

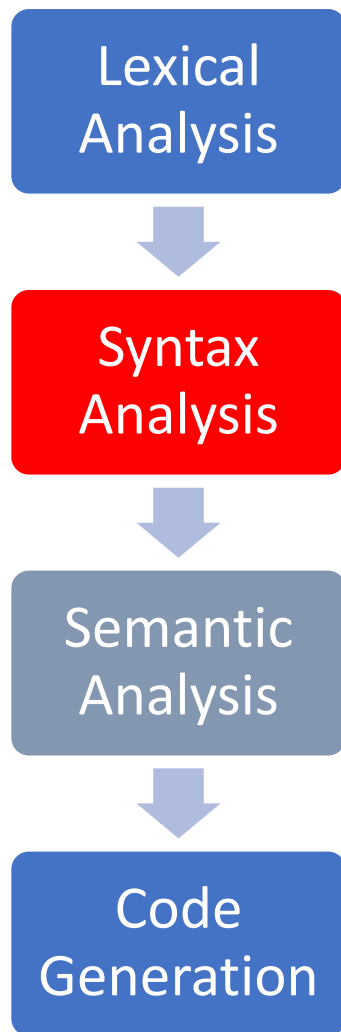
Thực hành  
Xây dựng chương trình dịch

## Bài 2. Phân tích cú pháp

ONE LOVE. ONE FUTURE.

- Tổng quan
- Văn phạm KPL
- Cài đặt bộ phân tích cú pháp

- Nhiệm vụ của bộ phân tích cú pháp



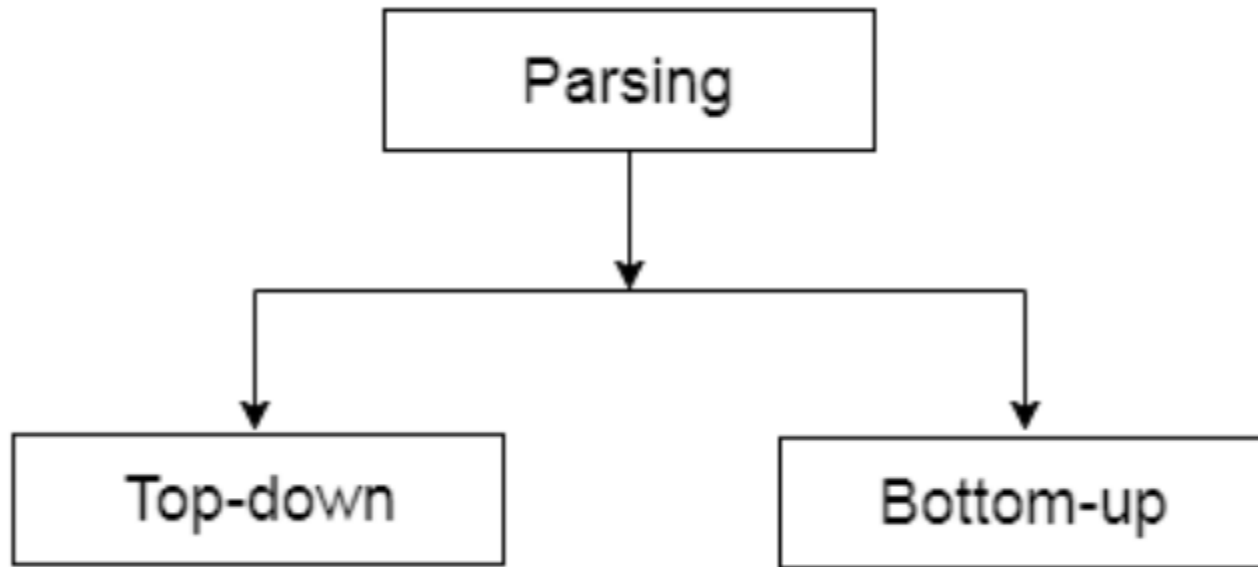
Kiểm tra cú pháp của chương trình nguồn

Cấu trúc cú pháp được biểu diễn bằng văn phạm hoặc sơ đồ cú pháp

Phục vụ cho các pha sau

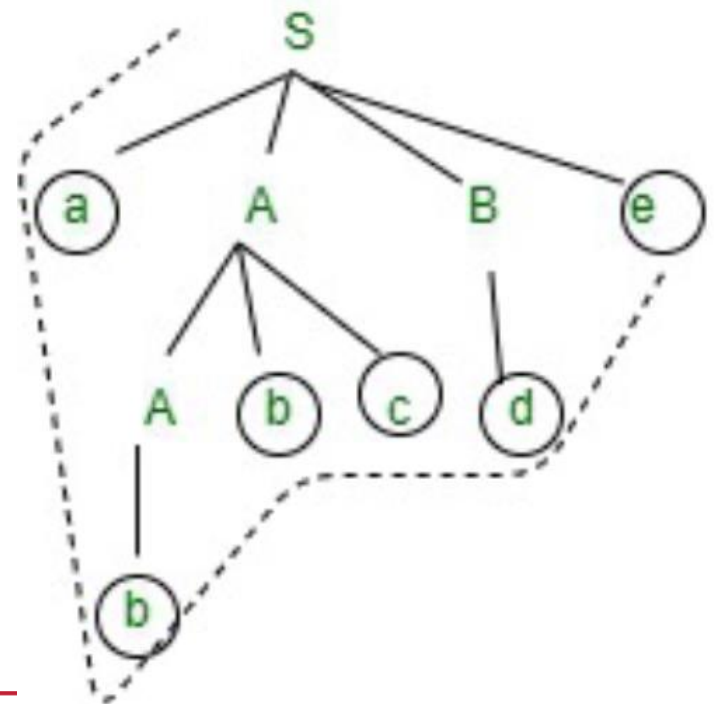
Compiler làm việc theo tiếp cận tự cú pháp nên bộ cú pháp rất quan trọng, quyết định tốc độ của compiler

# Phân loại các bộ PTCP của NN lập trình



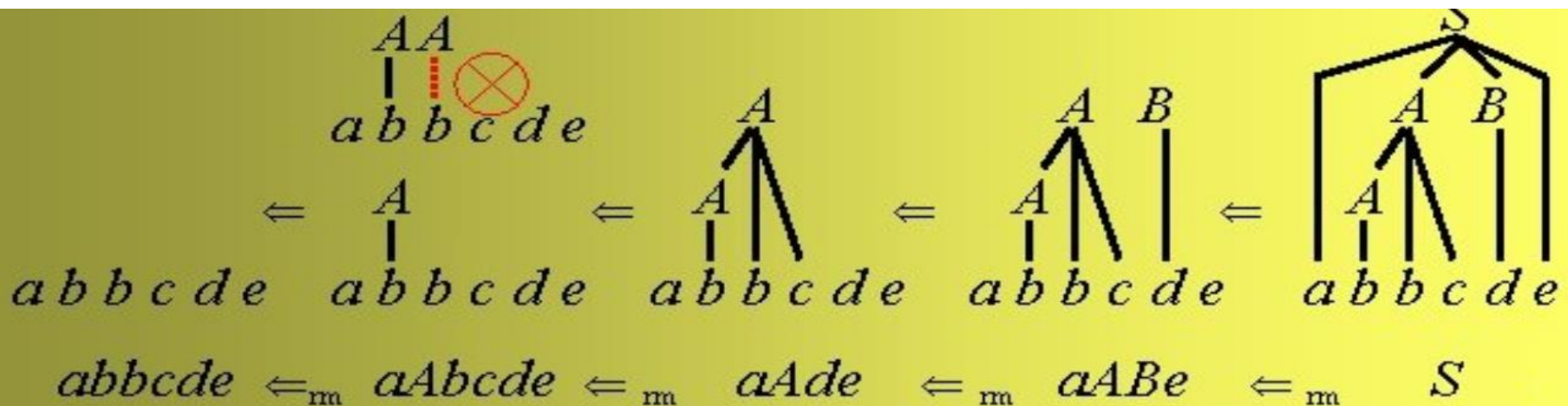
# Phân tích trên xuống

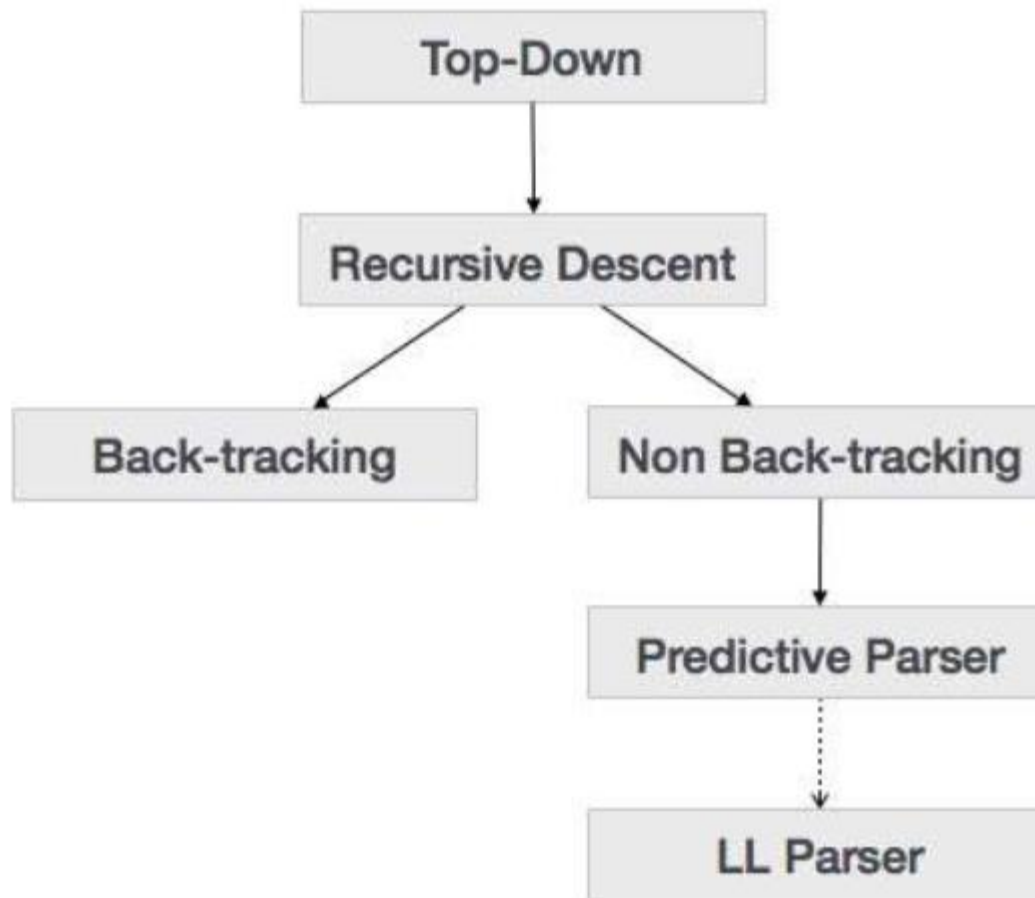
- Xây dựng cây phân tích cú pháp từ gốc xuống lá, đọc chương trình nguồn từ trái qua phải
- Việc xây dựng cây dựa vào suy dẫn trái
- Nếu một vế trái có nhiều hơn 1 vế phải, chọn vế phải nào cho các nút cấp dưới?
- Ví dụ: Văn phạm G với các sản xuất
  - G: (1)  $S \rightarrow a A B e$
  - (2, 3)  $A \rightarrow A b c | b$
  - (4)  $B \rightarrow d$
- Xâu vào: abbcde



# Phân tích cú pháp từ dưới lên

- Xây dựng cây phân tích cú pháp từ lá lên gốc, đọc chương trình nguồn từ trái qua phải
- Sử dụng suy dẫn phải It follows the rightmost derivation
- Ví dụ: Cho văn phạm G với các sản xuất:
  - G: (1)  $S \rightarrow a A B e$   
 $A \rightarrow A b c | b$   
 $B \rightarrow d$
  - Xâu vào:  $abbcde$





- Là phương pháp phân tích từ trên xuống, tức là vẽ cây từ gốc xuống lá
- Sử dụng các thủ tục có thể đệ quy, mỗi thủ tục cho một ký hiệu kết thúc
  - Đầu tiên gọi thủ tục phân tích ký hiệu đầu
  - Khi một ký hiệu không kết thúc có nhiều vế phải, mỗi vế phải là một nhánh trong thủ tục
  - Sử dụng ký hiệu xem trước lookAhead để quyết định vế phải được chọn.



- Với mỗi luật (sản xuất) có dạng

$\langle \text{phrase1} \rangle \rightarrow E$

bộ phân tích cú pháp có một hàm để phân tích luật E

```
void compilePhrase1( )  
{ /* phân tích luật E */ }
```

- E là một dãy có thể chứa ký hiệu kết thúc và không kết thúc
- Văn phạm cần **không đệ quy trái**

# Phân tích một luật (một vế phải)

- Một vế phải có thể gồm một dãy ký hiệu  $Y_1 Y_2 Y_3 \dots Y_n$  phải xem xét từng ký hiệu

- Mỗi ký hiệu không kết thúc tương ứng một hàm **compileY**

- Mỗi ký hiệu kết thúc tương ứng với lời gọi hàm **eat(y)**

để kiểm tra xem y có phải là ký hiệu kết thúc tiếp theo được sinh ra bằng cách áp dụng các luật cú pháp hay không

- Ký hiệu kết thúc chính là từ tố do bộ scanner đưa ra
- Nếu biến lookAhead chứa ký hiệu tiếp theo đúng với văn phạm thì chuyển xét từ tố mới, ngược lại báo lỗi

```
eat(y) : //giả ngữ
        if (lookAhead == y)
        then getNextToken()
        else SyntaxError()
```

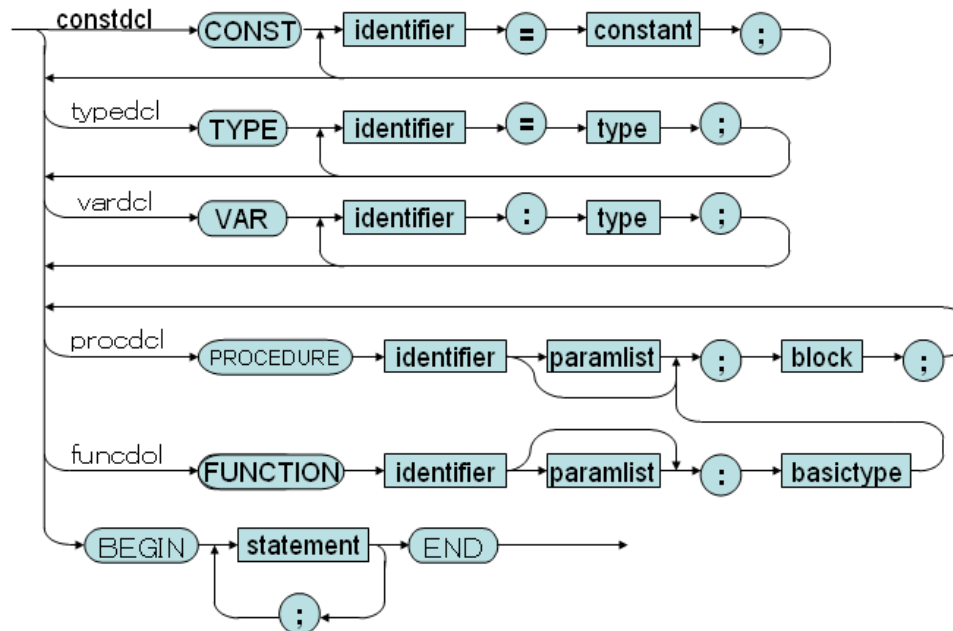


# Sơ đồ cú pháp của KPL

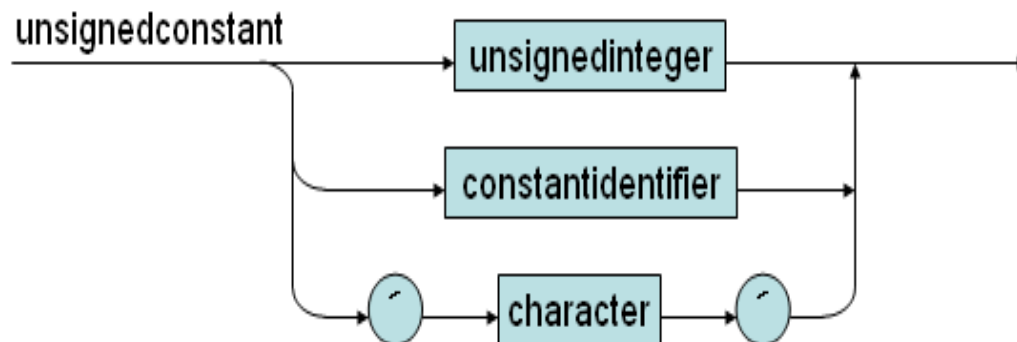
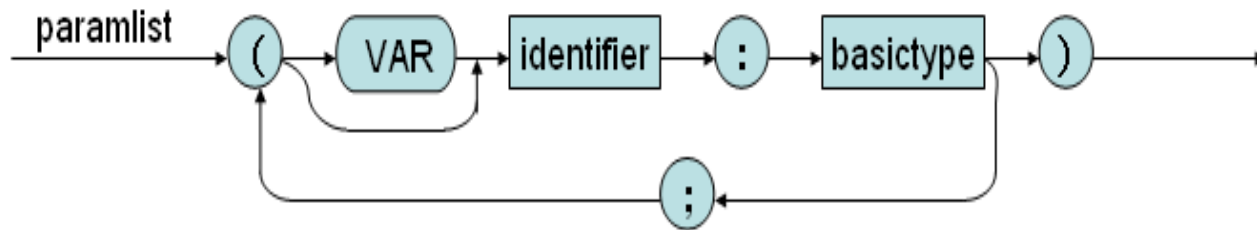
program



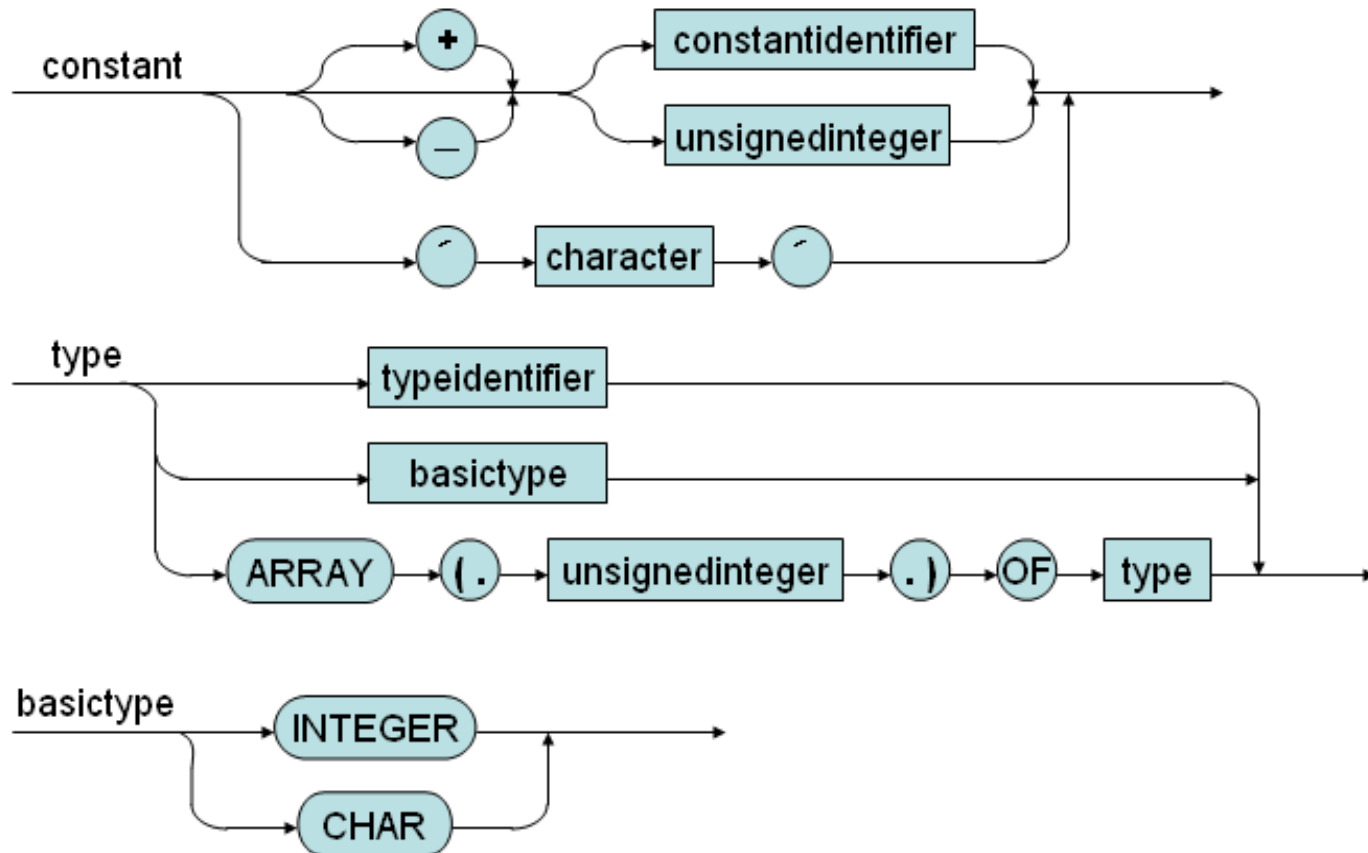
block



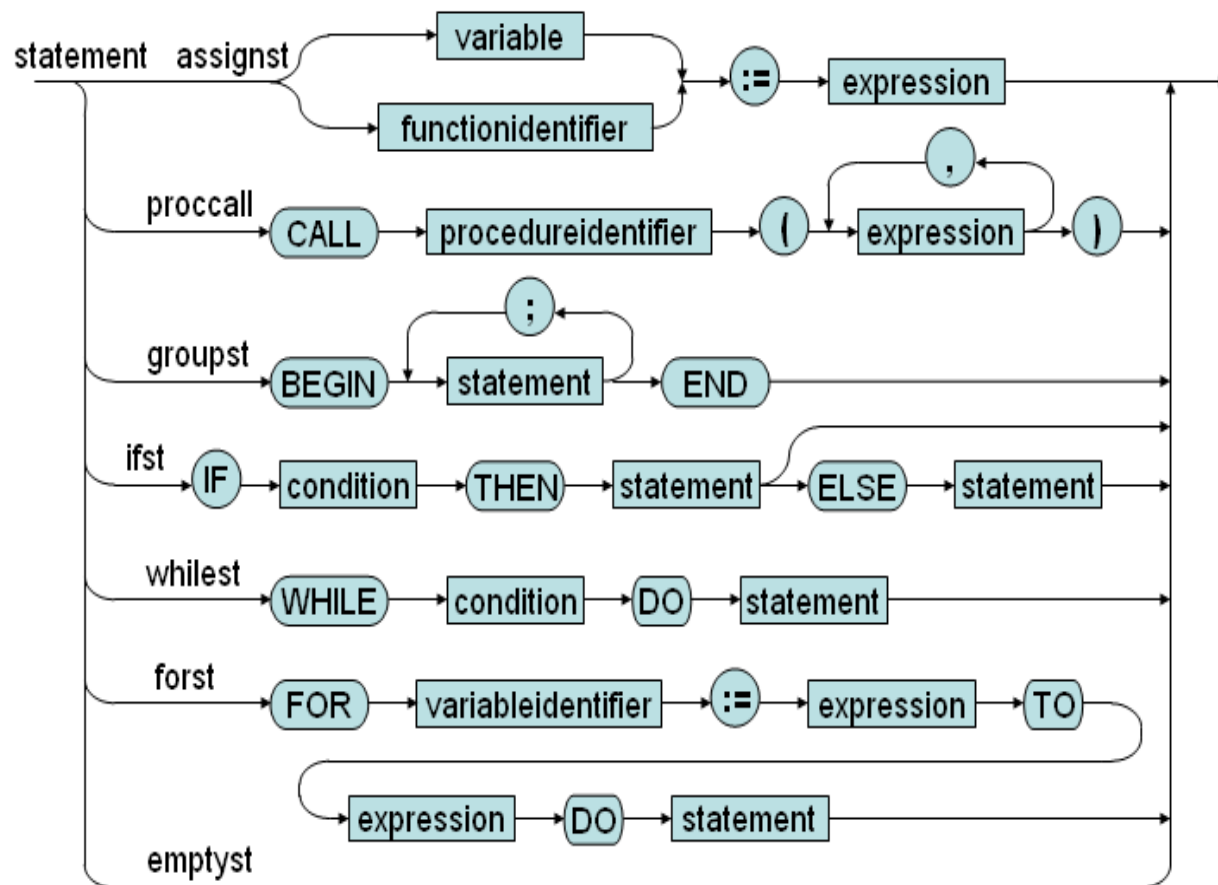
# Sơ đồ cú pháp của KPL



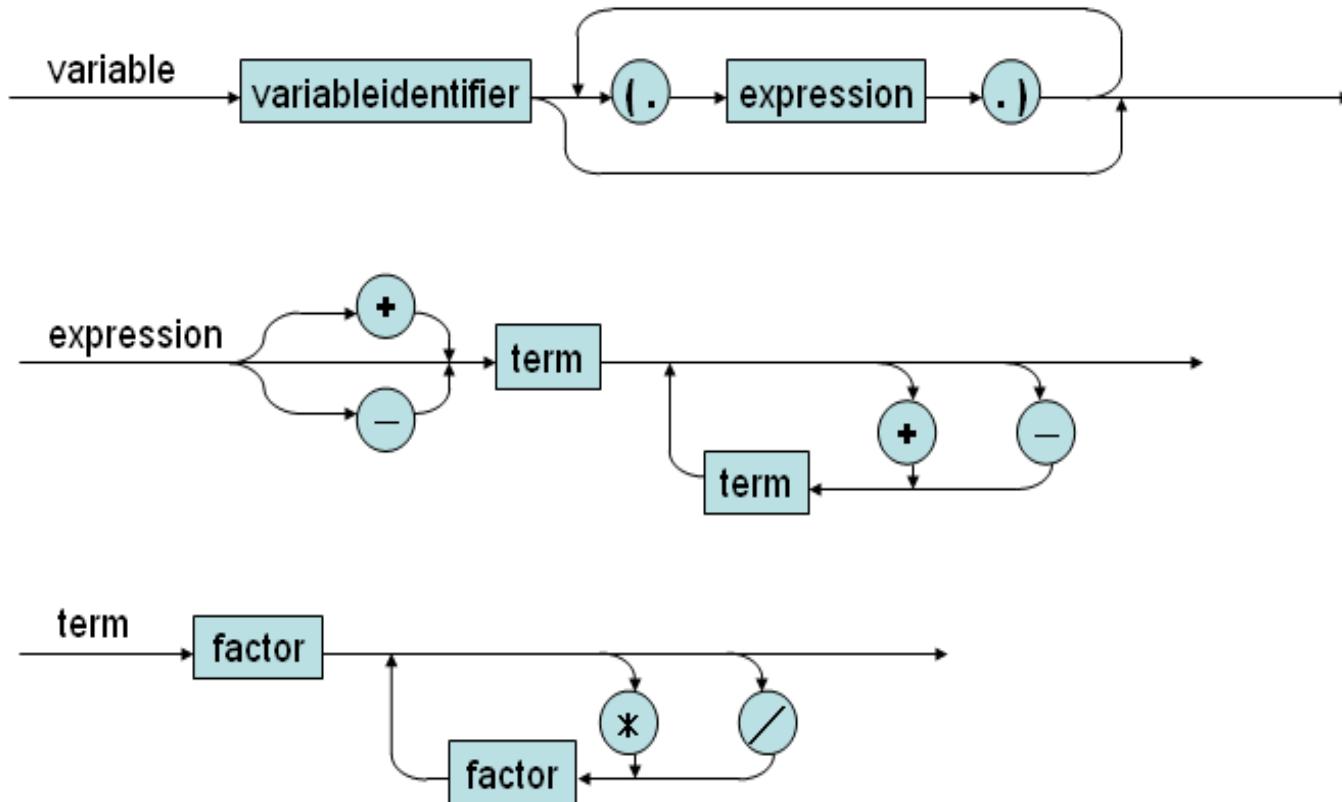
# Sơ đồ cú pháp của KPL



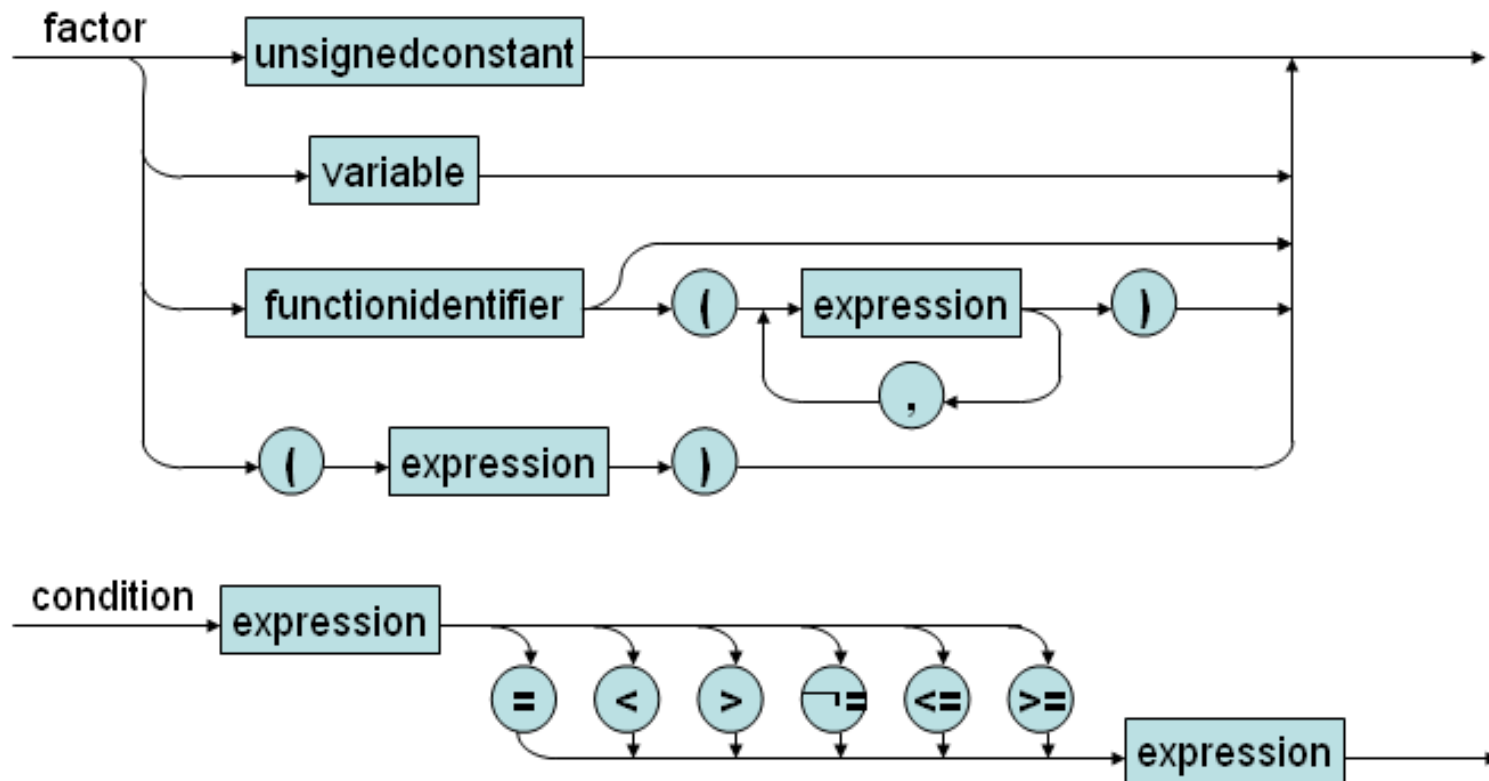
# Sơ đồ cú pháp của KPL



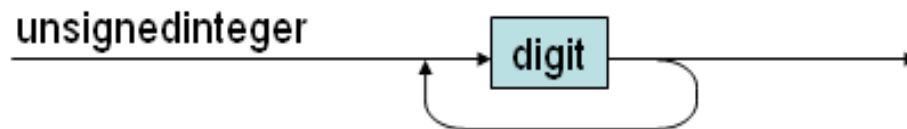
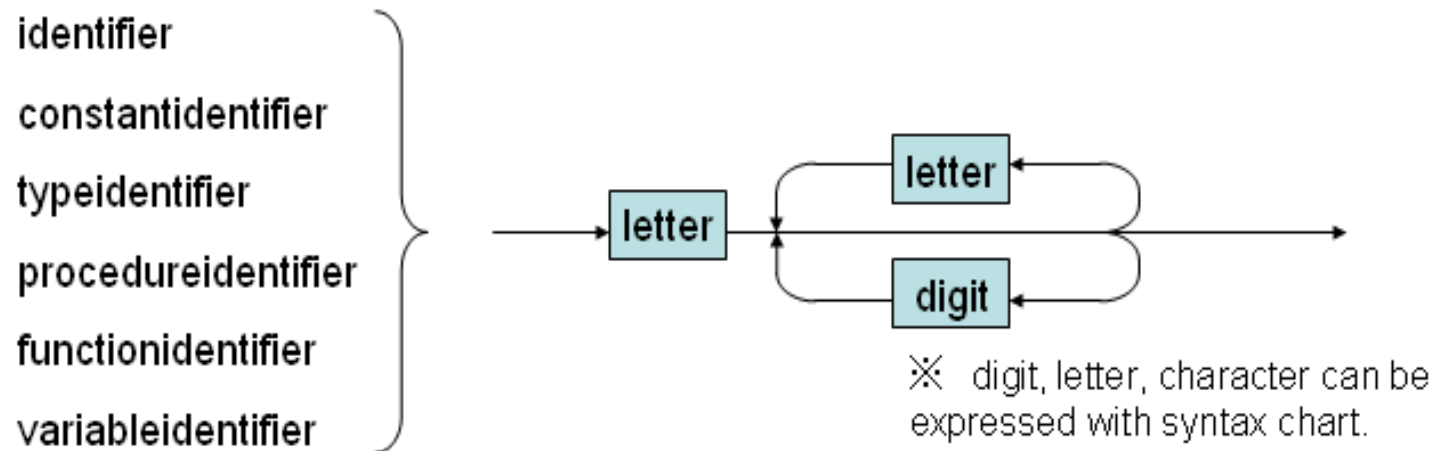
# Sơ đồ cú pháp của KPL



# Sơ đồ cú pháp của KPL







- Chuyển từ tập sơ đồ cú pháp sang văn phạm
- Khử đệ quy trái (đã thực hiện)
- Nhân tử trái (đã thực hiện)

- 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)  $\langle \text{ConstDecls} \rangle ::= \langle \text{ConstDecl} \rangle \langle \text{ConstDecls} \rangle$
- 12)  $\langle \text{ConstDecls} \rangle ::= \varepsilon$
- 13)  $\langle \text{ConstDecl} \rangle ::= \text{TK\_IDENT SB\_EQUAL} \langle \text{Constant} \rangle \text{SB\_SEMICOLON}$
- 14)  $\langle \text{TypeDecls} \rangle ::= \langle \text{TypeDecl} \rangle \langle \text{TypeDecls} \rangle$
- 15)  $\langle \text{TypeDecls} \rangle ::= \varepsilon$
- 16)  $\langle \text{TypeDecl} \rangle ::= \text{TK\_IDENT SB\_EQUAL} \langle \text{Type} \rangle \text{SB\_SEMICOLON}$
- 17)  $\langle \text{VarDecls} \rangle ::= \langle \text{VarDecl} \rangle \langle \text{VarDecls} \rangle$
- 18)  $\langle \text{VarDecls} \rangle ::= \varepsilon$
- 19)  $\langle \text{VarDecl} \rangle ::= \text{TK\_IDENT SB\_COLON} \langle \text{Type} \rangle \text{SB\_SEMICOLON}$
- 20)  $\langle \text{SubDecls} \rangle ::= \langle \text{FunDecl} \rangle \langle \text{SubDecls} \rangle$
- 21)  $\langle \text{SubDecls} \rangle ::= \langle \text{ProcDecl} \rangle \langle \text{SubDecls} \rangle$
- 22)  $\langle \text{SubDecls} \rangle ::= \varepsilon$

```

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>`
  
- 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)  $\langle \text{Statements} \rangle ::= \langle \text{Statement} \rangle \langle \text{Statements2} \rangle$

47)  $\langle \text{Statements2} \rangle ::= \text{SB\_SEMICOLON} \langle \text{Statement} \rangle \langle \text{Statements2} \rangle$

48)  $\langle \text{Statements2} \rangle ::= \varepsilon$

49)  $\langle \text{Statement} \rangle ::= \langle \text{AssignSt} \rangle$

50)  $\langle \text{Statement} \rangle ::= \langle \text{CallSt} \rangle$

51)  $\langle \text{Statement} \rangle ::= \langle \text{GroupSt} \rangle$

52)  $\langle \text{Statement} \rangle ::= \langle \text{IfSt} \rangle$

53)  $\langle \text{Statement} \rangle ::= \langle \text{WhileSt} \rangle$

54)  $\langle \text{Statement} \rangle ::= \langle \text{ForSt} \rangle$

55)  $\langle \text{Statement} \rangle ::= \varepsilon$

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



- 65)  $\langle \text{Arguments} \rangle ::= \text{SB\_LPAR } \langle \text{Expression} \rangle \langle \text{Arguments2} \rangle \text{SB\_RPAR}$
- 66)  $\langle \text{Arguments} \rangle ::= \varepsilon$
- 67)  $\langle \text{Arguments2} \rangle ::= \text{SB\_COMMA } \langle \text{Expression} \rangle \langle \text{Arguments2} \rangle$
- 68)  $\langle \text{Arguments2} \rangle ::= \varepsilon$
- 69)  $\langle \text{Condition} \rangle ::= \langle \text{Expression} \rangle \langle \text{Condition2} \rangle$
- 70)  $\langle \text{Condition2} \rangle ::= \text{SB\_EQ } \langle \text{Expression} \rangle$
- 71)  $\langle \text{Condition2} \rangle ::= \text{SB\_NEQ } \langle \text{Expression} \rangle$
- 72)  $\langle \text{Condition2} \rangle ::= \text{SB\_LE } \langle \text{Expression} \rangle$
- 73)  $\langle \text{Condition2} \rangle ::= \text{SB\_LT } \langle \text{Expression} \rangle$
- 74)  $\langle \text{Condition2} \rangle ::= \text{SB\_GE } \langle \text{Expression} \rangle$
- 75)  $\langle \text{Condition2} \rangle ::= \text{SB\_GT } \langle \text{Expression} \rangle$

- 76)  $\langle \text{Expression} \rangle ::= \text{SB\_PLUS } \langle \text{Expression2} \rangle$
- 77)  $\langle \text{Expression} \rangle ::= \text{SB\_MINUS } \langle \text{Expression2} \rangle$
- 78)  $\langle \text{Expression} \rangle ::= \langle \text{Expression2} \rangle$
  
- 79)  $\langle \text{Expression2} \rangle ::= \langle \text{Term} \rangle \langle \text{Expression3} \rangle$
  
- 80)  $\langle \text{Expression3} \rangle ::= \text{SB\_PLUS } \langle \text{Term} \rangle \langle \text{Expression3} \rangle$
- 81)  $\langle \text{Expression3} \rangle ::= \text{SB\_MINUS } \langle \text{Term} \rangle \langle \text{Expression3} \rangle$
- 82)  $\langle \text{Expression3} \rangle ::= \varepsilon$
  
- 83)  $\langle \text{Term} \rangle ::= \langle \text{Factor} \rangle \langle \text{Term2} \rangle$
  
- 84)  $\langle \text{Term2} \rangle ::= \text{SB\_TIMES } \langle \text{Factor} \rangle \langle \text{Term2} \rangle$
- 85)  $\langle \text{Term2} \rangle ::= \text{SB\_SLASH } \langle \text{Factor} \rangle \langle \text{Term2} \rangle$
- 86)  $\langle \text{Term2} \rangle ::= \varepsilon$
- 87)  $\langle \text{Factor} \rangle ::= \langle \text{UnsignedConstant} \rangle$
- 88)  $\langle \text{Factor} \rangle ::= \langle \text{Variable} \rangle$
- 89)  $\langle \text{Factor} \rangle ::= \langle \text{FunctionApptication} \rangle$
- 90)  $\langle \text{Factor} \rangle ::= \text{SB\_LPAR } \langle \text{Expression} \rangle \text{SB\_RPAR}$

91)  $\langle \text{Variable} \rangle ::= \text{TK\_IDENT } \langle \text{Indexes} \rangle$

92)  $\langle \text{FunctionApplication} \rangle ::= \text{TK\_IDENT } \langle \text{Arguments} \rangle$

93)  $\langle \text{Indexes} \rangle ::= \text{SB\_LSEL } \langle \text{Expression} \rangle \text{ SB\_RSEL } \langle \text{Indexes} \rangle$

94)  $\langle \text{Indexes} \rangle ::= \varepsilon$

- Vào: Dùng các hàm
  - ReadI: Đọc một số nguyên. Không tham số
  - ReadC: Đọc một ký tự. Không tham sốVí dụ  
var a: integer;  
a:= ReadI;
- Ra: Dùng thủ tục
  - Writel: In một số nguyên. 1 tham số
  - WriteC: In một ký tự. 1 tham số
  - WriteLn: In dấu xuống dòng.Ví dụ  
call Writel(a);  
call WriteLn;

- Viết hàm tính bình phương của một số nguyên.
- Viết một chương trình tính tổng bình phương của n số tự nhiên đầu tiên. N đọc từ bàn phím.

```
program example5;  
var n : integer; i: integer; sum: integer;  
  
function f(k : integer) : integer;  
begin  
    f := k * k;  
end;  
  
BEGIN  
    n := readI;  
    sum := 0;  
    for i:=1 to n do  
        sum:= sum + f(i);  
    call writeln;  
    call writeI(sum);  
END. (* example*)
```

# Cài đặt bộ phân tích cú pháp KPL

- Nói chung văn phạm KPL là LL(1)
- Thiết kế bộ phân tích cú pháp đệ quy trên xuống
  - *lookAhead*
    - Phân tích ký hiệu kết thúc
    - Phân tích ký hiệu không kết thúc
      - Xây dựng bảng phân tích
      - Tính các tập FIRST() và FOLLOW()

- Example

02) Block ::= KW\_CONST ConstDecl ConstDecls Block2  
=>RHS1

03) Block ::= Block2  
=>RHS2

FIRST(RHS1)={KW\_CONST}

FIRST(RHS2)={KW\_TYPE, KW\_VAR, KW\_FUNCTION,  
KW\_PROCEDURE, KW\_BEGIN}

$FIRST(RHS1) \cap FIRST(RHS2) = \emptyset$

LookAhead =KW\_BEGIN =>RHS2 được chọn =>LL(1)

- Đọc từ tổ tiếp sau trên chương trình nguồn

```
Token *currentToken;  
Token *lookAhead;  
  
void scan(void) {  
    Token* tmp = currentToken;  
    currentToken = lookAhead;  
    lookAhead = getValidToken();  
    free(tmp);  
}
```



```
void eat(TokenType tokenType) {  
    if (lookAhead->tokenType == tokenType) {  
        printToken(lookAhead);  
        scan();  
    } else  
        missingToken(tokenType, lookAhead->lineNo, lookAhead->colNo);  
}
```

```
int compile(char *fileName) {  
    if (openInputStream(fileName) == IO_ERROR)  
        return IO_ERROR;  
  
    currentToken = NULL;  
    lookAhead = getValidToken();  
  
    compileProgram();  
  
    free(currentToken);  
    free(lookAhead);  
    closeInputStream();  
    return IO_SUCCESS;  
}
```

# Phân tích một ký hiệu KKT, có 1 vế phải

Ví dụ: **Program**

1)  $\langle \text{Prog} \rangle ::= \text{KW\_PROGRAM TK\_IDENT SB\_SEMICOLON } \langle \text{Block} \rangle \text{ SB\_PERIOD}$

```
void compileProgram(void) {  
    assert("Parsing a Program ....");  
    eat(KW_PROGRAM);  
    eat(TK_IDENT);  
    eat(SB_SEMICOLON);  
    compileBlock();  
    eat(SB_PERIOD);  
    assert("Program parsed!");  
}
```

## Luật cho Basic Type

35) `<BasicType> ::= KW_INTEGER`

36) `<BasicType> ::= KW_CHAR`

```
void compileBasicType(void) {  
    switch (lookAhead->tokenType) {  
        case KW_INTEGER:  
            eat(KW_INTEGER);  
            break;  
        case KW_CHAR:  
            eat(KW_CHAR);  
            break;  
        default:  
            error(ERR_INVALIDBASICTYPE, lookAhead->lineNo,  
lookAhead->colNo);  
            break;  
    }  
}
```

# Phân tích statement

Statement: 7 về phải trong đó có 1 về phải là xâu rỗng

FIRST(<Statement>) = {TK\_IDENT, KW\_CALL, KW\_BEGIN, KW\_IF, KW\_WHILE,  
KW\_FOR,  $\epsilon$ }

FOLLOW(<Statement>) = {SB\_SEMICOLON, KW\_END, KW\_ELSE}

/\* Predict parse table for Expression \*/

Input	Production
-----	
TK_IDENT	49) <Statement> ::= <AssignSt>
KW_CALL	50) <Statement> ::= <CallSt>
KW_BEGIN	51) <Statement> ::= <GroupSt>
KW_IF	52) <Statement> ::= <IfSt>
KW_WHILE	53) <Statement> ::= <Whilst?
KW_FOR	54) <Statement> ::= <ForSt?
-----	
SB_SEMICOLON	55) $\epsilon$
KW_END	55) $\epsilon$
KW_ELSE	55) $\epsilon$
-----	
Khác	Error

# Hàm phân tích statement

```
void compileStatement(void)
{
    switch (lookAhead-
>tokenType) {
        case TK_IDENT:
            compileAssignSt();
            break;
        case KW_CALL:
            compileCallSt();
            break;
        case KW_BEGIN:
            compileGroupSt();
            break;
        case KW_IF:
            compileIfSt();
            break;
        case KW_WHILE:
            compileWhileSt();
            break;
```

```
        case KW_FOR:
            compileForSt();
            break;
            // check FOLLOW tokens
        case SB_SEMICOLON:
        case KW_END:
        case KW_ELSE:
            break;
            // Error occurs
        default:
            error(ERR_INVALIDSTATEMENT,
lookAhead->lineNo,
lookAhead->colNo);
            break;
    }
}
```

Có thể dùng đệ quy và tính FOLLOW. Tuy nhiên có thể viết ngắn gọn hơn khi biết đây thực chất là một chu trình trong sơ đồ cú pháp

10)  $\langle \text{ConstDecls} \rangle ::= \langle \text{ConstDecl} \rangle \langle \text{ConstDecls} \rangle$

11)  $\langle \text{ConstDecls} \rangle ::= \varepsilon$

```
void compileConstDecls(void) {  
    while (lookAhead->tokenType == TK_IDENT)  
        compileConstDecl();  
}
```

# Có thể kết hợp sơ đồ cú pháp

Cú pháp của Term (dùng BNF)

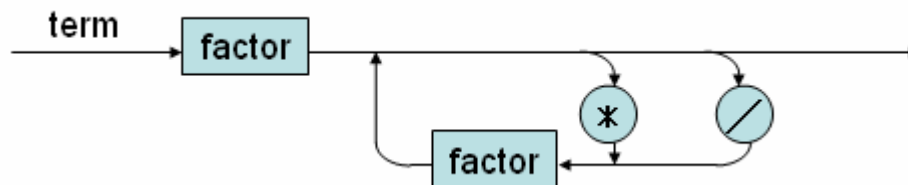
83)  $\langle \text{Term} \rangle ::= \langle \text{Factor} \rangle \langle \text{Term2} \rangle$

84)  $\langle \text{Term2} \rangle ::= \text{SB\_TIMES} \langle \text{Factor} \rangle \langle \text{Term2} \rangle$

85)  $\langle \text{Term2} \rangle ::= \text{SB\_SLASH} \langle \text{Factor} \rangle \langle \text{Term2} \rangle$

86)  $\langle \text{Term2} \rangle ::= \varepsilon$

Cú pháp của Term (theo sơ đồ cú pháp)





# Xử lý term và term 2 theo luật, tính Follow

```
void compileTerm(void)
{ compileFactor();
  compileTerm2();
}
```

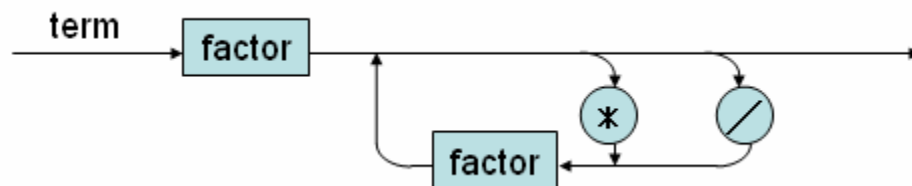
```
void compileTerm2(void) {
  switch (lookAhead->tokenType) {
    case SB_TIMES:
      eat(SB_TIMES);
      compileFactor();
      compileTerm2();
      break;
    case SB_SLASH:
      eat(SB_SLASH);
      compileFactor();
      compileTerm2();
      break;
    // check the FOLLOW set
    case SB_PLUS:
    case SB_MINUS:
    case KW_TO:
    case KW_DO:
```

```
    case SB_RPAR:
      case SB_COMMA:
      case SB_EQ:
      case SB_NEQ:
      case SB_LE:
      case SB_LT:
      case SB_GE:
      case SB_GT:
      case SB_RSEL:
      case SB_SEMICOLON:
      case KW_END:
      case KW_ELSE:
      case KW_THEN:
        break;
      default:
        error(ERR_INVALIDTERM, lookAhead-
          >lineNo, lookAhead->colNo);
    }
  }
}
```

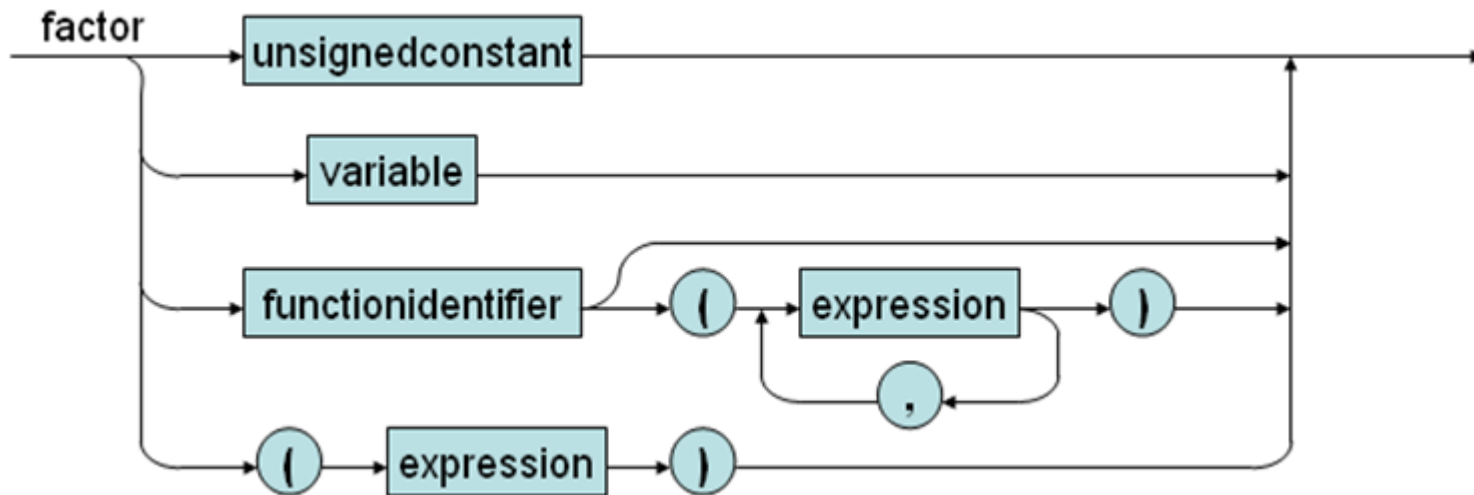


# Xử lý term theo sơ đồ cú pháp

```
void compileTerm(void)
{
    compileFactor();
    while(lookAhead->tokenType== SB_TIMES || lookAhead->tokenType ==
        SB_SLASH)
    {
        switch (lookAhead->tokenType)
        {
            case SB_TIMES:
                eat(SB_TIMES);
                compileFactor();
                break;
            case SB_SLASH:
                eat(SB_SLASH);
                compileFactor();
                break;
        }
    }
}
```



## Factor: Vi phạm điều kiện LL(1)



$\text{FIRST}(\text{unsignedconstant}) = \{\text{TK\_NUMBER}, \text{TK\_IDENT}, \text{TK\_CHAR}\}$

$\text{FIRST}(\text{variable}) = \{\text{TK\_IDENT}\}$

$\text{FIRST}(\text{functioncall}) = \{\text{TK\_IDENT}\}$

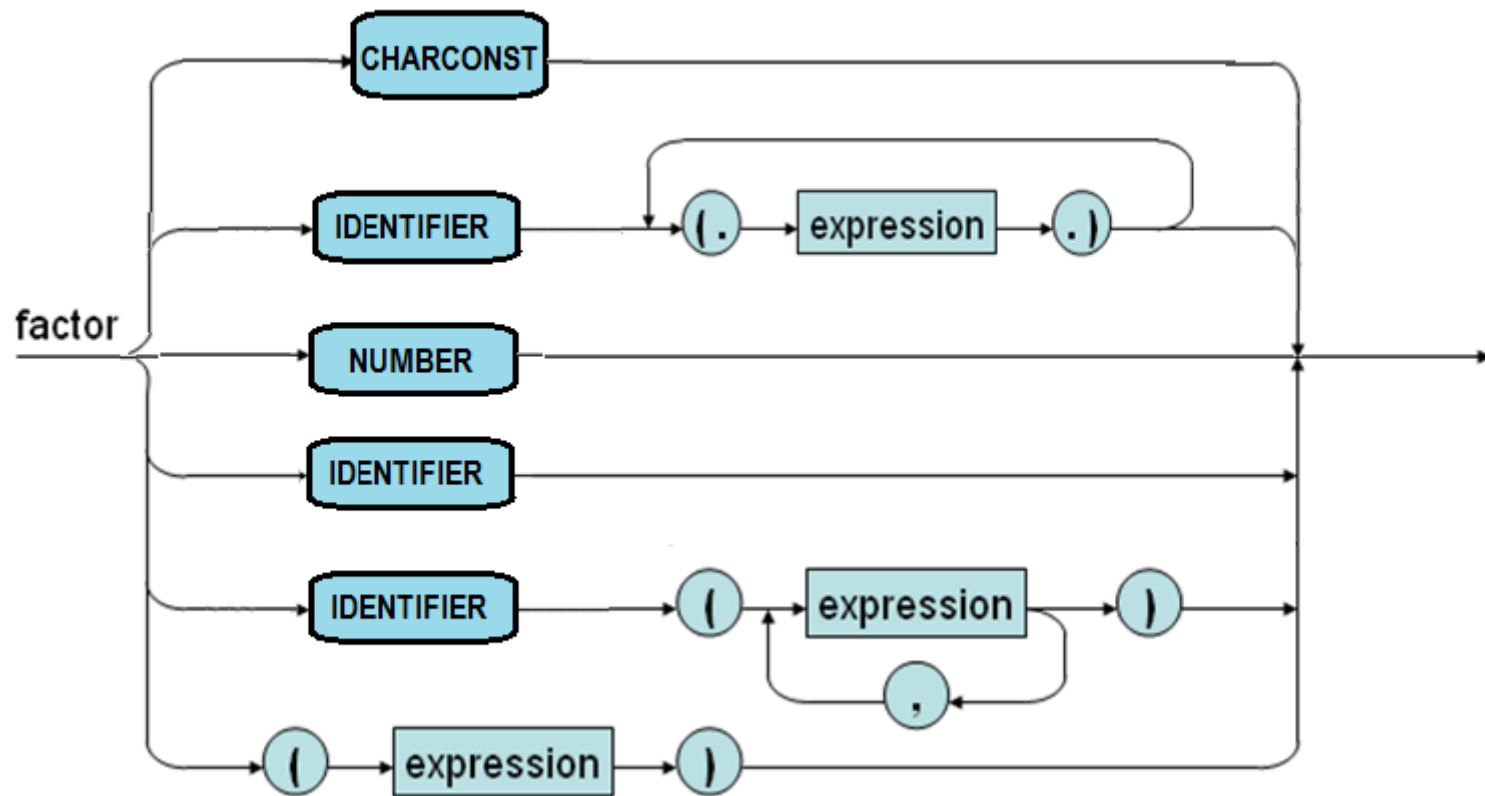
$\text{FIRST}(\text{unsignedconstant}) \cap \text{FIRST}(\text{functioncall}) = \{\text{TK\_IDENT}\}$

$\text{FIRST}(\text{variable}) \cap \text{FIRST}(\text{functioncall}) = \{\text{TK\_IDENT}\}$

$\text{FIRST}(\text{variable}) \cap \text{FIRST}(\text{unsignedconstant}) = \{\text{TK\_IDENT}\}$

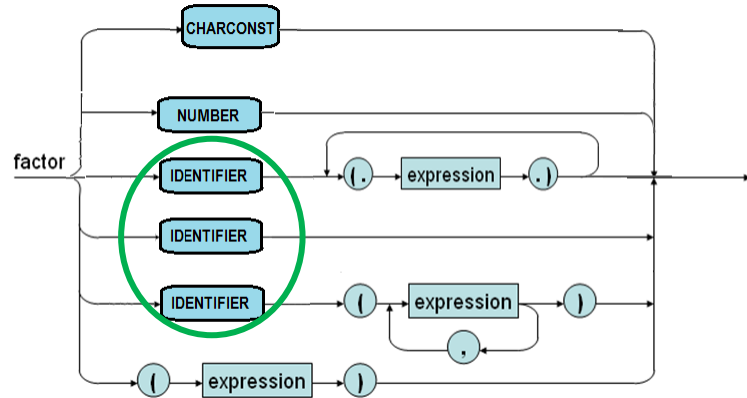
=>Vi phạm điều kiện LL(1)

## Sau khi sắp xếp lại



# Phân tích factor

```
void compileFactor(void) {  
    switch (lookAhead->tokenType) {  
    case TK_NUMBER:  
        eat(TK_NUMBER);  
        break;  
    case TK_CHAR:  
        eat(TK_CHAR);  
        break;  
    case TK_IDENT:  
        eat(TK_IDENT);  
        switch (lookAhead->tokenType) {  
        case SB_LSEL:  
            compileIndexes();  
            break;  
        case SB_LPAR:  
            compileArguments();  
            break;  
        default: break;  
        }  
        break;  
    }
```



```
    case SB_LPAR:  
        eat(SB_LPAR);  
        compileExpression();  
        eat(SB_RPAR);  
        break;  
    default:  
        error(ERR_INVALIDFACTOR,  
            lookAhead->lineNo, lookAhead->colNo);  
    }  
}
```