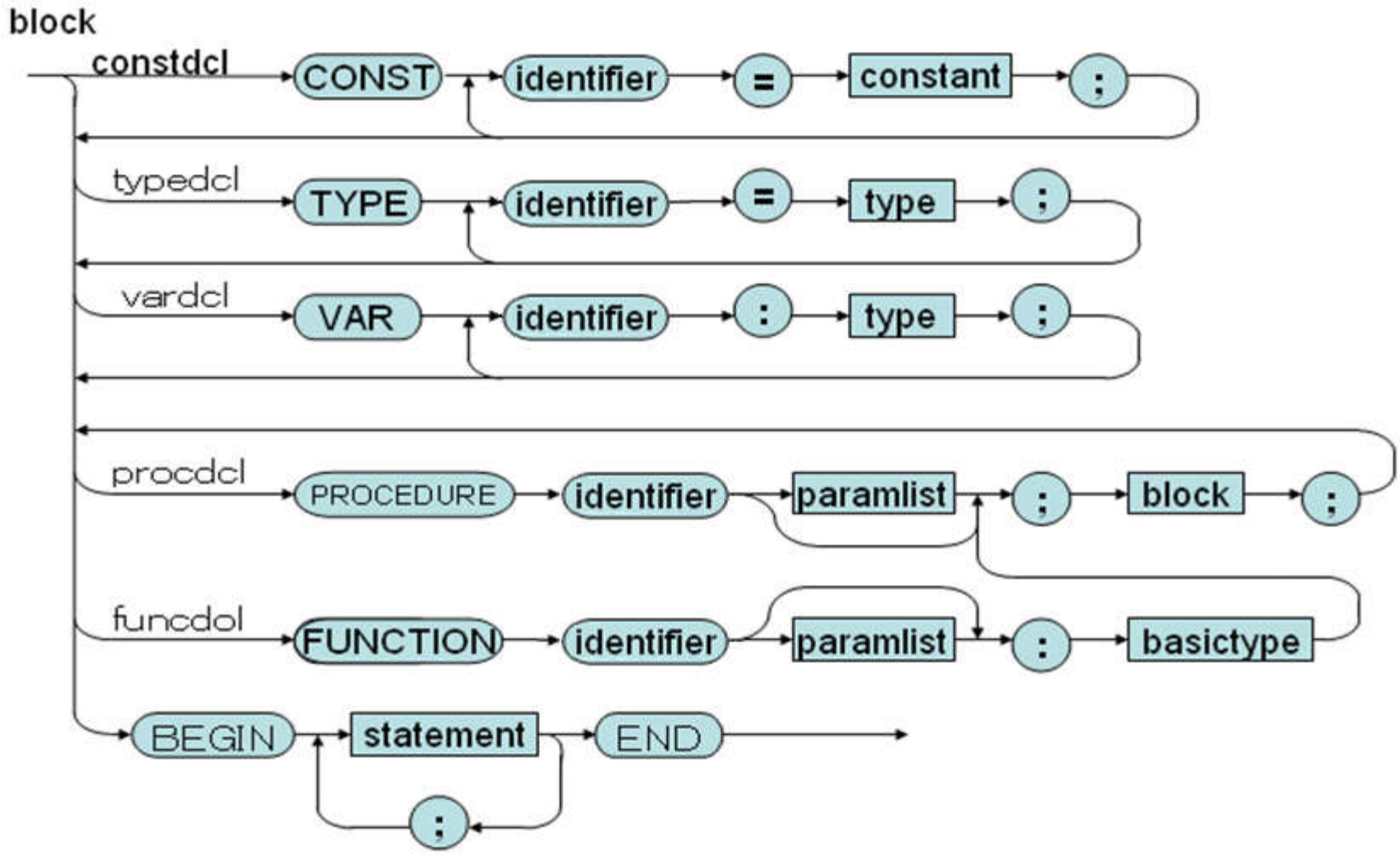


Syntax Diagrams of KPL (program)

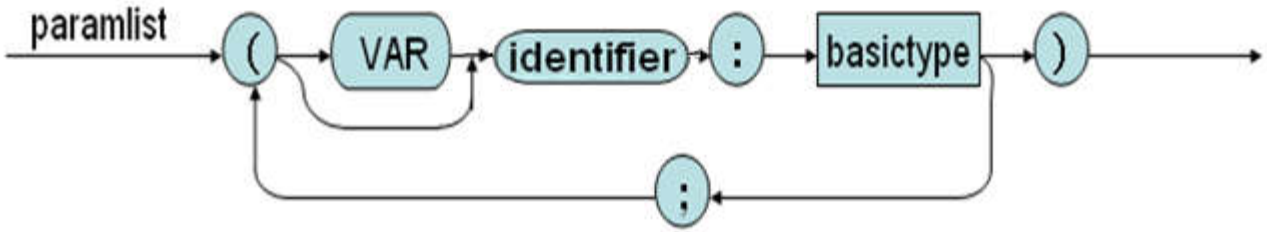
program

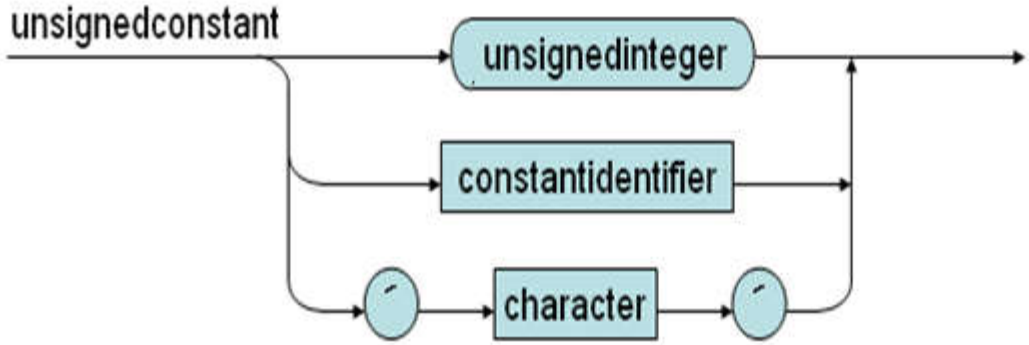


Syntax Diagrams of KPL(block)

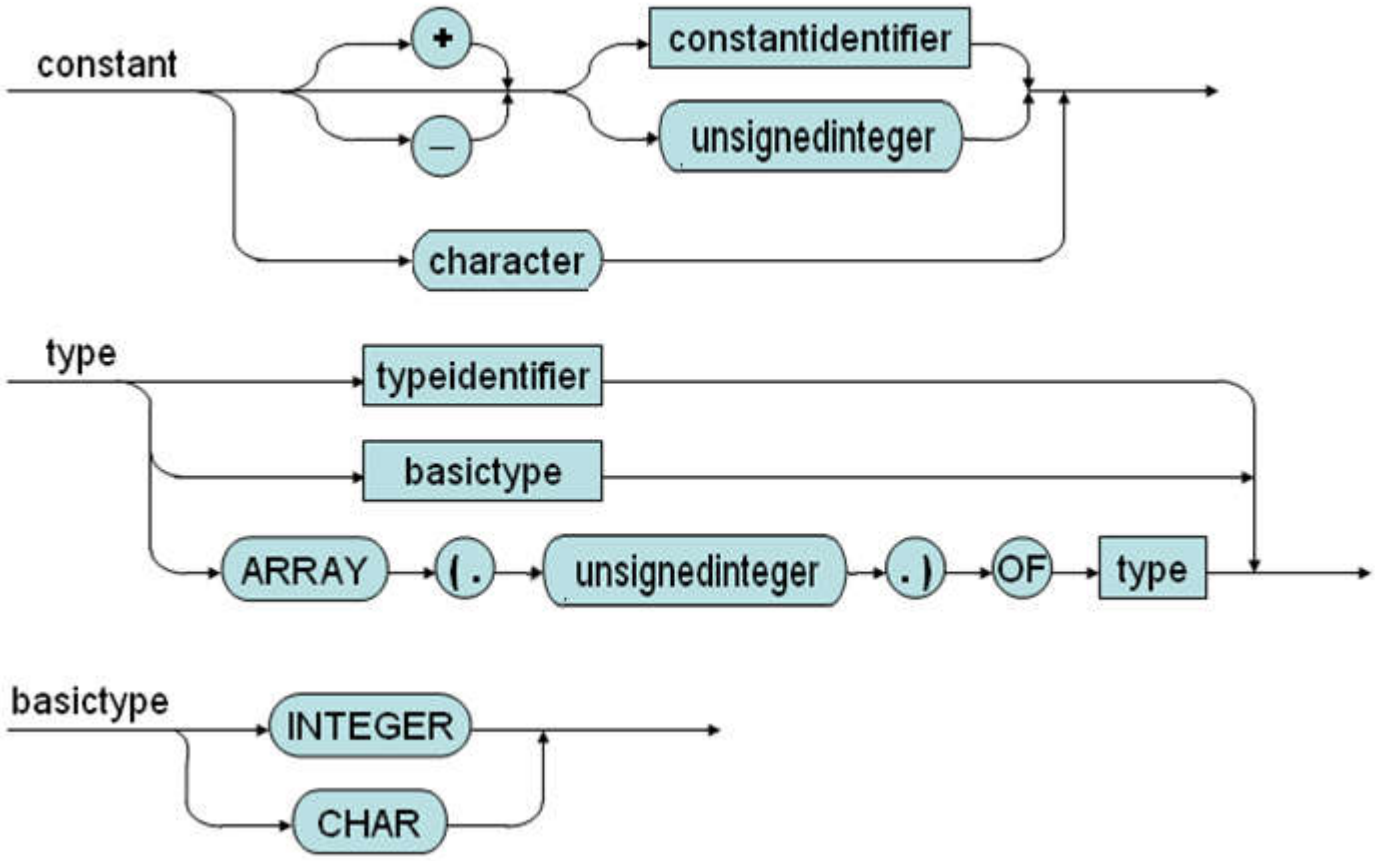


Syntax Diagrams of KPL (list of parameters, unsigned constant)

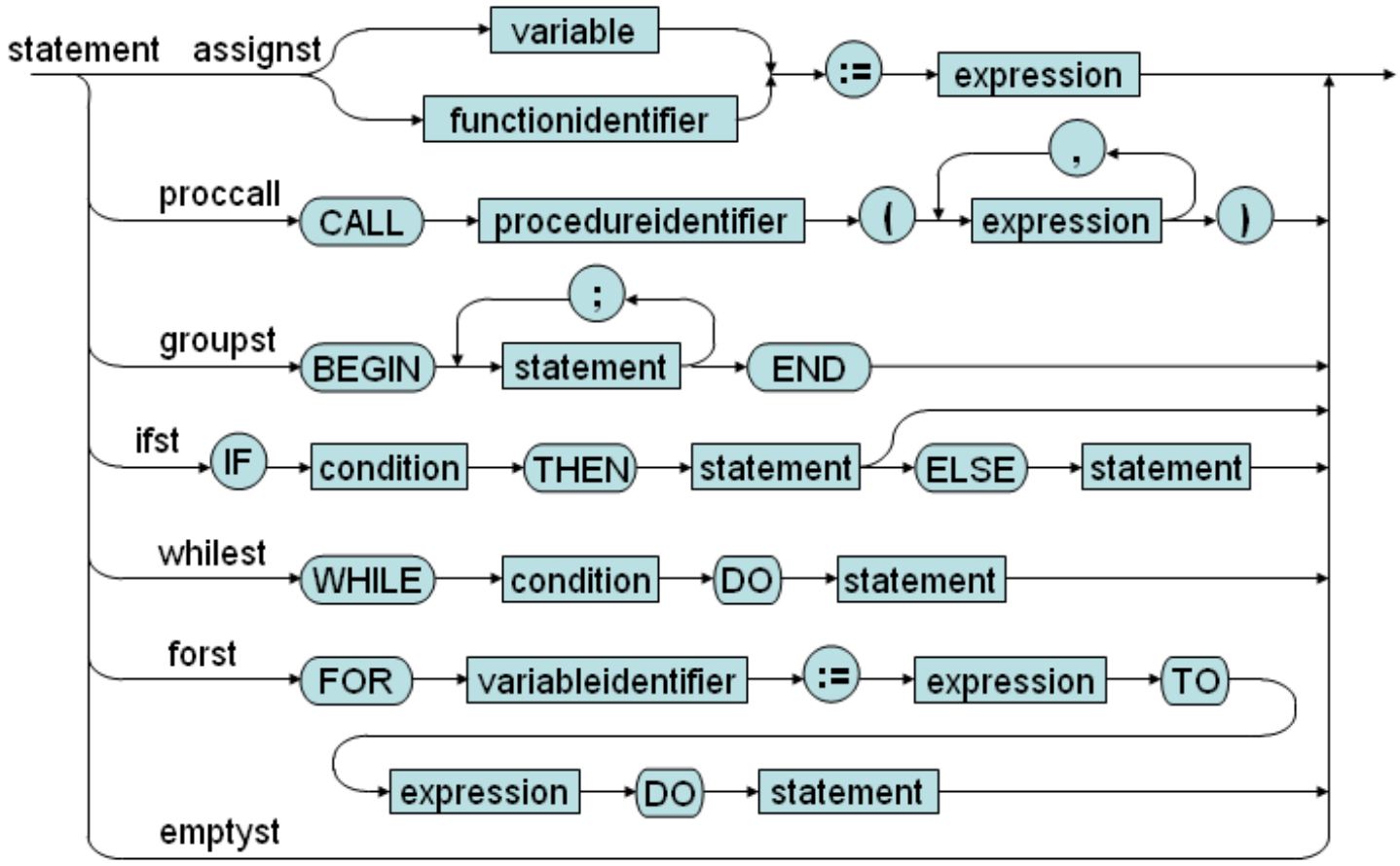




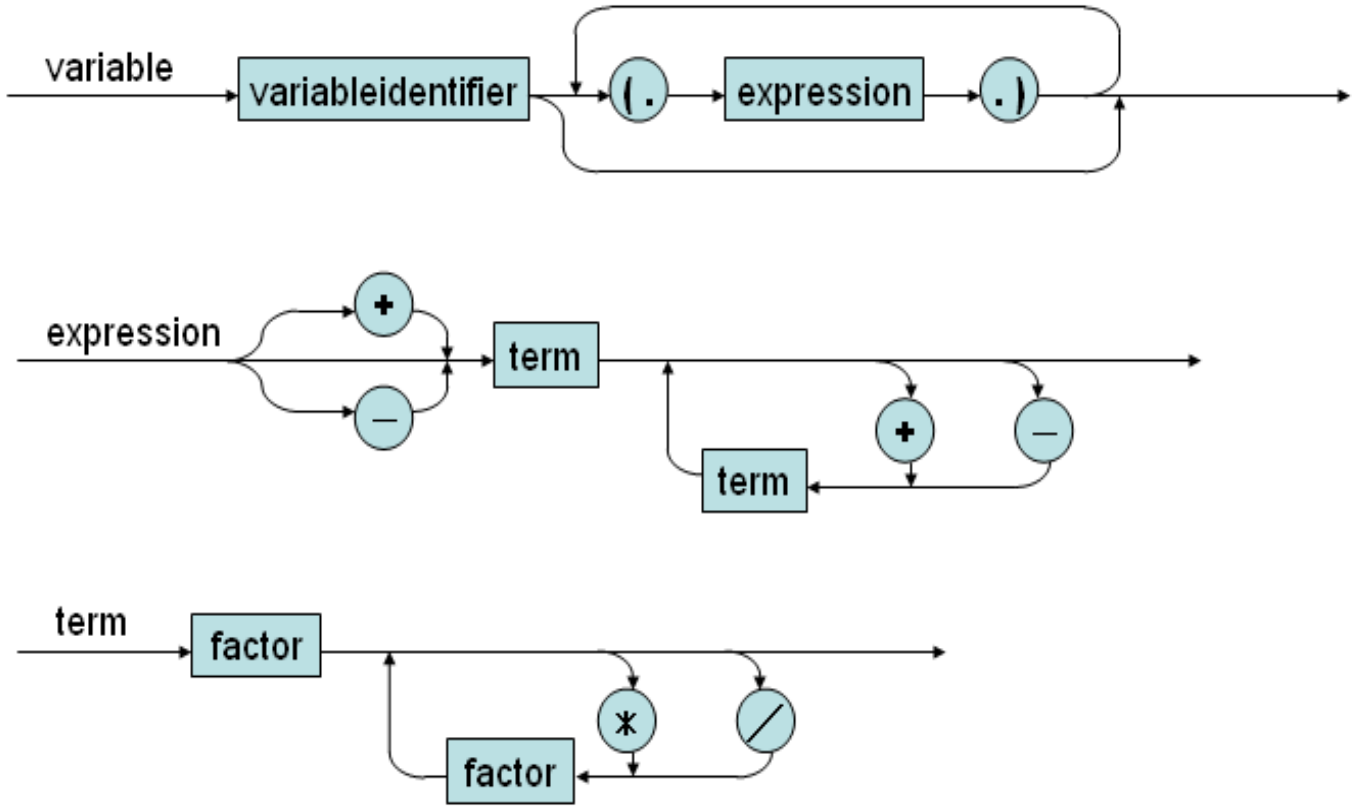
Syntax Diagrams of KPL (declarations)



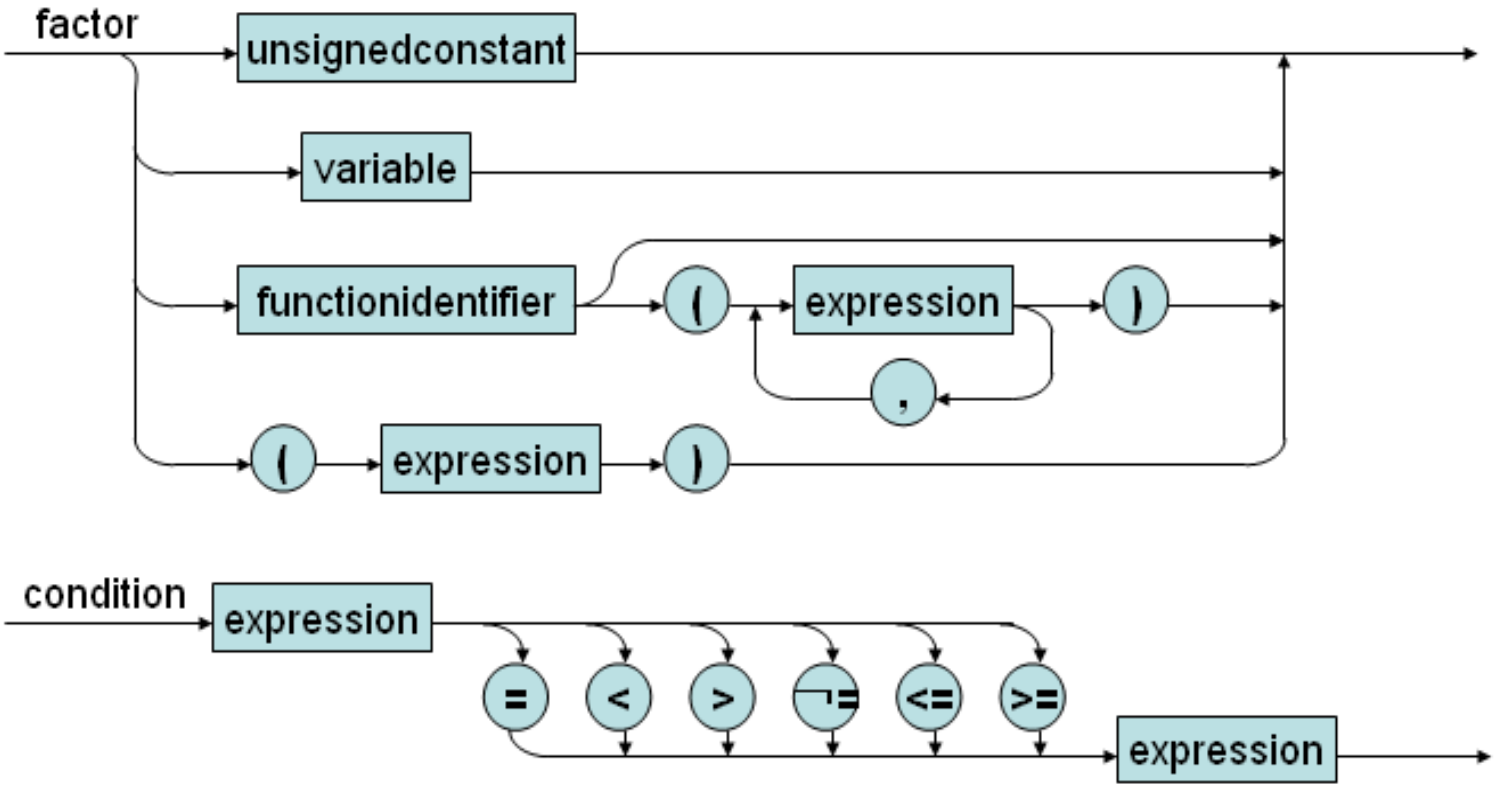
Syntax Diagrams of KPL (statement)



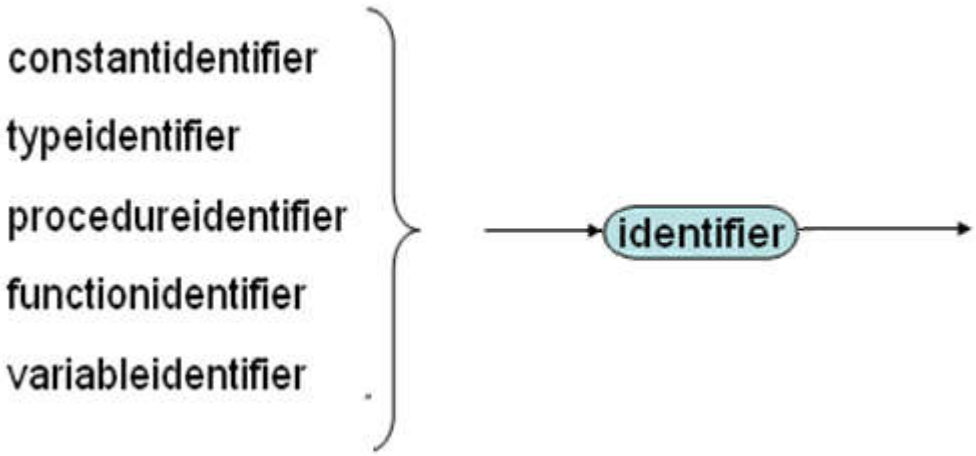
Syntax Diagrams of KPL (variable,expression,term)



Syntax Diagrams of KPL (factor,condition)



Syntax Diagrams of KPL (identifier,unsigned integer)



Exercise: a KPL program

Write a program that asks the user to type the value of an integer and compute its factorial.

Solution 1

```
program example1; (* Factorial *)
var n : integer; i: integer; f:integer;
BEGIN

n := readi;

f:=1;
if n >=2

begin

for i:= 2 to n do

f:= f*i;
call writeln;
call writeI(f);
end;
END. (* Factorial *)
```

Solution 2 (using KPL functions)

```
program example2; (* Factorial *)
var n : integer;
function f(k : integer) : integer;
  begin
    If k = 0 Then f := 1 Else f := k * f (k
- 1);
  end;

BEGIN
     n := readI;
     call writeln;
     call writeI(f(n));
    END. (* Factorial *)
```