

# Báo cáo LAB02

## IT3323 Mã lớp 161269

### Bài 3 : Phân tích ngữ nghĩa

Họ và tên: Dương Công Thuyết

MSSV: 20225932

#### I. Giải thích code Symtab

1. Mục đích: Symbol Table quản lý các đối tượng (constants, variables, types, functions, procedures, parameters) trong chương trình KPL, hỗ trợ kiểm tra khai báo và phân giải tên.

#### 2. Cấu trúc dữ liệu chính

- Object: đại diện một đối tượng (constant, variable, type, function, procedure, parameter, program)
- Chứa: tên, loại (kind), thuộc tính riêng theo loại
- Scope: phạm vi (block) chứa danh sách objects
- objList: danh sách objects trong scope
- owner: object sở hữu scope (program/function/procedure)
- outer: scope cha (hỗ trợ nested scopes)
- SymTab: bảng symbol chính
- program: object chương trình
- currentScope: scope hiện tại
- globalObjectList: danh sách built-in functions/procedures

#### 3. Các thao tác cơ bản

- Tạo objects: createConstantObject(), createVariableObject(), createTypeObject(), createFunctionObject(), createProcedureObject(), createParameterObject()
- Quản lý scope:
  - + createScope(): tạo scope mới
  - + enterBlock(): vào scope (đặt currentScope)
  - + exitBlock(): ra scope (quay về scope cha)
- Khai báo và tìm kiếm:
  - + declareObject(): thêm object vào scope hiện tại
  - + lookupObject(): tìm object theo tên (từ current scope lên outer scopes, rồi global list)
  - + findObject(): tìm trong một danh sách cụ thể

#### 4. Cơ chế Scope (Nested Scopes)

- Scope lồng nhau: mỗi function/procedure có scope riêng, scope con có thể truy cập scope cha

- Tìm kiếm theo thứ tự: current scope → outer scopes → global list
- Shadowing: object trong scope con có thể che (shadow) object cùng tên ở scope cha

## 5. Quy trình hoạt động trong Parser

Khởi tạo: initSymTab() tạo SymTab và khởi tạo built-in functions (READC, READI, WRITEI, WRITEC, WRITELN)

- Parse Program:
  - + Tạo program object
  - + enterBlock() vào scope của program
  - + Parse các khai báo (constants, types, variables, functions, procedures)
  - + exitBlock() khi kết thúc program
- Parse Block (Function/Procedure):
  - + Tạo function/procedure object
  - + enterBlock() vào scope của function/procedure
  - + Parse parameters (thêm vào scope và paramList)
  - + Parse body (constants, types, variables)
  - + exitBlock() khi kết thúc block
- Khai báo Objects:
  - + Kiểm tra duplicate trong current scope
  - + Tạo object tương ứng
  - + Gán thuộc tính (type, value, ...)
  - + declareObject() để thêm vào scope
- Tìm kiếm Objects:
  - + Khi gặp identifier, dùng lookupObject() để tìm
  - + Nếu không tìm thấy → báo lỗi undeclared identifier

## 6. Đặc điểm quan trọng

- Parameters được thêm vào cả scope của function/procedure và paramList của function/procedure đó
- Types được duplicate khi sử dụng để tránh thay đổi type gốc
- Constants có thể reference đến constants khác thông qua lookupObject()
- Array types hỗ trợ nested arrays (mảng nhiều chiều)

### II. Code đã implement

#### 1. compileProgram() - Xử lý chương trình chính

Tạo program object từ tên chương trình

Vào scope của program (enterBlock)

Parse các khai báo trong block

Ra khỏi scope (exitBlock) khi kết thúc

#### 2. compileBlock() - Xử lý khai báo constants

Kiểm tra từ khóa CONST

Với mỗi constant:

Kiểm tra duplicate trong scope hiện tại

Tạo constant object

Parse giá trị constant (số, ký tự, hoặc reference đến constant khác)

Gán giá trị và khai báo vào symbol table

    3. compileBlock2() - Xử lý khai báo types

Kiểm tra từ khóa TYPE

Với mỗi type alias:

Kiểm tra duplicate

Tạo type object

Parse actual type (INTEGER, CHAR, ARRAY, hoặc type alias khác)

Gán và khai báo vào symbol table

    4. compileBlock3() - Xử lý khai báo variables

Kiểm tra từ khóa VAR

Với mỗi variable:

Kiểm tra duplicate

Tạo variable object

Parse type của variable

Gán type và khai báo vào symbol table

    5. compileFuncDecl() - Xử lý khai báo function

Parse tên function

Kiểm tra duplicate

Tạo function object và khai báo

Vào scope của function

Parse parameters (thêm vào scope và paramList)

Parse return type và gán vào function object

Parse body của function

Ra khỏi scope khi kết thúc

    6. compileProcDecl() - Xử lý khai báo procedure

Tương tự function nhưng không có return type

Parse tên, kiểm tra duplicate, tạo object

Vào scope, parse parameters và body

Ra khỏi scope khi kết thúc

    7. compileConstant() và compileConstant2() - Tạo giá trị constant

Hỗ trợ số dương/âm, ký tự, và reference đến constant khác

compileConstant(): xử lý dấu +/- và ký tự

compileConstant2(): xử lý số nguyên và identifier (tìm constant đã khai báo)

Nếu là identifier: dùng lookupObject() để tìm constant, duplicate giá trị

    8. compileUnsignedConstant() - Tạo giá trị constant không dấu

Xử lý số nguyên, ký tự, hoặc identifier

Tương tự compileConstant2() nhưng không xử lý dấu

    9. compileType() - Tạo type

Hỗ trợ INTEGER, CHAR, ARRAY, và type alias

Với ARRAY: parse kích thước và element type (đệ quy)

Với type alias: dùng lookupObject() để tìm type đã khai báo, duplicate type

Luôn duplicate type để tránh thay đổi type gốc

#### 10. compileBasicType() - Tạo basic type

Chỉ xử lý INTEGER và CHAR

Dùng cho parameters và return type của function

Duplicate type từ intType/charType

#### 11. compileParam() - Xử lý parameter

Hỗ trợ value parameter và reference parameter (VAR)

Với mỗi parameter:

Kiểm tra duplicate trong scope hiện tại (không kiểm tra outer scope để cho phép shadowing)

Xác định loại (value/reference)

Tạo parameter object với owner là function/procedure hiện tại

Parse type (chỉ basic type)

Khai báo vào scope (tự động thêm vào paramList của function/procedure)

### III. Kết quả với examples:

#### 1. example1.kpl

```
PROGRAM EXAMPLE1;
BEGIN
END. (* Example 1 *)
```

The screenshot shows a code editor interface with the following details:

- Left Sidebar:** Shows a tree view of files in the 'SYMTAB\_INCOMPLETE' directory, including '.godo', 'tests', and several 'exampleX.kpl' files (example1.kpl, example2.kpl, example3.kpl, example4.kpl, example5.kpl, example6.kpl). It also lists source files like 'charcode.c', 'charcode.h', 'debug.c', 'debug.h', 'error.c', 'error.h', and 'Ketqua.pdf'.
- Central Area:** Displays the content of 'example1.kpl'. The code is:

```
PROGRAM EXAMPLE1;
BEGIN
END. (* Example 1 *)
```
- Terminal:** Shows the command being run and its output:

```
duongcongthuyet@Dongs-MacBook-Pro SymTab_Incomplete % ./parser tests/example1.kpl
Program EXAMPLE1
duongcongthuyet@Dongs-MacBook-Pro SymTab_Incomplete %
```
- Bottom Status Bar:** Shows various status icons and text, including 'Connected to Discord', 'UTF-8 LF', 'Plain Text', 'Pieces Code Lens', 'Go Live', 'Godo' (highlighted in orange), 'Colorize: 0 variables', 'Colorize', and 'Prettier'.

#### 2. example2.kpl

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar titled 'SYMTAB\_INCOMPLETED' containing files like .odo, tests, example1.kpl, example2.kpl (which is selected), example3.kpl, example4.kpl, example5.kpl, example6.kpl, Bai3\_Symtab.pdf, charcode.c, charcode.h, charcode.o, debug.c, debug.h, debug.o, error.c, error.h, error.o, Ketqua.pdf, main.c, main.o, parser.c, parser.h, reader.c, reader.h, reader.o, scanner.c, scanner.h, scanner.o, symtab.c, symtab.h, and symtab.o. The main editor area shows the content of example2.kpl:

```
1 PROGRAM EXAMPLE2; (* Factorial *)
2
3 VAR N : INTEGER;
4
5 FUNCTION F(N : INTEGER) : INTEGER;
6 BEGIN
7   IF N = 0 THEN F := 1 ELSE F := N * F (N - 1);
8 END;
9
10 BEGIN
11   FOR N := 1 TO 7 DO
12     BEGIN
13       CALL WRITELN;
14       CALL WRITEI(F(n));
15     END;
16 END. (* Factorial *)
```

The terminal pane at the bottom shows the output of running the program:

```
duongcongthuyet@Duongs-MacBook-Pro SymTab_Incompleted % ./parser tests/example2.kpl
Program EXAMPLE2
  Var N : Int
    Function F : Int
      Param : Int
duongcongthuyet@Duongs-MacBook-Pro SymTab_Incompleted %
```

### 3. example3.kpl

The screenshot shows a code editor interface with a dark theme, similar to the previous one. The sidebar 'SYMTAB\_INCOMPLETED' contains the same files as before. The main editor area shows the content of example3.kpl:

```
1 PROGRAM EXAMPLE2; (* Factorial *)
2
3 VAR N : INTEGER;
4
5 FUNCTION F(N : INTEGER) : INTEGER;
6 BEGIN
7   IF N = 0 THEN F := 1 ELSE F := N * F (N - 1);
8 END;
9
10 BEGIN
11   FOR N := 1 TO 7 DO
12     BEGIN
13       CALL WRITELN;
14       CALL WRITEI(F(n));
15     END;
16 END. (* Factorial *)
```

The terminal pane at the bottom shows the output of running the program:

```
duongcongthuyet@Duongs-MacBook-Pro SymTab_Incompleted % ./parser tests/example3.kpl
Program EXAMPLE3
  Var I : Int
  Var N : Int
  Var P : Int
  Var Q : Int
  Var R : Char
  Procedure Hanoi
    Param : Int
    Param : Int
    Param : Int
duongcongthuyet@Duongs-MacBook-Pro SymTab_Incompleted %
```

### 4. example4.kpl

```
PROGRAM EXAMPLE2; (* Factorial *)
VAR N : INTEGER;
FUNCTION F(N : INTEGER) : INTEGER;
BEGIN
  IF N = 0 THEN F := 1 ELSE F := N * F (N - 1);
END;
BEGIN
  FOR N := 1 TO 7 DO
    BEGIN
      CALL WRITELN;
      CALL WRITEI(F(n));
    END;
END. (* Factorial *)
```

The screenshot shows a code editor window with the file `example2.kpl` open. The code defines a program `EXAMPLE2` that calculates the factorial of a number `N`. It uses a recursive function `F` and a loop from 1 to 7 to print the results. The code editor interface includes a sidebar with project files, a terminal window at the bottom showing command-line output, and various toolbars and status bars.

## 5. example5.kpl

```
PROGRAM EXAMPLE5;
Const C = 1;
Type T = Char;
Function F : Char;
Param : Int;
Const B = 1;
Type A = Arr(5,Char);
BEGIN
  CALL WRITELN;
  CALL WRITEI(F);
END;
```

The screenshot shows a code editor window with the file `example2.kpl` open. The code defines a program `EXAMPLE5` with a parameter `C` set to 1. It also defines a function `F` that prints its value. The code editor interface includes a sidebar with project files, a terminal window at the bottom showing command-line output, and various toolbars and status bars.

## 6. example6.kpl

```

PROGRAM EXAMPLE2; (* Factorial *)
VAR N : INTEGER;
FUNCTION F(N : INTEGER) : INTEGER;
BEGIN
    IF N = 0 THEN F := 1 ELSE F := N * F (N - 1);
END;
BEGIN
    FOR N := 1 TO 7 DO
        BEGIN
            CALL WRITELN;
            CALL WRITEI(F(n));
        END;
    END. (* Factorial *)

```

The screenshot shows a code editor interface with multiple tabs and files. The current file is `example2.kpl`. The code itself is a simple KPL program that calculates factorials and prints them to the console. The code editor has a terminal window at the bottom showing the execution of the program and its output.

## IV. Kiểm tra với bài tập ở lớp lý thuyết

### 1. Exercise:

## XÂY DỰNG CHƯƠNG TRÌNH DỊCH

**Bài 1.** Một ma trận vuông là ma trận tam giác trên nếu mọi phần tử nằm dưới đường chéo chính là bằng 0. Viết chương trình trên ngôn ngữ KPL để nhập một ma trận vuông kích thước nxn, n nhập từ bàn phím. In ra 1 nếu ma trận là tam giác trên , 0 nếu ngược lại.

Ví dụ một ma trận tam giác trên:

Upper triangular matrix: U

1	1/2	3	0
0	5	0	1
0	0	4	-2
0	0	0	3

### Bài 2

Viết chương trình tính và in ra tổng 2 số bằng ngôn ngữ KPL. Chỉ ra phân tích trái của chương trình (dãy số hiệu sản xuất được dùng trong suy diễn trái)

### 2. Exercise1:

```

PROGRAM EX1;
TYPE T = INTEGER;
ROW = ARRAY(1..20) OF T;
VAR A : ARRAY(1..20) OF ROW;
N : INTEGER;
I : INTEGER;
J : INTEGER;
ISUPPER : INTEGER;
PROCEDURE INPUT;
BEGIN
  N := READI;
  FOR I := 1 TO N DO
    FOR J := 1 TO N DO
      A(I,J) := READI;
END;
PROCEDURE CHECKUPPER;
BEGIN
  ISUPPER := 1;
  I := 2;
  WHILE I <= N DO
    BEGIN
      J := 1;
      WHILE J < I DO
        BEGIN
          IF A(I,J) > A(I,ISUPPER) THEN
            ISUPPER := J;
          J := J + 1;
        END;
      I := I + 1;
    END;
  CALL INPUT;
  CALL CHECKUPPER;
  CALL WRITEI(ISUPPER);
  CALL WRITELN;
END.

```

```

PROGRAM EX1;
TYPE T = INTEGER;
ROW = ARRAY(1..20) OF T;
VAR A : ARRAY(1..20) OF ROW;
N : INTEGER;
I : INTEGER;
J : INTEGER;
ISUPPER : INTEGER;
PROCEDURE INPUT;
BEGIN
  N := READI;
  FOR I := 1 TO N DO
    FOR J := 1 TO N DO
      A(I,J) := READI;
END;
PROCEDURE CHECKUPPER;
BEGIN
  ISUPPER := 1;
  I := 2;
  WHILE I <= N DO
    BEGIN
      J := 1;
      WHILE J < I DO
        BEGIN
          IF A(I,J) > A(I,ISUPPER) THEN
            ISUPPER := J;
          J := J + 1;
        END;
      I := I + 1;
    END;
  END;
  WRITEI(I);
  WRITELN;
END.

```

Problems Output Debug Console Terminal Ports DevDb Postman Console

duongcongthuyet@uongs-MacBook-Pro SymTab\_Incompleted % ./parser tests/ex1.kpl

Program EX1

Type T = Int

Type ROW = Arr(20,Int)

Var A : Arr(20,Arr(20,Int))

Var I : Int

Var J : Int

Var I : Int

Var J : Int

Var ISUPPER : Int

Procedure INPUT

Procedure CHECKUPPER

### 3. Exercise2:

```
ex2.kpl — SymTab_Incompleted
parser.c ex1.kpl ex2.kpl
tests > ex2.kpl
1 PROGRAM EX2;
2 VAR A : INTEGER;
3 B : INTEGER;
4 SUM : INTEGER;
5
6 PROCEDURE INPUTANDSUM;
7 BEGIN
8   A := READI;
9   B := READI;
10  SUM := A + B;
11 END;
12
13 BEGIN
14   CALL INPUTANDSUM;
15   CALL WRITEI(SUM);
16   CALL WRITELN;
17 END.
18
19

Problems Output Debug Console Terminal Ports DevDb Postman Console
duongcongthuyet@uongs-MacBook-Pro SymTab_Incompleted % ./parser tests/ex2.kpl
Program EX2
  Var A : Int
  Var B : Int
  Var SUM : Int
  Procedure INPUTANDSUM
duongcongthuyet@uongs-MacBook-Pro SymTab_Incompleted %
```