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

Bài 9.
Phương pháp đệ quy trên xuống

Đặc điểm của phương pháp

- Sử dụng để phân tích cú pháp cho các văn phạm LL(1)
- Có thể mở rộng cho văn phạm LL(k), nhưng việc tính toán phức tạp
- Sử dụng để phân tích văn phạm khác có thể dẫn đến lặp vô hạn

Bộ phân tích cú pháp

- Bao gồm một tập thủ tục, mỗi thủ tục ứng với một sơ đồ cú pháp (một ký hiệu không kết thúc)
- Các thủ tục đệ quy : khi triển khai một ký hiệu không kết thúc có thể gặp các ký hiệu không kết thúc khác, dẫn đến các thủ tục gọi lẫn nhau, và có thể gọi trực tiếp hoặc gián tiếp đến chính nó.

Mô tả chức năng

- Giả sử mỗi thủ tục hướng tới một đích ứng với một sơ đồ cú pháp
- Tại mỗi thời điểm luôn có một đích được triển khai, kiểm tra cú pháp hết một đoạn nào đó trong văn bản nguồn

Thủ tục triển khai một đích

- Đối chiếu văn bản nguồn với một đường trên sơ đồ cú pháp
- Đọc từ tố tiếp
- Đối chiếu với nút tiếp theo trên sơ đồ
  - Nếu là nút tròn (ký hiệu kết thúc)thì từ tố vừa đọc phải phù hợp với từ tố trong nút
  - Nếu là nút chữ nhật nhãn A (ký hiệu không kết thúc), từ tố vừa đọc phải thuộc FIRST (A) => tiếp tục triển khai đích A
- Ngược lại, thông báo một lỗi cú pháp tại điểm đang xét

Từ sơ đồ thành thủ tục

- Mỗi sơ đồ ứng với một thủ tục
- Các nút xuất hiện tuần tự chuyển thành các câu lệnh kế tiếp nhau.
- Các điểm rẽ nhánh chuyển thành câu lệnh lựa chọn (if, case)
- Chu trình chuyển thành câu lệnh lặp (while, do while, repeat. . .)
- Nút tròn chuyển thành đoạn đối chiếu từ tố
- Nút chữ nhật chuyển thành lời gọi tới thủ tục khác

Chú ý

- Bộ phân tích cú pháp luôn đọc trước một từ tố
- Xem trước một từ tố cho phép chọn đúng đường đi khi gặp điểm rẽ nhánh trên sơ đồ cú pháp
- Khi thoát khỏi thủ tục triển khai một đích, có một từ tố đã được đọc dôi ra

Bộ phân tích cú pháp KPL

- void error(ErrorCode err, int lineNo, int colNo)
- void eat(TokenType tokenType) (kiểm tra từ tố hiện hành có thuộc loại được chỉ ra không?
- Các hàm phân tích cú pháp ứng với các sản xuất hoặc sơ đồ cú pháp

  - void compileFactor(void);//phân tích nhân tử
  - void compileTerm(void);//phân tích số hạng
  - void compileExpression(void); // phân tích biểu thức
  - void CompileCondition(void); // phân tích điều kiện
  - void CompileStatement(void); // phân tích câu lệnh
  - void compileBlock(void); // phân tích các khối câu lệnh
  - void compileBasictype(void); // các kiểu biến cơ bản
  - void compileProgram();

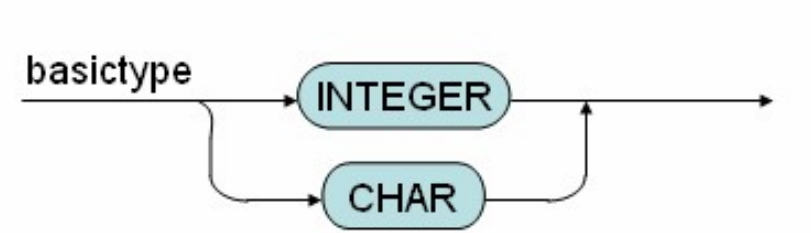
Hàm eat

So sánh k/h đỉnh stack và k/h đang xét (xem trước)

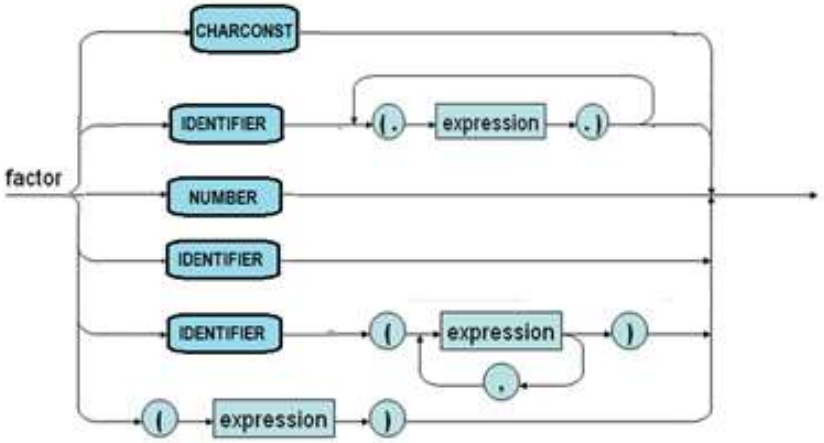
```
void eat(TokenType tokenType) {
  if (lookAhead->tokenType == tokenType) {
    printToken(lookAhead);
    scan();
  } else missingToken(tokenType, lookAhead->lineNo, lookAhead->colNo);
}
```

Phân tích basic type

```
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;
  }
}
```



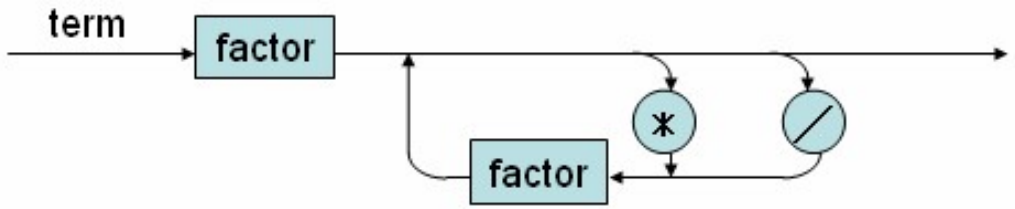
Phân tích factor




```
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);
  }
}
```

Phân tích term

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

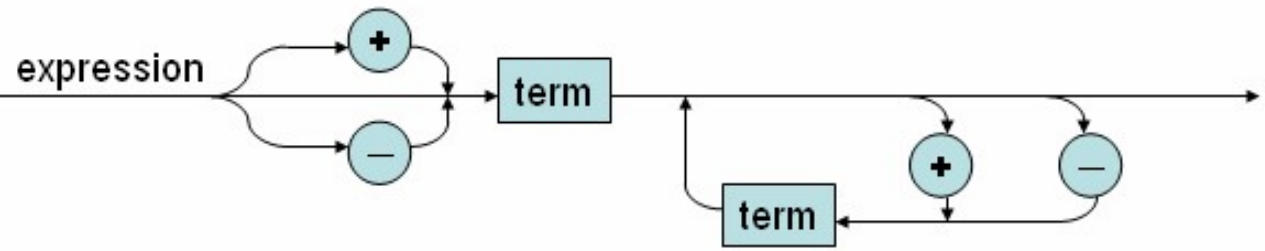


```
void compileTerm(void)
{ compileFactor();
  compileTerm2();}
void compileTerm2(void)
{switch (lookAhead->tokenType)
{case SB_TIMES:
    eat(SB_TIMES);
    compileFactor();
    compileTerm2();
    break;
 case SB_SLASH:
    eat(SB_SLASH);
    compileFactor();
    compileTerm2();
    break;
```

```
// 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);
  }
```

Phân tích expression

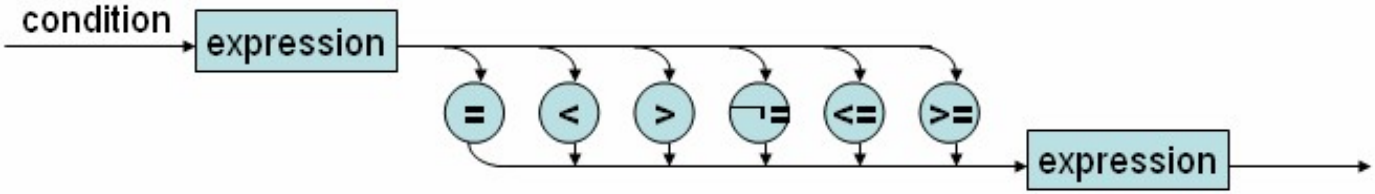
```
void compileExpression(void) {
  assert("Parsing an expression");
  switch (lookAhead->tokenType) {
  case SB_PLUS:
    eat(SB_PLUS);
    compileExpression2();
    break;
  case SB_MINUS:
    eat(SB_MINUS);
    compileExpression2();
    break;
  default:
    compileExpression2();
  }
```




```
void compileExpression3(void)
{
  switch (lookAhead->tokenType)
{
  case SB_PLUS:
    eat(SB_PLUS);
    compileTerm();
    compileExpression3();
    break;
  case SB_MINUS:
    eat(SB_MINUS);
    compileTerm();
    compileExpression3();
    break;
```

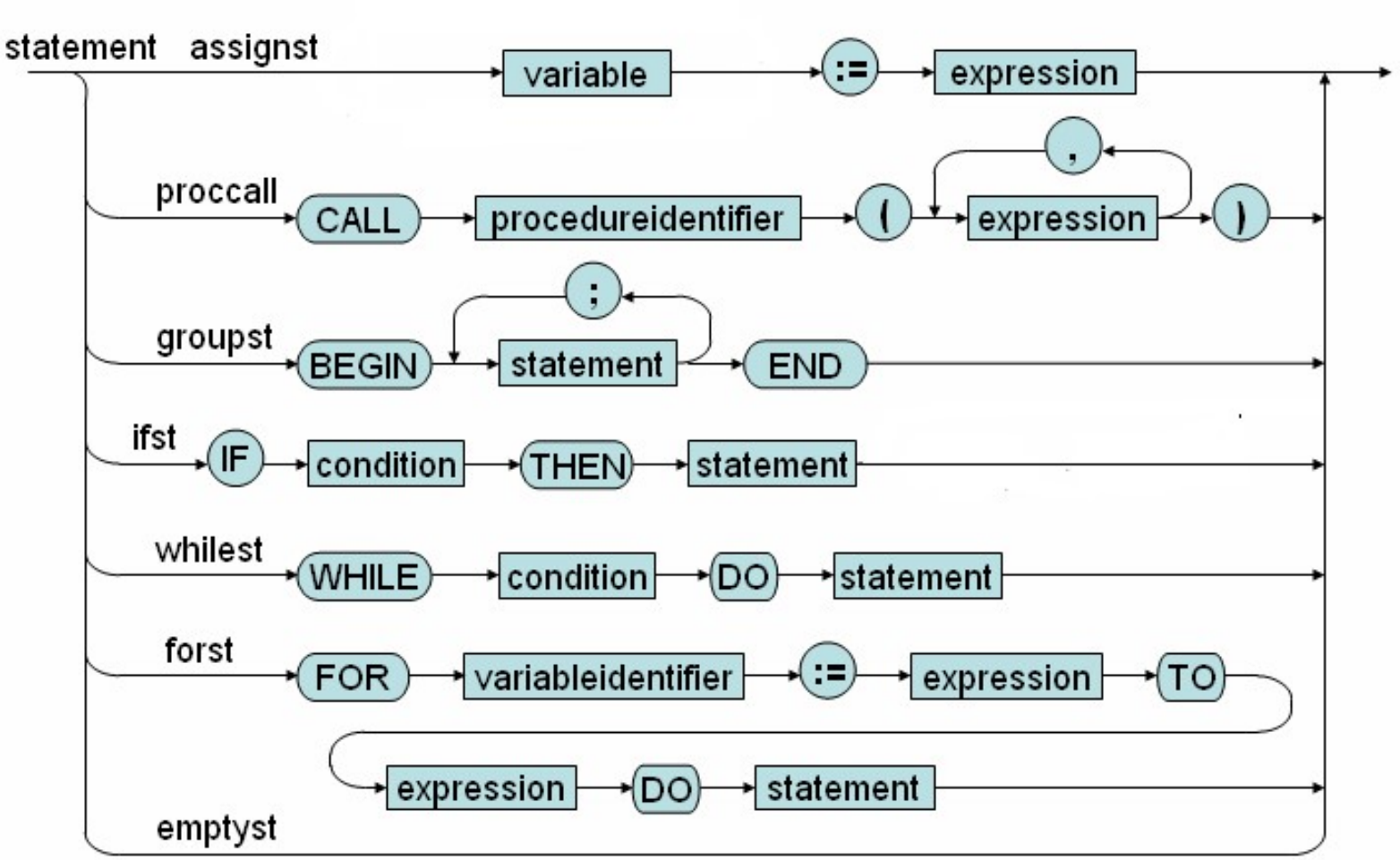
Phân tích condition

```
void compileCondition(void) {
  compileExpression();
  switch (lookAhead->tokenType) {
  case SB_EQ:
    eat(SB_EQ);
    compileExpression();
    break;
  case SB_NEQ:
    eat(SB_NEQ);
    compileExpression();
    break;
  case SB_LE:
    eat(SB_LE);
    compileExpression();
    break;
  case SB_LT:
    eat(SB_LT);
    compileExpression();
    break;
```




```
case SB_GE:
eat(SB_GE);
    compileExpression();
    break;
  case SB_GT:
    eat(SB_GT);
    compileExpression();
    break;
  default:
    error(ERR_INVALIDCOMPARATOR, lookAhead->lineNo,
 lookAhead->colNo);
  }
}
```

Sơ đồ cú pháp của lệnh KPL



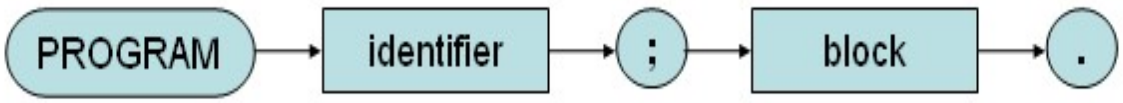
Phân tích statement

```
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;
  }
}
```

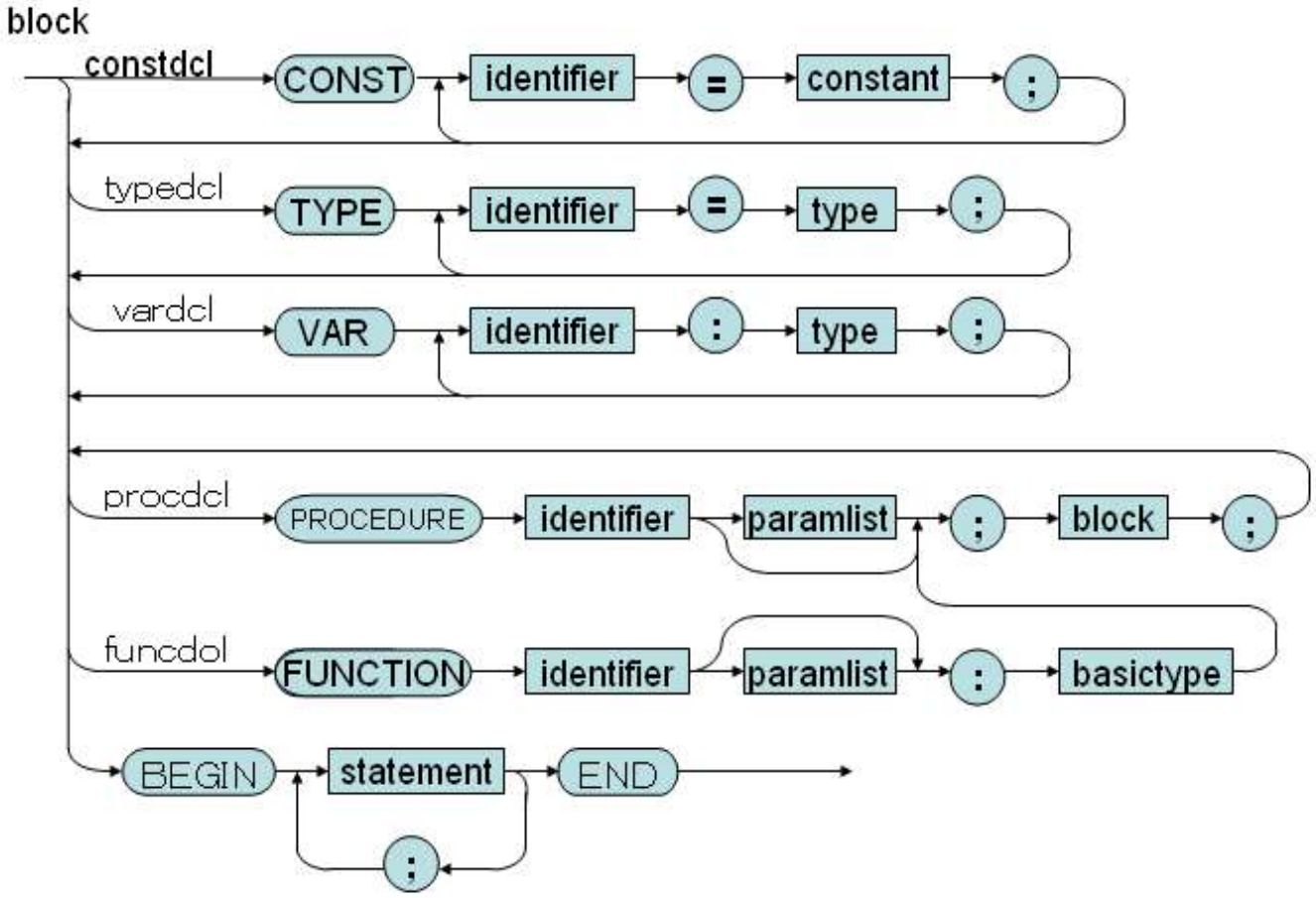
Phân tích program

```
void compileProgram(void)
{
  eat(KW_PROGRAM);
  eat(TK_IDENT);
  eat(SB_SEMICOLON);
  compileBlock();
  eat(SB_PERIOD);
}
```

program



Khối


block
constdcl
CONST
identifier
=
constant
;
typedcl
TYPE
identifier
=
type
;
vardcl
VAR
identifier
:
type
;
procdcl
PROCEDURE
identifier
paramlist
;
block
;
funcdol
FUNCTION
identifier
paramlist
:
basictype
BEGIN
statement
END
;

Phân tích block

```
void compileBlock(void)
{if (lookAhead->tokenType == KW_CONST)
   {eat(KW_CONST);
    while (lookAhead->tokenType == TK_IDENT)
   { eat(TK_IDENT);
      eat(SB_EQ);
      compileConstant();
      eat(SB_SEMICOLON); }
else
if (lookAhead->tokenType == KW_TYPE)
    {eat(KW_TYPE);
    while (lookAhead->tokenType == TK_IDENT)
     {eat(TK_IDENT);
      eat(SB_EQ);
      compileType();
      eat(SB_SEMICOLON);}
else
if (lookAhead->tokenType == KW_VAR)
  {eat(KW_VAR);
   while (lookAhead->tokenType == TK_IDENT)
   { eat(TK_IDENT);
      eat(SB_COLON);
      compileType();
      eat(SB_SEMICOLON);}
```

else

```
while ((lookAhead->tokenType == KW_FUNCTION)
|| (lookAhead->tokenType == KW_PROCEDURE))
       {if (lookAhead->tokenType ==
KW_FUNCTION)
      {eat(KW_FUNCTION);
       eat(TK_IDENT);
       compileParams();
       eat(SB_COLON);
       compileBasicType();
       eat(SB_SEMICOLON);
       compileBlock();
       eat(SB_SEMICOLON);}
    else
       {eat(KW_PROCEDURE);
       eat(TK_IDENT);
       compileParams();
       eat(SB_SEMICOLON);
       compileBlock();
       eat(SB_SEMICOLON);}}
else
 {eat(KW_BEGIN);
  compileStatement();
  while (lookAhead->tokenType ==
SB_SEMICOLON) {
    eat(SB_SEMICOLON);
    compileStatement();
     }
  eat(KW_END);}
}
```