# BÁO CÁO THỰC HÀNH
# XÂY DỰNG CHƯƠNG TRÌNH DỊCH
# TUẦN 5: TYPE CHECKING BUỔI 2

Họ và tên: Bùi Quang Hưng

Mã số sinh viên: 20225849

## I. Tổng quan về Type checking

Type Checking (Kiểm tra kiểu) là một giai đoạn quan trọng trong phân tích ngữ nghĩa của trình biên dịch, đảm bảo rằng các phép toán và biểu thức trong chương trình sử dụng đúng kiểu dữ liệu. Trong ngôn ngữ KPL, type checking được thực hiện trong quá trình phân tích cú pháp (parsing) để phát hiện sớm các lỗi về kiểu dữ liệu.

- Các kiểu dữ liệu trong KPL:
  - Kiểu cơ bản (Basic Type): INTEGER, CHAR
  - Kiểu mảng (Array Type): ARRAY[size] OF element_type
  - Kiểu do người dùng định nghĩa: Thông qua khai báo TYPE
- Mục tiêu của Type checking:
  - Đảm bảo sự nhất quán kiểu trong khai báo hằng, biến, tham số
  - Kiểm tra tính tương thích kiểu trong biểu thức, lệnh gán, điều kiện
  - Xác minh tính đúng đắn của các phép toán số học và logic
  - Phát hiện lỗi sử dụng mảng (số chiều, kiểu chỉ số)

## II. Mô tả hàm compileArgument và ứng dụng của hàm trong kiểm tra tương ứng kiểu giữa khai báo và sử dụng hàm, thủ tục

### 1. Mục đích

- Hàm compileArgument(Object* param) phân tích cú pháp và kiểm tra kiểu cho một tham số thực sự khi gọi hàm/thủ tục, đảm bảo tính tương thích với tham số hình thức tương ứng.

### 2. Tham số đầu vào

- param: Con trỏ đến đối tượng tham số hình thức (Object*) trong định nghĩa hàm/thủ tục

### 3. Cơ chế hoạt động:

- Kiểm tra tồn tại tham số hình thức:
  - Nếu param == NULL: Báo lỗi ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
  - Nguyên nhân: Số lượng tham số thực sự nhiều hơn tham số hình thức

- Phân loại tham số theo cách truyền:
  - Nếu là tham biến (PARAM_REFERENCE):
    - Kiểm tra token tiếp theo phải là định danh (TK_IDENT)
    - Gọi compileLValue() để đảm bảo tham số thực là biến/tham số/hàm hiện tại (có thể gán được)
    - Nếu không phải định danh: Báo lỗi ERR_TYPE_INCONSISTENCY
    - Lý do: Tham biến cần địa chỉ thật để có thể thay đổi giá trị, không thể là hằng số hay biểu thức tạm
  - Nếu là tham trị (PARAM_VALUE):
    - Gọi compileExpression() để phân tích biểu thức bất kỳ
    - Cho phép truyền hằng số, biến, hoặc biểu thức phức tạp
- Kiểm tra tương đương kiểu:
  - Gọi checkTypeEquality(argType, param->paramAttrs->type)
  - So sánh kiểu dữ liệu giữa tham số thực sự và tham số hình thức
  - Với kiểu mảng: kiểm tra cả kích thước và kiểu phần tử đệ quy

## 4. Ứng dụng trong kiểm tra tương ứng kiểu giữa khai báo và sử dụng hàm, thủ tục:

- Kiểm tra tính nhất quán về kiểu dữ liệu:
  - Khi hàm được khai báo với tham số kiểu INTEGER,
    hàm compileArgument() đảm bảo tại điểm gọi hàm, giá trị truyền vào cũng phải có kiểu INTEGER
  - Sử dụng checkTypeEquality() để so sánh kiểu của tham số thực (argType) với kiểu tham số hình thức (param->paramAttrs->type)
  - Ví dụ: FUNCTION F(N : INTEGER) thì F('a') sẽ báo lỗi vì CHAR ≠ INTEGER

- Đảm bảo tính đúng đắn của tham biến (pass by reference):
  - Tham biến cho phép hàm/thủ tục thay đổi giá trị của biến bên ngoài, do đó cần địa chỉ thực
  - Hàm kiểm tra param->paramAttrs->kind == PARAM_REFERENCE rồi
    gọi compileLValue() để xác minh tham số thực là LValue
  - compileLValue() sẽ gọi tiếp checkDeclaredLValueIdent() để đảm bảo định danh thuộc loại OBJ_VARIABLE, OBJ_PARAMETER, hoặc OBJ_FUNCTION (chỉ hàm hiện tại)
  - Ngăn chặn các trường hợp sai: truyền hằng số (CONST C = 10), biểu thức (A + B), hoặc literal (10, 'a') cho tham biến
  - Ví dụ: PROCEDURE P(VAR X : INTEGER) thì CALL P(10) hoặc CALL P(C1) (C1 là hằng) đều bị báo lỗi
- Phát hiện sớm lỗi kiểu trong quá trình biên dịch:
  - Thay vì để lỗi xảy ra khi chạy (runtime), kiểm tra kiểu tại compile-time giúp phát hiện ngay
  - Hàm compileArgument() được gọi trong compileArguments() cho từng tham số, tạo lớp kiểm tra chi tiết

- Mỗi lần gọi hàm/thủ tục đều được validate đầy đủ trước khi sinh mã
- Giảm thiểu lỗi logic do truyền sai kiểu dữ liệu

- Xử lý kiểu phức tạp (mảng, kiểu người dùng định nghĩa):
  - Với tham số kiểu mảng, checkTypeEquality() kiểm tra đệ quy cả arraySize và elementType
  - Đảm bảo ARRAY(. 10 .) OF INTEGER không tương thích với ARRAY(. 5 .) OF INTEGER hoặc ARRAY(. 10 .) OF CHAR
  - Với kiểu do người dùng định nghĩa (TYPE T = INTEGER), hàm truy xuất actualType để so sánh kiểu gốc
  - Tạo tính nhất quán chặt chẽ trong hệ thống kiểu của ngôn ngữ KPL

# III. Mô tả hàm compileArgument và ứng dụng của hàm trong kiểm tra tương ứng kiểu giữa khai báo và sử dụng hàm, thủ tục

## 1. Mục đích

- Hàm compileArguments (Object* paramList) phân tích cú pháp và kiểm tra toàn bộ danh sách tham số khi gọi hàm/thủ tục, đảm bảo số lượng, thứ tự và kiểu đều khớp với định nghĩa.

## 2. Tham số đầu vào

- paramList: Con trỏ đến danh sách liên kết các tham số hình thức (ObjectNode*)

## 3. Cơ chế hoạt động:

- Phân tích cú pháp danh sách tham số:
  - Trường hợp 1: Có cặp ngoặc đơn SB_LPAR
    - Đọc dấu mở ngoặc "(" bằng eat(SB_LPAR)
    - Kiểm tra tham số đầu tiên:
      - Nếu paramList == NULL: Báo lỗi vì có tham số thực nhưng không có tham số hình thức
      - Nếu có: Gọi compileArgument(node->object) và chuyển sang tham số tiếp theo (node = node->next)
    - Vòng lặp xử lý các tham số còn lại:
      - Mỗi lần gặp dấu phẩy , (SB_COMMA): Xử lý tham số tiếp theo
      - Nếu đã hết tham số hình thức (node == NULL) nhưng còn tham số thực: Báo lỗi ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
    - Sau vòng lặp:
      - Kiểm tra node != NULL: Nếu còn tham số hình thức chưa được truyền → Báo lỗi
    - Đọc dấu đóng ngoặc ) bằng eat(SB_RPAR)
  - Trường hợp 2: Không có ngoặc đơn (thuộc FOLLOW set)
    - Khi token tiếp theo là toán tử (SB_PLUS, SB_TIMES), dấu chấm phẩy (SB_SEMICOLON), từ khóa (KW_END, KW_THEN), v.v.

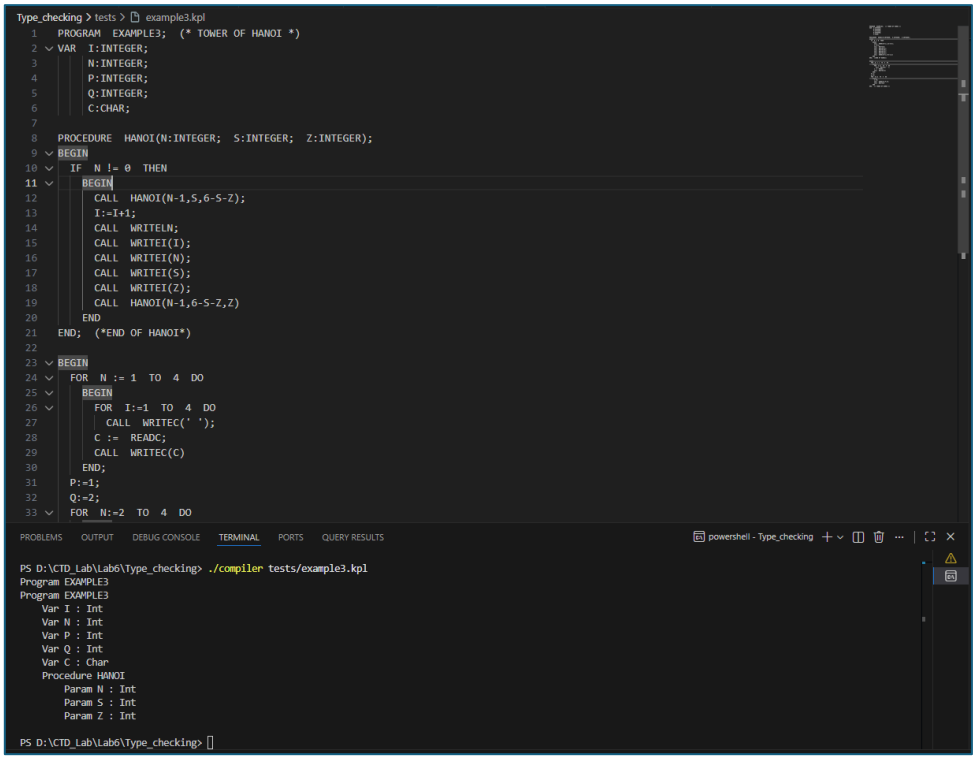
    - Kiểm tra paramList != NULL: Nếu còn tham số hình thức → Báo lỗi ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
    - Ý nghĩa: Trong KPL, hàm/thủ tục không có tham số được gọi không cần ngoặc đơn
- Kiểm tra tính nhất quán toàn diện
  - Số lượng: Số tham số thực = số tham số hình thức
  - Thứ tự: Tham số thứ i phải tương ứng với tham số hình thức thứ i
  - Kiểu: Mỗi cặp tham số phải có kiểu tương đương (qua compileArgument)

## 4. Ứng dụng trong kiểm tra tương ứng kiểu giữa khai báo và sử dụng hàm, thủ tục:

- Kiểm soát số lượng tham số:
  - Hàm duyệt song song danh sách tham số hình thức (paramList) và tham số thực (từ mã nguồn)
  - Sử dụng con trỏ node để theo dõi vị trí trong danh sách tham số hình thức
  - Nếu node == NULL mà vẫn còn tham số thực (có SB_COMMA tiếp theo): Lỗi "quá nhiều tham số"
  - Nếu hết tham số thực (gặp SB_RPAR) mà node != NULL: Lỗi "thiếu tham số"
  - Ví dụ: PROCEDURE P(A:INTEGER; B:CHAR) bắt buộc phải gọi với đúng 2 tham số
- Đảm bảo thứ tự và vị trí đúng:
  - Danh sách tham số hình thức có thứ tự cố định: F(P1:INTEGER; P2:CHAR; P3:INTEGER)
  - Hàm compileArguments() xử lý tuần tự từ trái sang phải, gọi compileArgument(node->object) cho từng vị trí
  - Tham số thực thứ nhất so sánh với tham số hình thức thứ nhất, thứ hai với thứ hai, v.v.
  - Điều này đảm bảo: F(10, 'a', 20) đúng nhưng F('a', 10, 20) sẽ lỗi tại tham số thứ 1 (CHAR ≠ INTEGER)
  - Tránh nhầm lẫn về ý nghĩa của từng tham số theo vị trí
- Phối hợp với compileArgument() để kiểm tra từng cặp:
  - compileArguments() đóng vai trò điều phối, compileArgument() thực hiện kiểm tra chi tiết
  - Với mỗi tham số, compileArguments() truyền node->object (tham số hình thức) vào compileArgument()
  - compileArgument() sẽ kiểm tra kiểu, cách truyền (tham trị/tham biến) cho từng cặp
  - Tạo cơ chế kiểm tra đa lớp: compileArguments() kiểm tra cấu trúc tổng thể, compileArgument() kiểm tra chi tiết từng phần tử
  - Ví dụ: PROCEDURE P(A:INTEGER; VAR B:CHAR) gọi bằng CALL P(10, C) → compileArguments() đảm bảo có 2 tham số, compileArgument() kiểm tra tham số 1 là INTEGER và tham số 2 là CHAR + Lvalue
- Xử lý đặc thù cú pháp KPL:

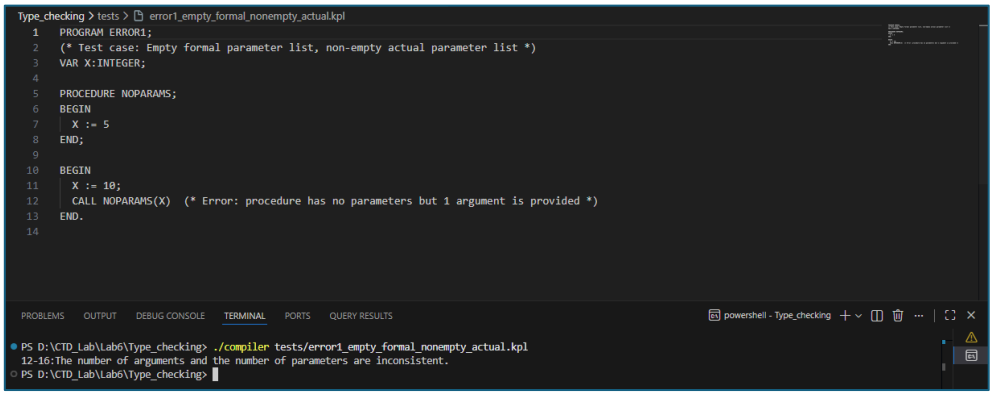
- KPL cho phép hàm/thủ tục không tham số gọi không cần (): N := GetMax thay vì N := GetMax()
- Hàm kiểm tra FOLLOW set để phân biệt trường hợp không có tham số vs. lỗi cú pháp
- Nếu lookAhead thuộc FOLLOW (toán tử, dấu ;, từ khóa) và paramList == NULL: Hợp lệ (gọi hàm không tham số)
- Nếu lookAhead thuộc FOLLOW nhưng paramList != NULL: Lỗi thiếu tham số
- Tạo tính linh hoạt về cú pháp mà vẫn đảm bảo kiểm tra chặt chẽ

# IV. Kết quả thực hiện với Example3.kpl không lỗi



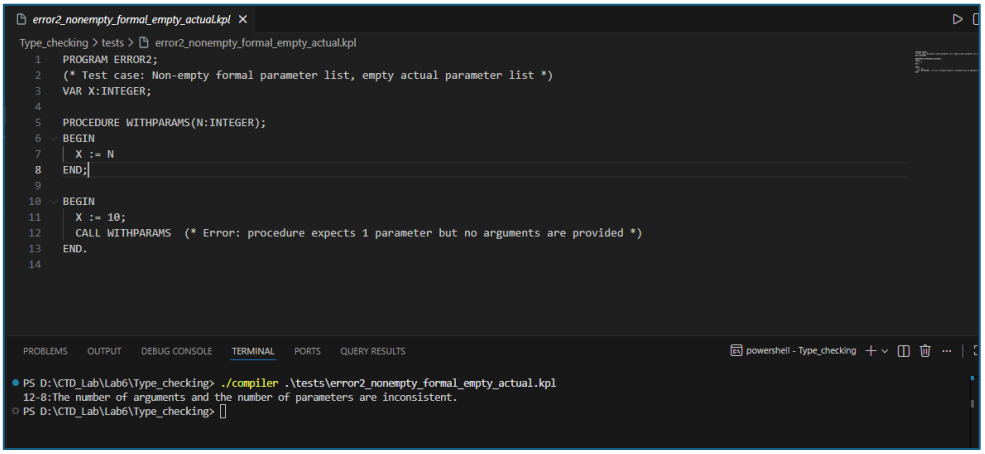

# V. Thay đổi các ví dụ đã cho để gây các lỗi

## 1. Danh sách tham số hình thức rỗng, danh sách tham số thực sự có ít nhất 1 phần tử



- Nguyên nhân: Sửa example 4, hàm SUM được khai báo không nhận tham số nào → paramList = NULL. Tại điểm gọi, người dùng truyền giá trị 10 vào hàm
- Luồng xử lý:
  - Parser gặp WRITEI(SUM(10)), bắt đầu phân tích biểu thức
  - Trong compileFactor(), nhận ra SUM là hàm → Gọi compileArguments(obj->funcAttrs->paramList)
  - paramList của hàm SUM là NULL (không có tham số hình thức)
  - compileArguments() kiểm tra token tiếp theo: SB_LPAR (dấu mở ngoặc)
  - Vào khối case SB_LPAR:, thực hiện eat(SB_LPAR)
  - Kiểm tra điều kiện: if (node == NULL) → TRUE (vì paramList = NULL)
  - Báo lỗi ngay lập tức: Có tham số thực nhưng không có tham số hình thức
- Hàm phát hiện: compileArguments(). Khi có dấu mở ngoặc ( nhưng paramList == NULL
- Thông báo lỗi: ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
- Vị trí phát hiện: Trong khối case SB_LPAR:, ngay sau lệnh eat(SB_LPAR), dòng kiểm tra if (node == NULL)

## 2. Danh sách tham số thực sự rỗng, danh sách tham số hình thức có ít nhất 1 phần tử

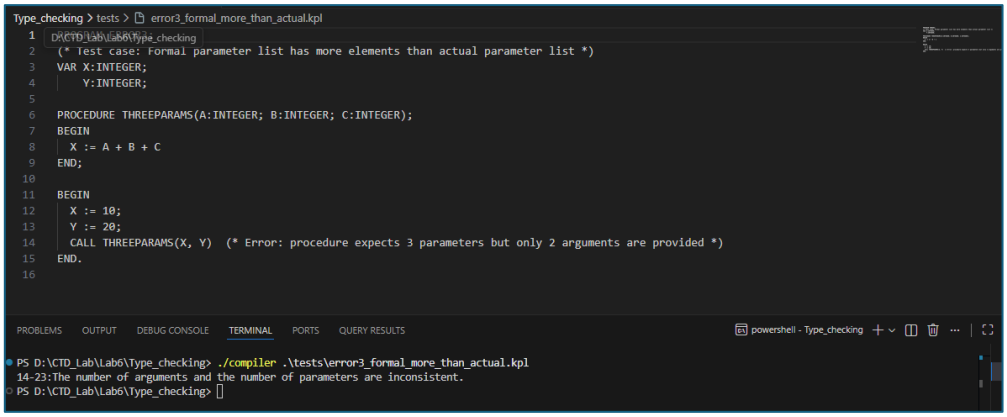



- Nguyên nhân: Sửa example2, hàm F yêu cầu 1 tham số kiểu INTEGER → paramList chứa 1 node (tham số N), tại điểm gọi, không có dấu ngoặc () và không có tham số nào được truyền
- Luồng xử lý:
  - Parser gặp WRITEI(F), bắt đầu phân tích biểu thức
  - Trong compileFactor(), nhận ra F là hàm → Gọi compileArguments(obj->funcAttrs->paramList)
  - paramList của hàm F có 1 node (tham số N)
  - compileArguments() kiểm tra token tiếp theo: SB_RPAR (dấu đóng ngoặc của WRITEI)
  - SB_RPAR thuộc FOLLOW set → Không phải trường hợp có ngoặc đơn

- Vào khối xử lý FOLLOW set (các case SB_TIMES, SB_SLASH, ...)
- Kiểm tra điều kiện: if (paramList != NULL) → TRUE (còn tham số N chưa được truyền)
- Báo lỗi: Thiếu tham số

- Hàm phát hiện: compileArguments(), token thuộc FOLLOW set nhưng còn tham số hình thức chưa được truyền
- Thông báo lỗi: ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
- Vị trí phát hiện: Trong khối xử lý FOLLOW set, dòng kiểm tra if (paramList != NULL)

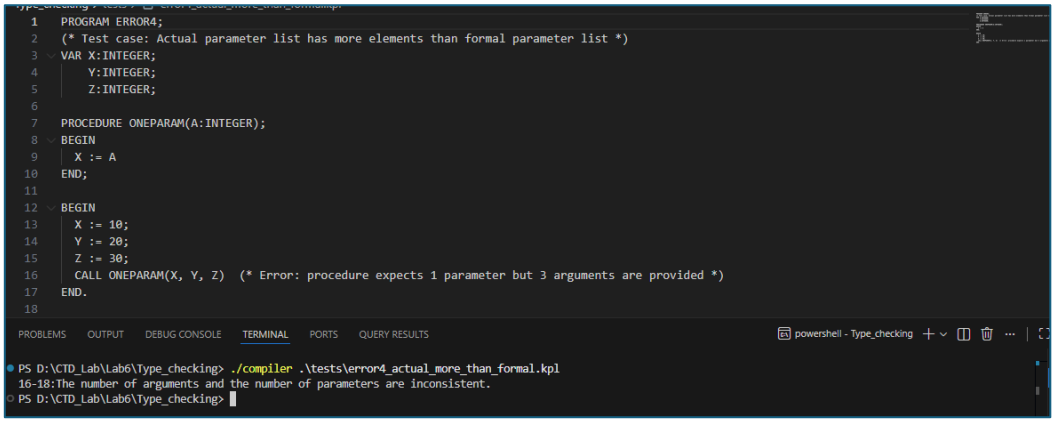
## 3. Danh sách tham số hình thức và tham số thực sự thực sự chứa tối thiểu 1 phần tử nhưng danh sách tham số hình thức chứa nhiều phần tử hơn danh sách tham số thực sự




- Nguyên nhân: Sửa example3, thủ tục HANOI được khai báo với 3 tham số (N, S, Z), tại điểm gọi, chỉ truyền 2 tham số (N, P)
- Luồng xử lý
  - Parser gặp CALL HANOI(N,P) → Gọi compileCallSt()
  - compileCallSt() kiểm tra HANOI là thủ tục → Gọi compileArguments(proc->procAttrs->paramList)
  - paramList có 3 node (N, S, Z), khởi tạo node = paramList
  - compileArguments() gặp SB_LPAR, vào khối case SB_LPAR:
  - Xử lý tham số 1: compileArgument(node->object) cho N ✓, node = node->next
  - Gặp SB_COMMA, vào vòng lặp while
  - Xử lý tham số 2: compileArgument(node->object) cho S ✓, node = node->next
  - Token tiếp theo là SB_RPAR (không còn SB_COMMA) → Thoát vòng lặp
  - Kiểm tra sau vòng lặp: if (node != NULL) → TRUE (còn tham số Z chưa được truyền)
  - Báo lỗi: Thiếu tham số
- Hàm phát hiện: compileArguments(), sau khi xử lý hết tham số thực nhưng node != NULL (còn tham số hình thức)

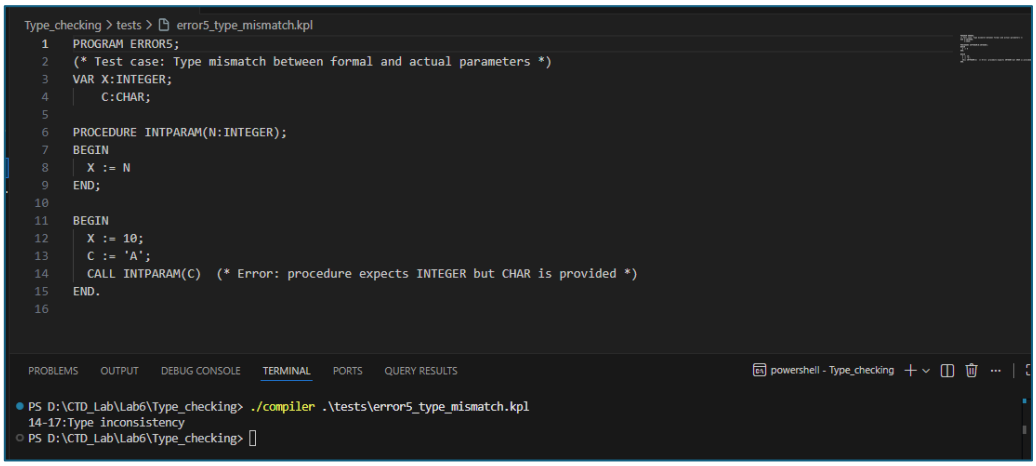
- Thông báo lỗi: ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
- Vị trí phát hiện: Trong khối case SB_LPAR:, sau vòng lặp while, dòng kiểm tra if (node != NULL)

## 4. Danh sách tham số hình thức và tham số thực sự thực sự chứa tối thiểu 1 phần tử nhưng danh sách tham số thực sự chứa nhiều phần tử hơn danh sách tham số hình thức



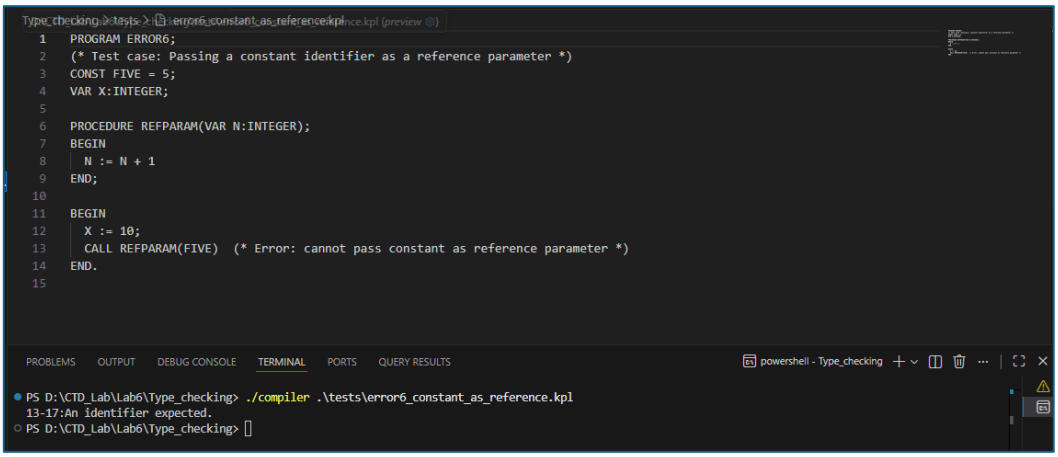

- Nguyên nhân: Sửa example6, hàm F được khai báo với 2 tham số (P1, P2), tại điểm gọi, truyền 3 tham số (C3, V3, V4)
- Luồng xử lý:
  - Parser gặp F(C3, V3, V4) trong biểu thức gán
  - compileFactor() nhận ra F là hàm → Gọi compileArguments(obj->funcAttrs->paramList)
  - paramList có 2 node (P1, P2), khởi tạo node = paramList
  - compileArguments() gặp SB_LPAR, vào khối case SB_LPAR:
  - Xử lý tham số 1: compileArgument(node->object) cho P1 với C3 ✓, node = node->next
  - Gặp SB_COMMA, vào vòng lặp while
  - Xử lý tham số 2: compileArgument(node->object) cho P2 với V3 ✓, node = node->next
  - Gặp SB_COMMA lần nữa, tiếp tục vòng lặp
  - Kiểm tra: if (node == NULL) → TRUE (đã hết tham số hình thức)
  - Báo lỗi: Có tham số thực thừa (V4) nhưng không có tham số hình thức tương ứng
- Hàm phát hiện: compileArguments(), trong vòng lặp while (SB_COMMA), gặp dấu phẩy nhưng node == NULL
- Thông báo lỗi: ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY
- Vị trí phát hiện: Trong khối case SB_LPAR:, vòng lặp while (lookAhead->tokenType == SB_COMMA), dòng kiểm tra if (node == NULL)

## 5. Không tương ứng kiểu giữa tham số hình thức và tham số thực sự



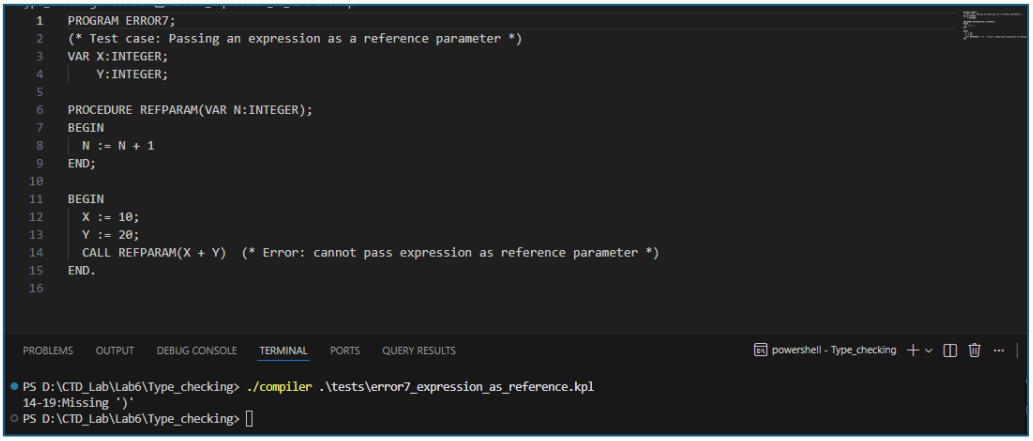

- Nguyên nhân: Sửa example3, tham số hình thức thứ 3 (Z) được khai báo kiểu INTEGER, tại điểm gọi, tham số thực thứ 3 là biến C có kiểu CHAR (khai báo: C:CHAR)
- Luồng xử lý:
  - Parser gặp CALL HANOI(N,P,C) → Gọi compileCallSt()
  - compileCallSt() kiểm tra HANOI là thủ tục → Gọi compileArguments(proc->procAttrs->paramList)
  - compileArguments() xử lý từng tham số:
    - Tham số 1: N (INTEGER) ✓ khớp với tham số hình thức N:INTEGER
    - Tham số 2: P (INTEGER) ✓ khớp với tham số hình thức S:INTEGER
    - Tham số 3: Gọi compileArgument(node->object) với node->object là tham số hình thức Z:INTEGER
  - Trong compileArgument():
    - Tham số hình thức không phải tham biến (là PARAM_VALUE) → Gọi compileExpression()
    - compileExpression() → compileFactor() → Nhận ra C là biến kiểu CHAR
    - Trả về argType = charType (kiểu CHAR)
  - Gọi checkTypeEquality(argType, param->paramAttrs->type): So sánh: charType (TP_CHAR) vs intType (TP_INT)
  - Trong checkTypeEquality(): Kiểm tra: type1->typeClass != type2->typeClass → TRUE (TP_CHAR ≠ TP_INT)
  - Báo lỗi: Kiểu không nhất quán
- Hàm phát hiện: checkTypeEquality() Được gọi từ: compileArgument()
- Vị trí phát hiện: Trong checkTypeEquality(), dòng kiểm tra if (type1->typeClass != type2->typeClass)

## 6. Truyền tham biến bằng một hằng biểu diễn bởi định danh




- Nguyên nhân: Sửa example6, tham số P2 là tham biến (PARAM_REFERENCE), yêu cầu truyền một LValue (biến có thể gán), C1 là hằng biểu diễn bởi định danh: Được khai báo với tên C1 (CONST C1 = 'a') → Không thể thay đổi giá trị, tham biến cần địa chỉ thực để có thể sửa đổi, hằng số không đáp ứng được yêu cầu này
- Luồng xử lý:
  - Parser gặp F(C3, C1) trong biểu thức gán
  - compileFactor() nhận ra F là hàm → Gọi compileArguments(obj->funcAttrs->paramList)
  - compileArguments() xử lý:
    - Tham số 1: C3 (INTEGER) ✓ đúng kiểu với P1:INTEGER
    - Tham số 2: Gọi compileArgument(node->object) với node->object là tham số hình thức P2
  - Trong compileArgument():
    - Kiểm tra: param->paramAttrs->kind == PARAM_REFERENCE → TRUE
    - Kiểm tra: lookAhead->tokenType == TK_IDENT → TRUE (token là C1)
    - Gọi compileLValue() để xử lý C1
  - Trong compileLValue():
    - Đọc token C1: eat(TK_IDENT)
    - Gọi checkDeclaredLValueIdent("C1") để kiểm tra
  - Trong checkDeclaredLValueIdent():
    - Tìm đối tượng: obj = lookupObject("C1") → Tìm thấy
    - Kiểm tra obj->kind: Là OBJ_CONSTANT (hằng số)
    - Duyệt qua switch (obj->kind): Không phải OBJ_VARIABLE, OBJ_PARAMETER, hay OBJ_FUNCTION => Rơi vào khối default
  - Báo lỗi: Định danh không hợp lệ cho LValue (không thể là hằng số)
- Hàm phát hiện: checkDeclaredLValueIdent(). Được gọi từ : compileLValue() ← compileArgument()
- Thông báo lỗi: ERR_INVALID_IDENT
- Vị trí phát hiện: Trong checkDeclaredLValueIdent(), khối default của switch (obj->kind)

## 7. Truyền tham biến bằng một biểu thức



```
PROGRAM ERROR7;
(* Test case: Passing an expression as a reference parameter *)
VAR X:INTEGER;
    Y:INTEGER;

PROCEDURE REFPARAM(VAR N:INTEGER);
BEGIN
  N := N + 1
END;

BEGIN
  X := 10;
  Y := 20;
  CALL REFPARAM(X + Y)  (* Error: cannot pass expression as reference parameter *)
END.
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS        powershell - Type_checking

PS D:\CTD_Lab\Lab6\Type_checking> ./compiler .\tests\error7_expression_as_reference.kpl
14-19:Missing ')'
PS D:\CTD_Lab\Lab6\Type_checking>
```

- Nguyên nhân: Sửa example6, tham số P2 là tham biến, yêu cầu một định danh biến (token phải là TK_IDENT), giá trị truyền vào là (X + Y) * 2 → Đây là biểu thức số học bắt đầu bằng (, token đầu tiên là SB_LPAR, không phải định danh biến, biểu thức tạo giá trị tạm thời trong thanh ghi/stack, không có địa chỉ cố định để tham chiếu
- Luồng xử lý:
  - Parser gặp F(C3, (X + Y)*2) trong biểu thức gán
  - compileFactor() nhận ra F là hàm → Gọi compileArguments(obj->funcAttrs->paramList)
  - compileArguments() xử lý:
    - Tham số 1: C3 (INTEGER) ✓ đúng kiểu với P1:INTEGER
    - Gặp dấu phẩy , → Chuyển sang tham số thứ 2
    - Gọi compileArgument(node->object) với node->object là tham số hình thức P2
  - Trong compileArgument():
    - Kiểm tra: param->paramAttrs->kind == PARAM_REFERENCE → TRUE
    - Kiểm tra: lookAhead->tokenType == TK_IDENT → FALSE (token là SB_LPAR)
    - Điều kiện không thỏa mãn → Rơi vào nhánh else
  - Báo lỗi ngay lập tức: Kiểu không nhất quán (token không phải định danh biến)
- Hàm phát hiện: compileArgument(), token đầu tiên của tham số là SB_LPAR thay vì TK_IDENT
- Thông báo lỗi: ERR_TYPE_INCONSISTENCY
- Vị trí phát hiện: Trong compileArgument(), nhánh else của điều kiện if (lookAhead->tokenType == TK_IDENT)