# BÁO CÁO THỰC HÀNH XÂY DỰNG CHƯƠNG TRÌNH DỊCH

# BÀI 3: PHÂN TÍCH NGỮ NGHĨA

Họ và tên: Vũ Văn Minh

MSSV: 20225747

# I. Giải thích code

## 1. Cấu trúc dữ liệu nền tảng

Hệ thống được tổ chức theo phân cấp 3 tầng để quản lý định danh hiệu quả:

- **SymTab (Symbol Table):** Đối tượng quản lý cao nhất.
  - Chứa con trỏ tới chương trình chính (program).
  - Theo dõi phạm vi hiện tại (currentScope).
  - Duy trì danh sách các đối tượng toàn cục (globalObjectList - ví dụ: các hàm built-in).
- **Scope (Phạm vi):** Đại diện cho một khối khai báo (Program, Function, Procedure).
  - Chứa danh sách các Object (objList) được khai báo trong phạm vi đó.
  - Lưu con trỏ tới "chủ sở hữu" (owner) và phạm vi bao ngoài (outer) để tạo thành chuỗi liên kết scope.
- **Object (Đối tượng):** Đại diện cho một định danh (identifier) cụ thể.
  - Chứa tên, loại (kind) và các thuộc tính cụ thể.

## 2. Phân loại Object và Cơ chế lưu trữ

Mỗi Object được phân loại qua ObjectKind và sử dụng union để tiết kiệm bộ nhớ, chỉ lưu các thuộc tính phù hợp với loại của nó:

- **OBJ_CONSTANT:** Hằng số (Lưu giá trị constant).
- **OBJ_TYPE:** Kiểu dữ liệu (Lưu định nghĩa kiểu).
- **OBJ_VARIABLE:** Biến (Lưu kiểu dữ liệu và scope chứa nó).
- **OBJ_FUNCTION:** Hàm (Lưu kiểu trả về, danh sách tham số và scope riêng).
- **OBJ_PROCEDURE:** Thủ tục (Lưu danh sách tham số và scope riêng).
- **OBJ_PARAMETER:** Tham số (Lưu kiểu và cách truyền: giá trị hoặc tham chiếu).
- **OBJ_PROGRAM:** Chương trình gốc (Đối tượng gốc chứa scope toàn cục).

## 3. Hệ thống quản lý Type (Kiểu dữ liệu)

Hệ thống hỗ trợ các kiểu cơ bản và cơ chế so sánh:

- **Các kiểu cơ bản:**
    - TP_INT: Số nguyên.
    - TP_CHAR: Ký tự.
    - TP_ARRAY: Mảng (chứa kích thước mảng và kiểu của phần tử).
- **Các hàm tiện ích:**
    - makeIntType(), makeCharType(): Tạo kiểu đơn giản.
    - makeArrayType(size, elementType): Tạo kiểu mảng.
    - duplicateType(): Sao chép sâu (deep copy) một kiểu, xử lý đệ quy cho mảng lồng nhau.
    - compareType(): Kiểm tra sự tương đồng giữa hai kiểu (so sánh typeClass; với mảng thì so sánh cả kích thước và kiểu phần tử đệ quy).

## 4. Hệ thống quản lý Constant Value

Giá trị hằng số được quản lý qua cấu trúc ConstantValue sử dụng union để lưu trữ giá trị int hoặc char, kèm theo thông tin type để phân biệt.

Các hàm hỗ trợ: makeIntConstant, makeCharConstant, duplicateConstantValue.

## 5. Cơ chế tạo và quản lý Object

- **Nguyên tắc chung:** Mỗi loại Object có hàm tạo riêng (ví dụ createVariableObject), chịu trách nhiệm cấp phát bộ nhớ, gán tên, gán kind, và khởi tạo thuộc tính mặc định.
- **Xử lý đặc biệt:**
    - createProgramObject: Tạo scope gốc và gán vào symtab->program.
    - createFunctionObject / createProcedureObject: Tự động tạo scope riêng với owner là chính nó.
    - createParameterObject: Cần biết owner (hàm/thủ tục chứa nó).
    - createVariableObject: Lưu con trỏ tới currentScope.
- **Danh sách Object (ObjectNode):** Sử dụng danh sách liên kết đơn.
    - addObject(): Thêm Object vào cuối danh sách.
    - findObject(): Duyệt tuần tự để tìm kiếm Object theo tên (trả về NULL nếu không thấy).

## 6. Cơ chế quản lý Scope (Phạm vi)

- **Cấu trúc lồng nhau:**
    - Scope toàn cục (outer = NULL).
    - Scope con (Function/Procedure) trỏ outer về scope nơi nó được khai báo.
- **Quản lý Scope động (Enter/Exit Block):**

- enterBlock(scope): Khi Parser vào một khối code (Function/Procedure), currentScope của SymTab được cập nhật sang scope mới. Mọi khai báo mới sẽ nằm trong scope này.
- exitBlock(): Khi kết thúc khối, currentScope quay về outer.

- **Tìm kiếm (Lookup) và Shadowing:**
  - lookupObject(): Tìm kiếm từ currentScope, nếu không thấy thì tìm ngược lên outer, cuối cùng là globalObjectList (built-in).
  - Shadowing: Khai báo ở scope trong sẽ "che khuất" khai báo cùng tên ở scope ngoài do thứ tự tìm kiếm ưu tiên scope hiện tại.

### 7. Khởi tạo Symbol Table

Hàm initSymTab() thực hiện:

- Cấp phát bộ nhớ cho SymTab.
- Tạo các kiểu toàn cục dùng chung (intType, charType).
- Tạo các hàm/thủ tục Built-in (có sẵn):
  - Hàm: READC (đọc char), READI (đọc int).
  - Thủ tục: WRITEI (in int), WRITEC (in char), WRITELN (xuống dòng).

## II. ỨNG DỤNG SYMTAB TRONG PARSER

Phần này mô tả cách Parser gọi các hàm của SymTab để xây dựng bảng ký hiệu và kiểm tra ngữ nghĩa (semantic) trong quá trình biên dịch.

### 1. Quy trình tổng thể trong compile()

1. Khởi tạo: Gọi initSymTab() để tạo môi trường rỗng với các built-in.
2. Biên dịch: Gọi compileProgram() để phân tích toàn bộ mã nguồn.
3. Debug: Gọi printObject() để in toàn bộ Symbol Table kiểm tra.
4. Dọn dẹp: Gọi cleanSymTab() giải phóng bộ nhớ.

### 2. Phân tích cấu trúc chương trình (PROGRAM)

- **Bắt đầu:** Sau khi đọc từ khóa PROGRAM và tên chương trình, gọi createProgramObject().
- **Vào Scope:** Gọi enterBlock() để vào scope của chương trình.
- **Xử lý thân:** Gọi compileBlock() để xử lý các khai báo và lệnh.
- **Kết thúc:** Gọi exitBlock() để quay ra scope ngoài (về NULL).

### 3. Phân tích các phần khai báo (Declarations)

Các hàm compileBlock, compileBlock2, compileBlock3 xử lý tuần tự:

- **CONST (Hằng):**

- Kiểm tra trùng tên (checkFreshIdent).
- Tạo đối tượng (createConstantObject).
- Phân tích giá trị (compileConstant) và gán vào thuộc tính.
- Đăng ký vào bảng (declareObject).

- **TYPE (Kiểu):**
  - Tương tự CONST. Tạo đối tượng OBJ_TYPE.
  - Phân tích định nghĩa kiểu (compileType - xử lý mảng, int, char) và gán actualType.
- **VAR (Biến):**
  - Tương tự. Tạo đối tượng OBJ_VARIABLE.
  - Biến tự động lưu scope hiện tại để biết nó thuộc về đâu.

## 4. Phân tích chương trình con (Function & Procedure)

Quy trình cho compileFuncDecl và compileProcDecl:

1. Khai báo: Kiểm tra tên (checkFreshIdent), tạo Object (Func/Proc), và declareObject vào scope cha.
2. Vào Scope con: Gọi enterBlock() vào scope riêng của hàm/thủ tục đó.
3. Tham số (Parameters):
   - compileParams xử lý danh sách tham số.
   - Từng tham số được tạo (createParameterObject) và đăng ký (declareObject).
   - Lưu ý: Tham số được lưu ở 2 nơi: trong objList của scope (để tra cứu tên) và trong paramList của hàm/thủ tục (để kiểm tra thứ tự/kiểu khi gọi).
4. Kiểu trả về: (Chỉ cho Function) Phân tích và gán returnType.
5. Thân hàm: Gọi compileBlock() xử lý các lệnh bên trong.
6. Kết thúc: Gọi exitBlock().

## 5. Kiểm tra ngữ nghĩa (Semantic Checks)

Hệ thống sử dụng các hàm kiểm tra chặt chẽ:

- **Kiểm tra trùng tên (checkFreshIdent):** Chỉ tìm trong currentScope. Nếu thấy -> Lỗi ERR_DUPLICATE_IDENT. Dùng khi khai báo mới.
- **Kiểm tra đã khai báo (checkDeclaredIdent):** Tìm trong toàn bộ các scope lồng nhau (Lookup). Nếu không thấy -> Lỗi ERR_UNDECLARED_IDENT. Dùng khi sử dụng biến/hàm.
- **Kiểm tra loại cụ thể:**
  - Các hàm như checkDeclaredConstant, checkDeclaredType, checkDeclaredVariable, checkDeclaredFunction, checkDeclaredProcedure.

- Cơ chế 2 tầng: (1) Lookup xem có tồn tại không -> (2) Kiểm tra kind có đúng loại mong muốn không.

# 6. Xử lý biểu thức và lệnh (Statements & Expressions)

## A. Xử lý hằng số trong biểu thức

- compileConstant2: Xử lý số (TK_NUMBER) hoặc định danh (TK_IDENT). Nếu là định danh, nó tra cứu Constant Object và sao chép giá trị.
- compileConstant: Xử lý dấu âm/dương. Kiểm tra lỗi ERR_TYPE_INCONSISTENCY (ví dụ: dấu trừ trước kiểu char).

## B. Xử lý Factor (Thành phần biểu thức)

Khi gặp một định danh (TK_IDENT), Parser kiểm tra ngữ cảnh (LookAhead):

1. Lời gọi hàm (SB_LPAR - `(`): Kiểm tra checkDeclaredFunction -> Xử lý tham số (compileArguments).
2. Truy cập mảng (SB_LSEL - `[`): Kiểm tra checkDeclaredIdent (phải là Variable/Parameter) -> Xử lý chỉ số (compileIndexes).
3. Biến đơn: Kiểm tra checkDeclaredIdent. Hợp lệ nếu là: Variable, Parameter, Constant, hoặc tên Function (để lấy giá trị trả về).

## C. Lệnh gọi thủ tục (CALL)

- Cú pháp: CALL <tên_thủ_tục> (tham_số)
- Kiểm tra: Gọi checkDeclaredProcedure (đảm bảo tồn tại và đúng là Procedure). Sau đó xử lý tham số.

## D. Lệnh gán (:=)

- Cú pháp: LValue := Expression
- LValue (Vế trái): Gọi compileLValue. Kiểm tra định danh phải là Variable, Parameter, hoặc tên Function (trong thân hàm đó). Nếu không -> Lỗi ERR_INVALID_LVALUE.

## E. Vòng lặp FOR

- Kiểm tra biến điều khiển sau từ khóa FOR bằng checkDeclaredVariable.

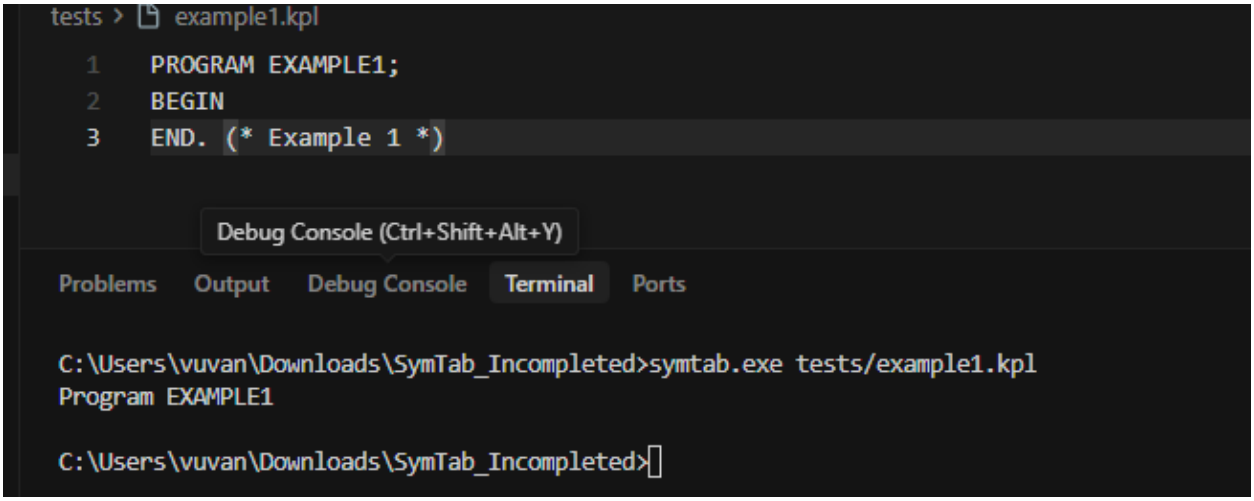
# 7. Tổng kết luồng hoạt động

Sự tương tác giữa Parser và SymTab tạo nên một quy trình biên dịch có kiểm tra ngữ nghĩa cơ bản:

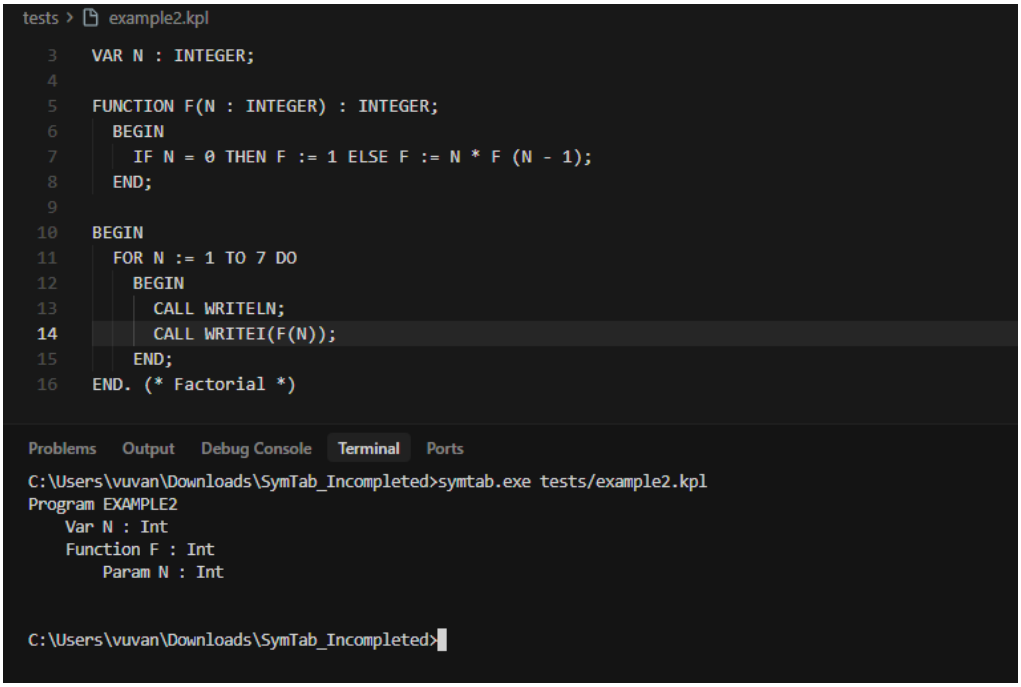
- Parser: Nhận diện cấu trúc ngữ pháp (Function, Var, If, For...).
- SymTab: Lưu trữ thông tin ngữ cảnh (Scope, Type, Attributes).
- Kết hợp: Parser gọi SymTab đúng thời điểm (Enter/Exit scope, Declare, Lookup) để đảm bảo chương trình không chỉ đúng cú pháp mà còn đúng logic.

# III. Kết quả thực hiện 6 examples

1. Example1




2. Example2




3. Example3

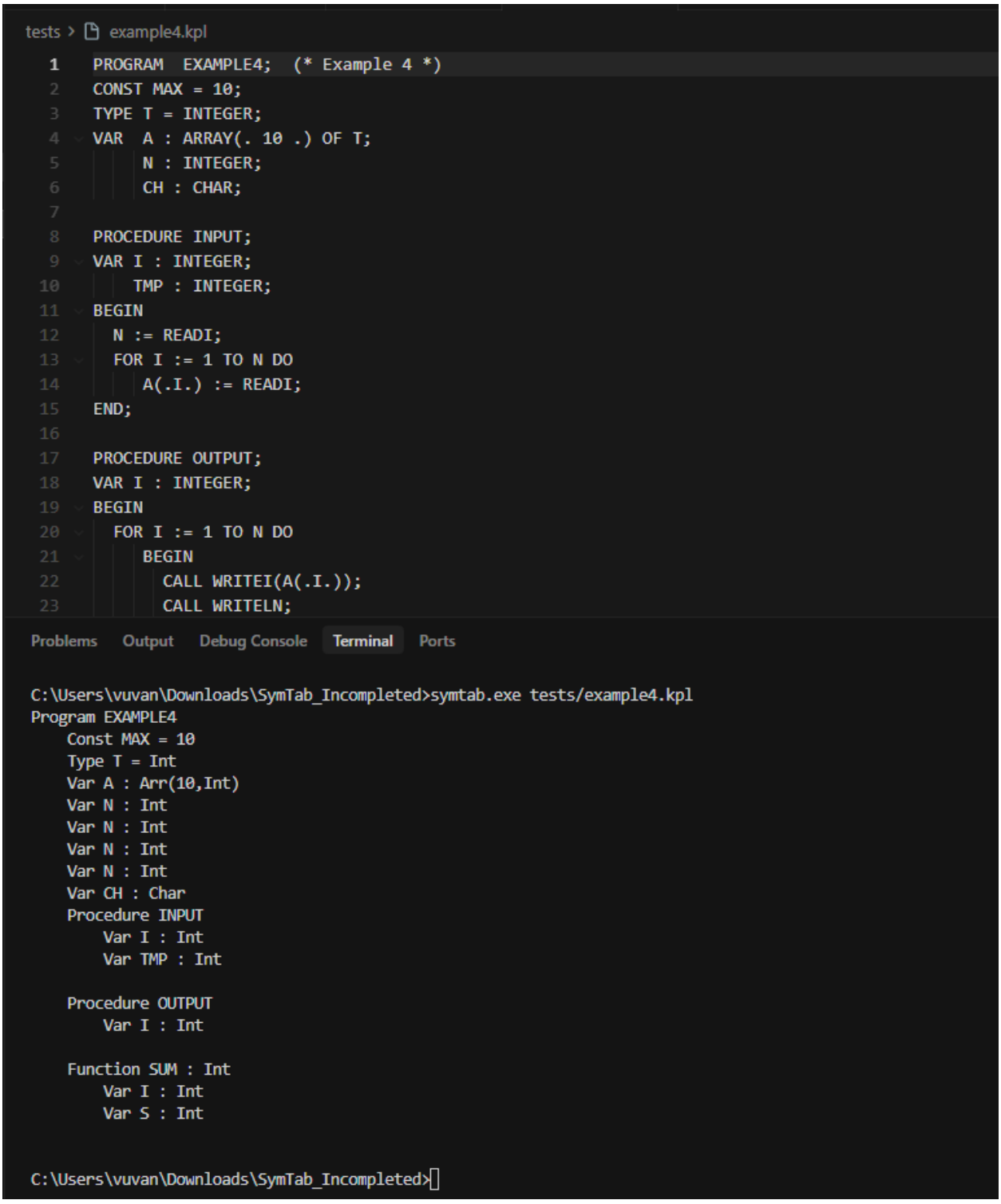
tests > example3.kpl

```
1   PROGRAM  EXAMPLE3;  (* TOWER OF HANOI *)
2   VAR  I:INTEGER;
3        N:INTEGER;
4        P:INTEGER;
5        Q:INTEGER;
6        C:CHAR;
7
8   PROCEDURE  HANOI(N:INTEGER;  S:INTEGER;  Z:INTEGER);
9   BEGIN
10    IF  N != 0  THEN
11      BEGIN
12        CALL  HANOI(N-1,S,6-S-Z);
13        I:=I+1;
14        CALL  WRITELN;
15        CALL  WRITEI(I);
16        CALL  WRITEI(N);
17        CALL  WRITEI(S);
18        CALL  WRITEI(Z);
19        CALL  HANOI(N-1,6-S-Z,Z)
20      END
21  END;  (*END OF HANOI*)
22
23  BEGIN
24    FOR  N := 1  TO  4  DO
25      BEGIN
26        FOR  I:=1  TO  4  DO
27          CALL  WRITEC(' ');
28        C :=  READC;
29        CALL  WRITEC(C)
30      END;
31    P:=1;
```

Problems Output Debug Console **Terminal** Ports

```
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/example3.kpl
Program EXAMPLE3
    Var I : Int
    Var N : Int
    Var P : Int
    Var Q : Int
    Var C : Char
    Procedure HANOI
        Param N : Int
        Param S : Int
        Param Z : Int
```

4. Example4


tests > example4.kpl
1 PROGRAM EXAMPLE4; (* Example 4 *)
2 CONST MAX = 10;
3 TYPE T = INTEGER;
4 VAR A : ARRAY(. 10 .) OF T;
5 N : INTEGER;
6 CH : CHAR;
7
8 PROCEDURE INPUT;
9 VAR I : INTEGER;
10 TMP : INTEGER;
11 BEGIN
12 N := READI;
13 FOR I := 1 TO N DO
14 A(.I.) := READI;
15 END;
16
17 PROCEDURE OUTPUT;
18 VAR I : INTEGER;
19 BEGIN
20 FOR I := 1 TO N DO
21 BEGIN
22 CALL WRITEI(A(.I.));
23 CALL WRITELN;
Problems Output Debug Console Terminal Ports
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/example4.kpl
Program EXAMPLE4
Const MAX = 10
Type T = Int
Var A : Arr(10,Int)
Var N : Int
Var N : Int
Var N : Int
Var N : Int
Var CH : Char
Procedure INPUT
Var I : Int
Var TMP : Int
Procedure OUTPUT
Var I : Int
Function SUM : Int
Var I : Int
Var S : Int
C:\Users\vuvan\Downloads\SymTab_Incompleted>

5. Example5

tests › example5.kpl

```
1   PROGRAM EXAMPLE5; (* Example 5 *)
2   CONST C = 1;
3   TYPE T = CHAR;
4   FUNCTION F(I : INTEGER):CHAR;
5   CONST B = C;
6   TYPE A = ARRAY(.5.) OF T;
7   BEGIN
8   END;
9   BEGIN
10  END. (* Example 5 *)
```

Problems Output Debug Console **Terminal** Ports

```
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/example5.kpl
Program EXAMPLE5
    Const C = 1
    Type T = Char
    Function F : Char
        Param I : Int
        Const B = 1
        Type A = Arr(5,Char)
```

6. Example6

tests › example6.kpl

```
1   PROGRAM EXAMPLE6;
2     CONST C1 = 10;
3           C2 = 'a';
4     TYPE T1 = ARRAY(. 10 .) OF INTEGER;
5     VAR V1 : INTEGER;
6         V2 : ARRAY(. 10 .) OF T1;
7
8     FUNCTION F(P1 : INTEGER; VAR P2 : CHAR) : INTEGER;
9       BEGIN
10        F := C1;
11      END;
12
13    PROCEDURE P(V1 : INTEGER);
14      CONST C1 = 'a';
15            C3 = 10;
16      TYPE T1 = INTEGER;
17           T2 = ARRAY(. 10 .) OF T1;
18      VAR V2 : T2;
19          V3 : CHAR;
20      BEGIN
21      V3 := 'a';
22         V1 := F(C3,V3);
23      END;
24
25  BEGIN
```

Problems Output Debug Console **Terminal** Ports

```
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/example6.kpl
Program EXAMPLE6
    Const C1 = 10
    Const C2 = 'a'
    Type T1 = Arr(10,Int)
    Var V1 : Int
    Var V2 : Arr(10,Arr(10,Int))
    Function F : Int
        Param P1 : Int
        Param VAR P2 : Char

    Procedure P
        Param V1 : Int
        Const C1 = 'a'
        Const C3 = 10
        Type T1 = Int
        Type T2 = Arr(10,Int)
        Var V2 : Arr(10,Int)
        Var V3 : Char
```

# IV. Kết quả với assign

1. Assign 1

tests > assign1.kpl

```
1   PROGRAM ASSIGN1;
2   CONST MAX = 100;
3   VAR N : INTEGER;
4       U : ARRAY(. 100 .) OF ARRAY(. 100 .) OF INTEGER;
5       I : INTEGER;
6       J : INTEGER;
7
8   PROCEDURE INPUT;
9   BEGIN
10    CALL WRITEC('N');
11    CALL WRITEC('=');
12    N := READI;
13
14    FOR I := 1 TO N DO
15      FOR J := 1 TO N DO
16        BEGIN
17          U(.I.)(.J.) := READI;
18        END
19  END;
20
21  PROCEDURE CHECKUPPER;
22  VAR ISUPPER : INTEGER;
23  BEGIN
24    ISUPPER := 1;
25
26    FOR I := 1 TO N DO
27      FOR J := 1 TO N DO
28        IF I > J THEN
29          IF U(.I.)(.J.) != 0 THEN
30            ISUPPER := 0;
31
```

Problems Output Debug Console **Terminal** Ports

```
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/assign1.kpl
Program ASSIGN1
    Const MAX = 100
    Var N : Int
    Var U : Arr(100,Arr(100,Int))
    Var I : Int
    Var J : Int
    Procedure INPUT

    Procedure CHECKUPPER
        Var ISUPPER : Int


C:\Users\vuvan\Downloads\SymTab_Incompleted>
```

2. Assign 2

tests > assign2.kpl

```
1   PROGRAM ASSIGN2;
2   VAR A : INTEGER;
3       B : INTEGER;
4       SUM : INTEGER;
5
6   PROCEDURE INPUTANDSUM;
7   BEGIN
8     CALL WRITEC('A');
9     CALL WRITEC('=');
10    A := READI;
11
12    CALL WRITEC('B');
13    CALL WRITEC('=');
14    B := READI;
15
16    SUM := A + B;
17
18    CALL WRITEI(SUM);
19    CALL WRITELN
20  END;
21
22  BEGIN
23    CALL INPUTANDSUM
24  END.
```

Problems Output Debug Console **Terminal** Ports

```
C:\Users\vuvan\Downloads\SymTab_Incompleted>symtab.exe tests/assign2.kpl
Program ASSIGN2
    Var A : Int
    Var B : Int
    Var SUM : Int
    Procedure INPUTANDSUM
```