

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



BÀI THỰC HÀNH SỐ 6: KIỂM TRA KIỂU - Phần 2
HỌC PHẦN: THỰC HÀNH XÂY DỰNG CHƯƠNG TRÌNH DỊCH

Mã lớp học: 161629
GVHD: Nguyễn Thị Thu Hương
TA: Đỗ Gia Huy
Sinh viên thực hiện: Trịnh Hữu An - 20225593

- Hà Nội, tháng 12 năm 2025 -

Mục lục

1	Mô tả các hàm <code>compileArgument</code> và <code>compileArguments</code>	4
1.1	Hàm <code>compileArgument</code>	4
1.1.1	Tham số	4
1.1.2	Chức năng	4
1.1.3	Code	5
1.2	Hàm <code>compileArguments</code>	6
1.2.1	Tham số	6
1.2.2	Chức năng	6
1.2.3	Code	7
2	Ứng dụng trong kiểm tra tương ứng kiểu	8
2.1	Quy trình kiểm tra	8
2.2	Các trường hợp đặc biệt	9
2.2.1	Tham số tham chiếu (VAR)	9
2.2.2	Tham số giá trị	9
3	Kết quả thực hiện với <code>EXAMPLE3.KPL</code>	9
4	Các test case gây lỗi	10
4.1	Test 1: Danh sách tham số hình thức rỗng, danh sách tham số thực sự có ít nhất 1 phần tử	10
4.1.1	Code	10
4.1.2	Phân tích lý do gây lỗi	10
4.1.3	Kết quả	11
4.2	Test 2: Danh sách tham số thực sự rỗng, danh sách tham số hình thức có ít nhất 1 phần tử	11
4.2.1	Code	11
4.2.2	Phân tích lý do gây lỗi	11
4.2.3	Kết quả	12
4.3	Test 3: Danh sách tham số hình thức chứa nhiều phần tử hơn danh sách tham số thực sự	12
4.3.1	Code	12
4.3.2	Phân tích lý do gây lỗi	12
4.3.3	Kết quả	13
4.4	Test 4: Danh sách tham số thực sự chứa nhiều phần tử hơn danh sách tham số hình thức	13
4.4.1	Code	13
4.4.2	Phân tích lý do gây lỗi	13
4.4.3	Kết quả	14
4.5	Test 5: Không tương ứng kiểu giữa tham số hình thức và tham số thực sự	14
4.5.1	Code	14
4.5.2	Phân tích lý do gây lỗi	14
4.5.3	Kết quả	15
4.6	Test 6: Truyền tham biến bằng một hằng biểu diễn bởi định danh	15
4.6.1	Code	15
4.6.2	Phân tích lý do gây lỗi	15

4.6.3	Kết quả	16
4.7	Test 7: Truyền tham biến bằng một biểu thức	16
4.7.1	Code	16
4.7.2	Phân tích lý do gây lỗi	16
4.7.3	Kết quả	17

1 Mô tả các hàm `compileArgument` và `compileArguments`

1.1 Hàm `compileArgument`

Hàm `compileArgument(Object* param)` có nhiệm vụ biên dịch và kiểm tra một đối số (argument) đơn lẻ khi gọi hàm/thủ tục.

1.1.1 Tham số

- `param`: Con trỏ đến đối tượng tham số (parameter) tương ứng từ khai báo hàm/thủ tục

1.1.2 Chức năng

Hàm thực hiện các bước sau:

1. **Kiểm tra tham số NULL**: Nếu `param == NULL`, báo lỗi không tương ứng số lượng tham số và đối số.
2. **Xử lý tham số tham chiếu (PARAM_REFERENCE)**:
 - Chỉ chấp nhận biến, tham số, hoặc tên hàm (lvalue)
 - KHÔNG chấp nhận: hằng số, biểu thức, kết quả hàm
 - Nếu là `TK_IDENT`:
 - Kiểm tra xem có phải lời gọi hàm không (có `SB_LPAR` sau đó) - báo lỗi nếu đúng
 - Kiểm tra xem có phải biểu thức không (có toán tử sau đó) - báo lỗi nếu đúng
 - Nếu là lvalue hợp lệ: sử dụng `checkDeclaredLValueIdent` để kiểm tra và lấy kiểu
 - Nếu không phải `TK_IDENT`: báo lỗi và parse như biểu thức
3. **Xử lý tham số giá trị (PARAM_VALUE)**:
 - Chấp nhận bất kỳ biểu thức nào (số, ký tự, biến, hàm, ...)
 - Gọi `compileExpression()` để parse và lấy kiểu
4. **Kiểm tra tương ứng kiểu**:
 - So sánh kiểu của đối số (`argType`) với kiểu của tham số (`param->paramAttrs->type`)
 - Sử dụng `compareType()`: trả về 1 nếu khớp, 0 nếu không khớp
 - Báo lỗi `ERR_TYPE_INCONSISTENCY` nếu kiểu không khớp

1.1.3 Code

```
1 void compileArgument(Object* param) {
2     // TODO: parse an argument, and check type consistency
3     if (param == NULL) {
4         error(ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY,
5             currentToken->lineNo, currentToken->colNo);
6         return;
7     }
8
9     Type* argType;
10    Object* obj;
11
12    if (param->paramAttrs->kind == PARAM_REFERENCE) {
13        // Tham bien: chi chap nhan lvalue (bien, tham so, ten ham)
14        if (lookAhead->tokenType == TK_IDENT) {
15            eat(TK_IDENT);
16            if (lookAhead->tokenType == SB_LPAR) {
17                // Function call - khong the truyen ket qua ham cho tham
18                // bien
19                error(ERR_TYPE_INCONSISTENCY,
20                    currentToken->lineNo, currentToken->colNo);
21                obj = checkDeclaredFunction(currentToken->string);
22                compileArguments(obj->funcAttrs->paramList);
23                argType = obj->funcAttrs->returnType;
24            } else {
25                // Kiem tra xem co phai bieu thuc khong (co toan tu theo
26                // sau)
27                if (lookAhead->tokenType == SB_PLUS ||
28                    lookAhead->tokenType == SB_MINUS ||
29                    lookAhead->tokenType == SB_TIMES ||
30                    lookAhead->tokenType == SB_SLASH) {
31                    error(ERR_TYPE_INCONSISTENCY,
32                        currentToken->lineNo, currentToken->colNo);
33                    argType = compileExpression();
34                } else {
35                    // Kiem tra xem co phai lvalue hop le
36                    obj = checkDeclaredLValueIdent(currentToken->string);
37                    switch (obj->kind) {
38                        case OBJ_VARIABLE:
39                            argType = obj->varAttrs->type;
40                            if (argType->typeClass == TP_ARRAY)
41                                argType = compileIndexes(argType);
42                            break;
43                        case OBJ_PARAMETER:
44                            argType = obj->paramAttrs->type;
45                            break;
46                        case OBJ_FUNCTION:
47                            argType = obj->funcAttrs->returnType;
48                            break;
49                        default:
```

```

48         argType = NULL;
49         break;
50     }
51 }
52 }
53 } else {
54     // Không phải identifier - không thể truyền tham biến
55     error(ERR_TYPE_INCONSISTENCY,
56         lookahead->lineNo, lookahead->colNo);
57     argType = compileExpression();
58 }
59 } else {
60     // Tham số giá trị: chấp nhận bất kỳ biểu thức nào
61     argType = compileExpression();
62 }
63
64 // Kiểm tra tương ứng kiểu giữa đối số và tham số
65 if (argType != NULL && param->paramAttrs->type != NULL) {
66     if (compareType(argType, param->paramAttrs->type) == 0) {
67         error(ERR_TYPE_INCONSISTENCY,
68             currentToken->lineNo, currentToken->colNo);
69     }
70 }
71 }

```

1.2 Hàm compileArguments

Hàm `compileArguments(ObjectNode* paramList)` có nhiệm vụ biên dịch danh sách các đối số khi gọi hàm/thủ tục và kiểm tra tính nhất quán với danh sách tham số.

1.2.1 Tham số

- `paramList`: Con trỏ đến danh sách liên kết các tham số từ khai báo

1.2.2 Chức năng

Hàm thực hiện các bước sau:

1. Xử lý trường hợp có đối số (SB_LPAR):

- Parse dấu (
- Kiểm tra nếu `paramList == NULL` (quá nhiều đối số): báo lỗi và parse các đối số còn lại
- Lặp qua từng đối số:
 - Gọi `compileArgument()` cho đối số hiện tại
 - Di chuyển con trỏ đến tham số tiếp theo
 - Nếu còn dấu phẩy, tiếp tục với đối số tiếp theo
 - Nếu hết đối số nhưng còn tham số: báo lỗi

- Parse dấu)

2. Xử lý trường hợp không có đối số:

- Kiểm tra các token FOLLOW (,,), toán tử, ...)
- Nếu paramList != NULL (thiếu đối số): báo lỗi

3. Xử lý lỗi cú pháp: Báo lỗi nếu không thuộc các trường hợp trên

1.2.3 Code

```

1 void compileArguments(ObjectNode* paramList) {
2     //TODO: parse a list of arguments, check the consistency
3     //      of the arguments and the given parameters
4     ObjectNode* currentParam = paramList;
5
6     switch (lookAhead->tokenType) {
7     case SB_LPAR:
8         eat(SB_LPAR);
9         if (currentParam == NULL) {
10             // Qua nhieu doi so
11             error(ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY,
12                 currentToken->lineNo, currentToken->colNo);
13             compileExpression();
14             while (lookAhead->tokenType == SB_COMMA) {
15                 eat(SB_COMMA);
16                 compileExpression();
17             }
18             eat(SB_RPAR);
19             return;
20         }
21         compileArgument(currentParam->object);
22         currentParam = currentParam->next;
23
24         while (lookAhead->tokenType == SB_COMMA) {
25             eat(SB_COMMA);
26             if (currentParam == NULL) {
27                 // Qua nhieu doi so
28                 error(ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY,
29                     currentToken->lineNo, currentToken->colNo);
30                 compileExpression();
31                 while (lookAhead->tokenType == SB_COMMA) {
32                     eat(SB_COMMA);
33                     compileExpression();
34                 }
35                 eat(SB_RPAR);
36                 return;
37             }
38             compileArgument(currentParam->object);
39             currentParam = currentParam->next;
40         }

```

```

41
42     if (currentParam != NULL) {
43         // Thieu doi so
44         error(ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY,
45             currentToken->lineNo, currentToken->colNo);
46     }
47
48     eat(SB_RPAR);
49     break;
50
51     // Kiem tra FOLLOW set - khong co doi so
52     case SB_TIMES:
53     case SB_SLASH:
54     case SB_PLUS:
55     case SB_MINUS:
56     case KW_TO:
57     case KW_DO:
58     case SB_RPAR:
59     case SB_COMMA:
60     case SB_EQ:
61     case SB_NEQ:
62     case SB_LE:
63     case SB_LT:
64     case SB_GE:
65     case SB_GT:
66     case SB_RSEL:
67     case SB_SEMICOLON:
68     case KW_END:
69     case KW_ELSE:
70     case KW_THEN:
71         if (paramList != NULL) {
72             // Thieu doi so
73             error(ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY,
74                 currentToken->lineNo, currentToken->colNo);
75         }
76         break;
77     default:
78         error(ERR_INVALID_ARGUMENTS,
79             lookAhead->lineNo, lookAhead->colNo);
80 }
81 }

```

2 Ứng dụng trong kiểm tra tương ứng kiểu

2.1 Quy trình kiểm tra

Các hàm `compileArgument` và `compileArguments` hoạt động cùng nhau để đảm bảo tính nhất quán giữa khai báo và sử dụng hàm/thủ tục:

1. **Kiểm tra số lượng:** `compileArguments` đảm bảo số lượng đối số khớp với số lượng

tham số

2. **Kiểm tra từng đối số:** `compileArgument` được gọi cho mỗi cặp (tham số, đối số) để:
 - Parse đối số và xác định kiểu
 - Kiểm tra tính hợp lệ của đối số (đặc biệt với tham biến)
 - So sánh kiểu với tham số tương ứng
3. **Kiểm tra kiểu:** Sử dụng `compareType()` để so sánh kiểu của đối số và tham số

2.2 Các trường hợp đặc biệt

2.2.1 Tham số tham chiếu (VAR)

- **Chấp nhận:** Biến, tham số, tên hàm (lvalue)
- **Từ chối:** Hằng số, biểu thức, kết quả hàm
- **Cách kiểm tra:** Sử dụng `checkDeclaredLValueIdent` để đảm bảo đối số là lvalue hợp lệ

2.2.2 Tham số giá trị

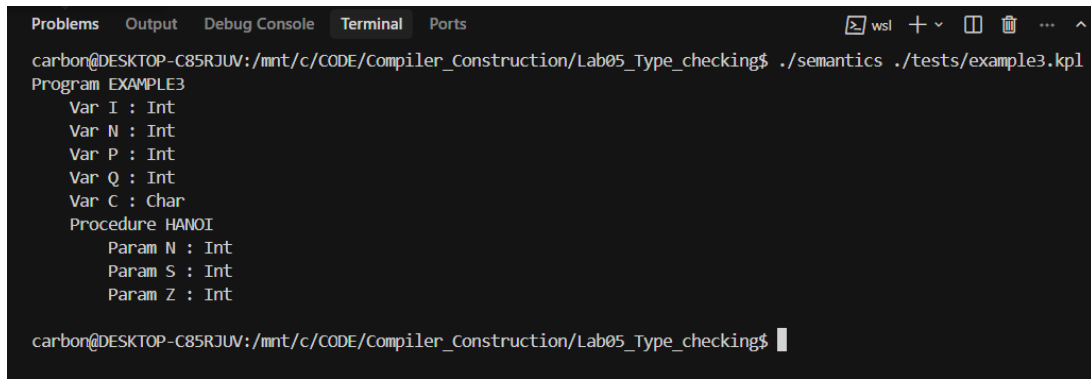
- **Chấp nhận:** Bất kỳ biểu thức nào
- **Cách kiểm tra:** Gọi `compileExpression()` để parse và lấy kiểu

3 Kết quả thực hiện với EXAMPLE3.KPL

File `example3.kpl` là chương trình Tower of Hanoi, sử dụng procedure `HANOI` với 3 tham số kiểu `INTEGER`. Chương trình này không có lỗi và được biên dịch thành công.

Procedure `HANOI` được khai báo với 3 tham số hình thức có kiểu `INTEGER`: `N`, `S`, và `Z`. Trong chương trình, có các lời gọi procedure này như `CALL HANOI(N-1,S,6-S-Z)` và `CALL HANOI(N,P,Q)`.

Khi compiler xử lý các lời gọi này, hàm `compileArguments` được gọi với danh sách tham số của procedure `HANOI`. Compiler kiểm tra số lượng đối số: với `CALL HANOI(N,P,Q)` có đúng 3 đối số tương ứng với 3 tham số. Sau đó, hàm `compileArgument` được gọi cho từng cặp (tham số, đối số): đối số `N` được so sánh với tham số `N:INTEGER`, đối số `P` với tham số `S:INTEGER`, và đối số `Q` với tham số `Z:INTEGER`. Tất cả các đối số đều có kiểu `INTEGER` (vì `N`, `P`, `Q` được khai báo là `N:INTEGER`, `P:INTEGER`, `Q:INTEGER`), nên hàm `compareType` trả về 1 (khớp kiểu) cho tất cả các cặp. Do đó, không có lỗi nào được phát hiện và chương trình được biên dịch thành công.



```
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$ ./semantics ./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

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$
```

Hình 1: Kết quả biên dịch example3.kpl không có lỗi

4 Các test case gây lỗi

4.1 Test 1: Danh sách tham số hình thức rỗng, danh sách tham số thực sự có ít nhất 1 phần tử

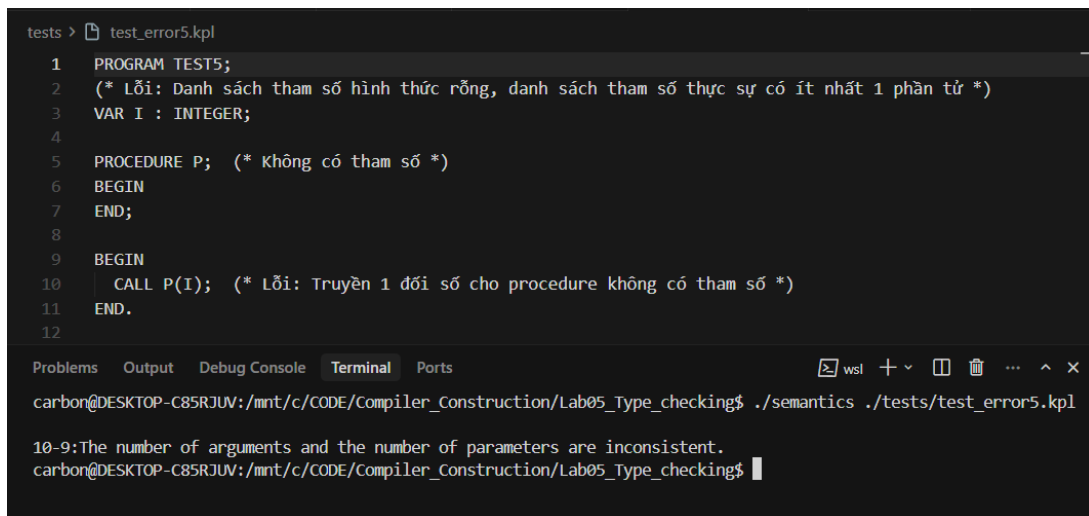
4.1.1 Code

```
1 PROGRAM TEST5;
2 (* Loi: Danh sach tham so hinh thuc rong,
3     danh sach tham so thuc su co it nhat 1 phan tu *)
4 VAR I : INTEGER;
5
6 PROCEDURE P; (* Khong co tham so *)
7 BEGIN
8 END;
9
10 BEGIN
11   CALL P(I); (* Loi: Truyen 1 doi so cho procedure
12              khong co tham so *)
13 END.
```

4.1.2 Phân tích lý do gây lỗi

Procedure P được khai báo không có tham số nào, do đó `paramList` của nó là `NULL`. Khi compiler gặp lời gọi `CALL P(I)`, hàm `compileArguments` được gọi với `paramList = NULL`. Compiler parse dấu mở ngoặc `SB_LPAR` và kiểm tra điều kiện `currentParam == NULL`, điều này đúng vì procedure không có tham số. Do đó, compiler phát hiện rằng có đối số được truyền vào nhưng procedure không có tham số nào để nhận, tạo ra lỗi không tương ứng giữa số lượng tham số và đối số. **Hàm phát hiện lỗi** là `compileArguments`, báo lỗi `ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY` với thông báo "The number of arguments and the number of parameters are inconsistent."

4.1.3 Kết quả



```
tests > test_error5.kpl
1 PROGRAM TEST5;
2 (* Lỗi: Danh sách tham số hình thức rỗng, danh sách tham số thực sự có ít nhất 1 phần tử *)
3 VAR I : INTEGER;
4
5 PROCEDURE P; (* Không có tham số *)
6 BEGIN
7 END;
8
9 BEGIN
10 CALL P(I); (* Lỗi: Truyền 1 đối số cho procedure không có tham số *)
11 END.
12
```

Problems Output Debug Console Terminal Ports

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$./semantics ./tests/test_error5.kpl

10-9:The number of arguments and the number of parameters are inconsistent.

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$

Hình 2: Kết quả test_error5.kpl

4.2 Test 2: Danh sách tham số thực sự rỗng, danh sách tham số hình thức có ít nhất 1 phần tử

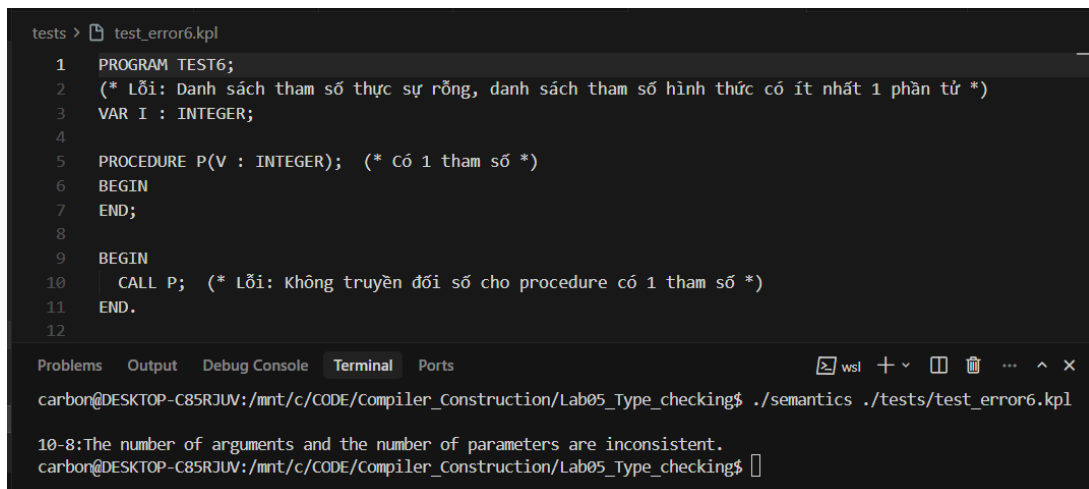
4.2.1 Code

```
1 PROGRAM TEST6;
2 (* Lỗi: Danh sách tham số thực sự rỗng,
3     danh sách tham số hình thức có ít nhất 1 phần tử *)
4 VAR I : INTEGER;
5
6 PROCEDURE P(V : INTEGER); (* Có 1 tham số *)
7 BEGIN
8 END;
9
10 BEGIN
11 CALL P; (* Lỗi: Không truyền đối số cho procedure có 1 tham số
12     *)
13 END.
```

4.2.2 Phân tích lý do gây lỗi

Procedure P được khai báo có 1 tham số V : INTEGER, do đó paramList không phải là NULL. Khi compiler gặp lời gọi CALL P không có dấu ngoặc và đối số, hàm compileArguments được gọi với paramList chứa 1 tham số. Compiler kiểm tra token tiếp theo và phát hiện không phải là SB_LPAR (không có đối số), mà là các token trong FOLLOW set như SB_SEMICOLON hoặc KW_END. Trong trường hợp này, điều kiện paramList != NULL là true nhưng không có đối số nào được cung cấp, tạo ra lỗi thiếu đối số. **Hàm phát hiện lỗi là compileArguments, báo lỗi ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY.**

4.2.3 Kết quả



```
tests > test_error6.kpl
1 PROGRAM TEST6;
2 (* Lỗi: Danh sách tham số thực sự rỗng, danh sách tham số hình thức có ít nhất 1 phần tử *)
3 VAR I : INTEGER;
4
5 PROCEDURE P(V : INTEGER); (* Có 1 tham số *)
6 BEGIN
7 END;
8
9 BEGIN
10 CALL P; (* Lỗi: Không truyền đối số cho procedure có 1 tham số *)
11 END.
12

Problems Output Debug Console Terminal Ports
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$ ./semantics ./tests/test_error6.kpl
10-8:The number of arguments and the number of parameters are inconsistent.
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$
```

Hình 3: Kết quả test_error6.kpl

4.3 Test 3: Danh sách tham số hình thức chứa nhiều phần tử hơn danh sách tham số thực sự

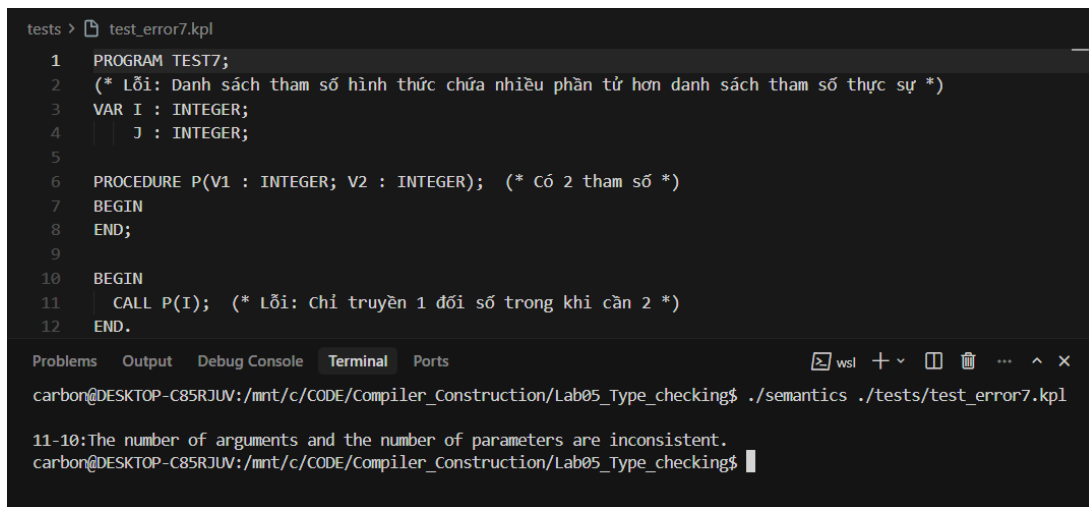
4.3.1 Code

```
1 PROGRAM TEST7;
2 (* Lỗi: Danh sách tham số hình thức chứa nhiều phần tử
3   hơn danh sách tham số thực sự *)
4 VAR I : INTEGER;
5     J : INTEGER;
6
7 PROCEDURE P(V1 : INTEGER; V2 : INTEGER); (* Có 2 tham số *)
8 BEGIN
9 END;
10
11 BEGIN
12 CALL P(I); (* Lỗi: Chỉ truyền 1 đối số trong khi cần 2 *)
13 END.
```

4.3.2 Phân tích lý do gây lỗi

Procedure P được khai báo có 2 tham số V1 : INTEGER và V2 : INTEGER. Khi compiler gặp lời gọi CALL P(I) chỉ với 1 đối số, hàm `compileArguments` bắt đầu parse đối số đầu tiên bằng cách gọi `compileArgument` với tham số V1. Sau khi parse xong đối số I, con trỏ `currentParam` được di chuyển đến tham số tiếp theo V2. Compiler kiểm tra token tiếp theo và phát hiện không phải là SB_COMMA (không còn đối số nào), mà là SB_RPAR (dấu đóng ngoặc). Điều kiện `currentParam != NULL` là true, cho thấy vẫn còn tham số V2 chưa được cung cấp đối số tương ứng. **Hàm phát hiện lỗi** là `compileArguments`, báo lỗi `ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY` vì thiếu đối số cho tham số thứ hai.

4.3.3 Kết quả



```
tests > test_error7.kpl
1 PROGRAM TEST7;
2 (* Lỗi: Danh sách tham số hình thức chứa nhiều phần tử hơn danh sách tham số thực sự *)
3 VAR I : INTEGER;
4     J : INTEGER;
5
6 PROCEDURE P(V1 : INTEGER; V2 : INTEGER); (* Có 2 tham số *)
7 BEGIN
8 END;
9
10 BEGIN
11 CALL P(I); (* Lỗi: Chỉ truyền 1 đối số trong khi cần 2 *)
12 END.
```

Problems Output Debug Console Terminal Ports

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$./semantics ./tests/test_error7.kpl

11-10: The number of arguments and the number of parameters are inconsistent.
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$

Hình 4: Kết quả test_error7.kpl

4.4 Test 4: Danh sách tham số thực sự chứa nhiều phần tử hơn danh sách tham số hình thức

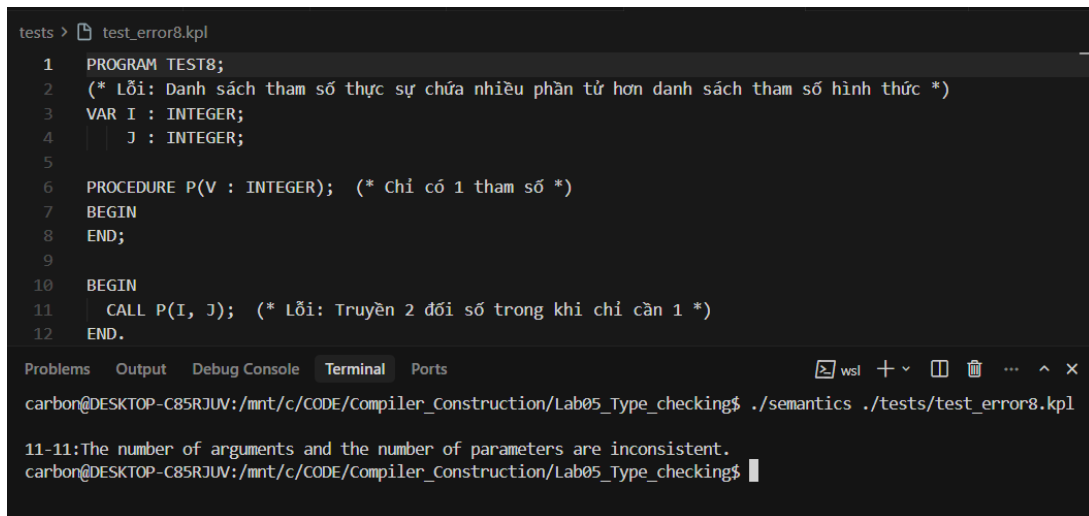
4.4.1 Code

```
1 PROGRAM TEST8;
2 (* Lỗi: Danh sách tham số thực sự chứa nhiều phần tử
3     hơn danh sách tham số hình thức *)
4 VAR I : INTEGER;
5     J : INTEGER;
6
7 PROCEDURE P(V : INTEGER); (* Chỉ có 1 tham số *)
8 BEGIN
9 END;
10
11 BEGIN
12 CALL P(I, J); (* Lỗi: Truyền 2 đối số trong khi chỉ cần 1 *)
13 END.
```

4.4.2 Phân tích lý do gây lỗi

Procedure P được khai báo chỉ có 1 tham số `V : INTEGER`. Khi compiler gặp lời gọi `CALL P(I, J)` với 2 đối số, hàm `compileArguments` parse đối số đầu tiên `I` và di chuyển con trỏ `currentParam` đến `NULL` vì đã hết tham số. Sau đó, compiler gặp dấu phẩy `SB_COMMA` và vào vòng lặp `while` để parse đối số tiếp theo. Tuy nhiên, điều kiện kiểm tra `currentParam == NULL` là `true`, cho thấy đã hết tham số nhưng vẫn còn đối số cần parse. **Hàm phát hiện lỗi** là `compileArguments`, báo lỗi `ERR_PARAMETERS_ARGUMENTS_INCONSISTENCY` vì có quá nhiều đối số so với số lượng tham số được khai báo.

4.4.3 Kết quả



```
tests > test_error8.kpl
1 PROGRAM TEST8;
2 (* Lỗi: Danh sách tham số thực sự chứa nhiều phần tử hơn danh sách tham số hình thức *)
3 VAR I : INTEGER;
4     J : INTEGER;
5
6 PROCEDURE P(V : INTEGER); (* Chỉ có 1 tham số *)
7 BEGIN
8 END;
9
10 BEGIN
11 CALL P(I, J); (* Lỗi: Truyền 2 đối số trong khi chỉ cần 1 *)
12 END.
```

Problems Output Debug Console Terminal Ports

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$./semantics ./tests/test_error8.kpl

11-11:The number of arguments and the number of parameters are inconsistent.
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$

Hình 5: Kết quả test_error8.kpl

4.5 Test 5: Không tương ứng kiểu giữa tham số hình thức và tham số thực sự

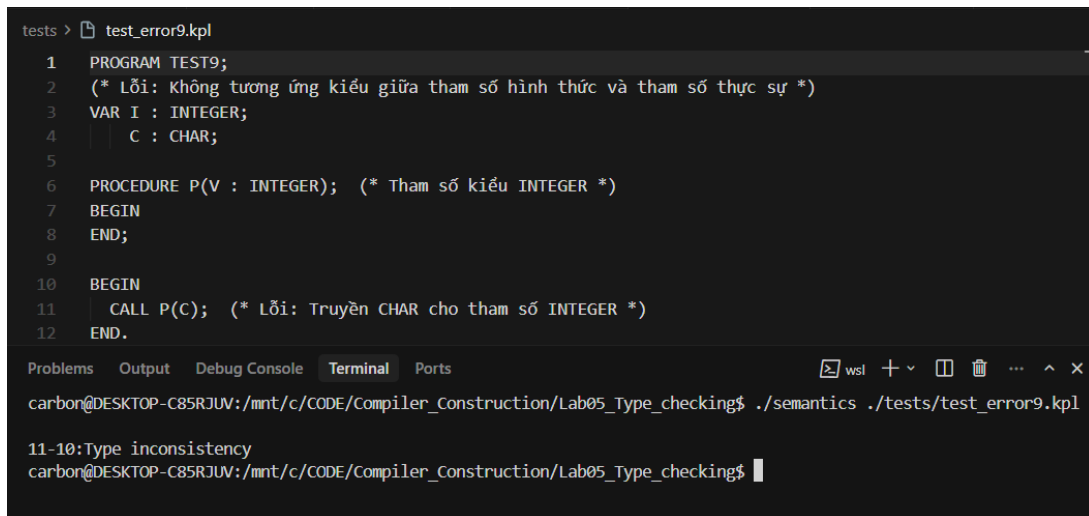
4.5.1 Code

```
1 PROGRAM TEST9;
2 (* Lỗi: Không tương ứng kiểu giữa tham số hình thức
3     và tham số thực sự *)
4 VAR I : INTEGER;
5     C : CHAR;
6
7 PROCEDURE P(V : INTEGER); (* Tham số kiểu INTEGER *)
8 BEGIN
9 END;
10
11 BEGIN
12 CALL P(C); (* Lỗi: Truyền CHAR cho tham số INTEGER *)
13 END.
```

4.5.2 Phân tích lý do gây lỗi

Procedure P được khai báo với tham số V : INTEGER, có kiểu TP_INT. Khi compiler gặp lời gọi CALL P(C) với đối số C được khai báo là CHAR, hàm compileArgument được gọi với tham số này. Vì tham số là PARAM_VALUE (không phải PARAM_REFERENCE), compiler gọi compileExpression() để parse đối số C và xác định kiểu của nó là TP_CHAR. Sau đó, compiler kiểm tra tính tương ứng kiểu bằng cách gọi compareType(argType, param->paramAttrs->type), so sánh TP_CHAR (kiểu của C) với TP_INT (kiểu của tham số V). Hàm compareType trả về 0 vì hai kiểu không khớp, dẫn đến điều kiện compareType(...) == 0 là true. **Hàm phát hiện lỗi** là compileArgument, báo lỗi ERR_TYPE_INCONSISTENCY với thông báo "Type inconsistency".

4.5.3 Kết quả



```
tests > test_error9.kpl
1 PROGRAM TEST9;
2 (* Lỗi: Không tương ứng kiểu giữa tham số hình thức và tham số thực sự *)
3 VAR I : INTEGER;
4     C : CHAR;
5
6 PROCEDURE P(V : INTEGER); (* Tham số kiểu INTEGER *)
7 BEGIN
8 END;
9
10 BEGIN
11 CALL P(C); (* Lỗi: Truyền CHAR cho tham số INTEGER *)
12 END.
```

Problems Output Debug Console Terminal Ports

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$./semantics ./tests/test_error9.kpl

11-10:Type inconsistency
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$

Hình 6: Kết quả test_error9.kpl

4.6 Test 6: Truyền tham biến bằng một hằng biểu diễn bởi định danh

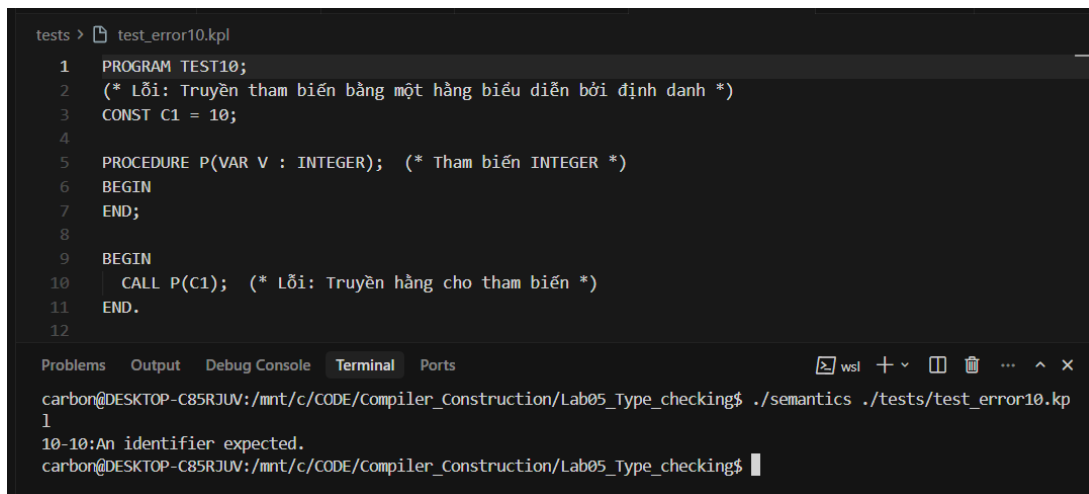
4.6.1 Code

```
1 PROGRAM TEST10;
2 (* Lỗi: Truyền tham biến bằng một hằng biểu diễn bởi định danh *)
3 CONST C1 = 10;
4
5 PROCEDURE P(VAR V : INTEGER); (* Tham biến INTEGER *)
6 BEGIN
7 END;
8
9 BEGIN
10 CALL P(C1); (* Lỗi: Truyền hằng cho tham biến *)
11 END.
```

4.6.2 Phân tích lý do gây lỗi

Procedure P được khai báo với tham biến VAR V : INTEGER, yêu cầu đối số phải là lvalue (biến, tham số, hoặc tên hàm) chứ không thể là hằng số hay biểu thức. Khi compiler gặp lời gọi CALL P(C1) với C1 là hằng số, hàm compileArgument nhận diện đây là tham biến (PARAM_REFERENCE) và kiểm tra token đầu tiên là TK_IDENT. Compiler parse token C1 và gọi hàm checkDeclaredLValueIdent để kiểm tra xem C1 có phải là lvalue hợp lệ không. Hàm checkDeclaredLValueIdent trong semantics.c tra cứu đối tượng C1 và phát hiện obj->kind == OBJ_CONSTANT. Khi kiểm tra switch case, constant không thuộc các trường hợp hợp lệ (OBJ_VARIABLE, OBJ_PARAMETER, OBJ_FUNCTION), nên rơi vào case default. **Hàm phát hiện lỗi** là checkDeclaredLValueIdent trong semantics.c, báo lỗi ERR_INVALID_IDENT với thông báo "An identifier expected." vì hằng số không thể được truyền cho tham biến.

4.6.3 Kết quả



```
tests > test_error10.kpl
1 PROGRAM TEST10;
2 (* Lỗi: Truyền tham biến bằng một hằng biểu diễn bởi định danh *)
3 CONST C1 = 10;
4
5 PROCEDURE P(VAR V : INTEGER); (* Tham biến INTEGER *)
6 BEGIN
7 END;
8
9 BEGIN
10 CALL P(C1); (* Lỗi: Truyền hằng cho tham biến *)
11 END.
12

Problems Output Debug Console Terminal Ports
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$ ./semantics ./tests/test_error10.kp
1
10-10: An identifier expected.
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking$
```

Hình 7: Kết quả test_error10.kpl

4.7 Test 7: Truyền tham biến bằng một biểu thức

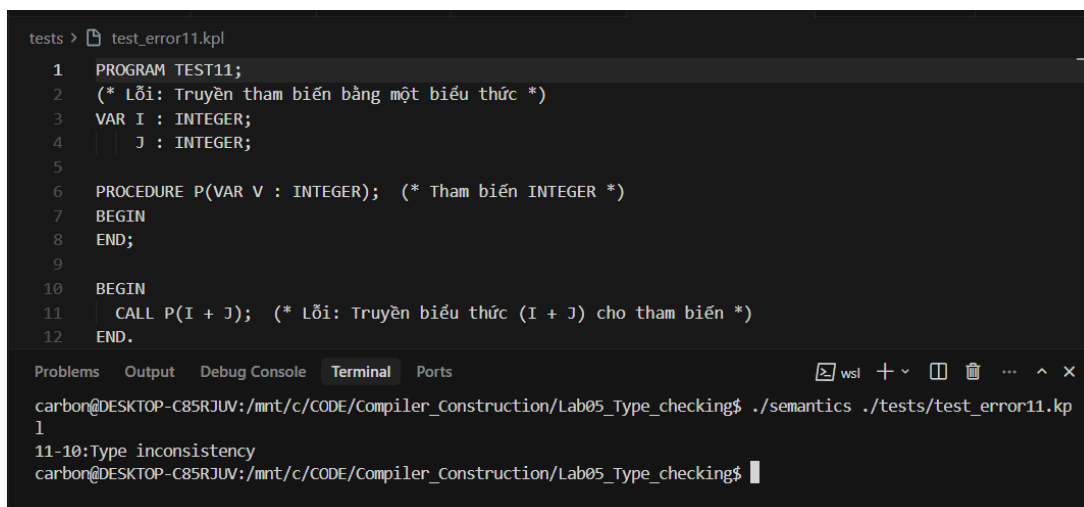
4.7.1 Code

```
1 PROGRAM TEST11;
2 (* Lỗi: Truyền tham biến bằng một biểu thức *)
3 VAR I : INTEGER;
4     J : INTEGER;
5
6 PROCEDURE P(VAR V : INTEGER); (* Tham biến INTEGER *)
7 BEGIN
8 END;
9
10 BEGIN
11     CALL P(I + J); (* Lỗi: Truyền biểu thức (I + J) cho tham biến *)
12 END.
```

4.7.2 Phân tích lý do gây lỗi

Procedure P được khai báo với tham biến VAR V : INTEGER, yêu cầu đối số phải là lvalue đơn lẻ (biến, tham số, hoặc tên hàm), không thể là biểu thức. Khi compiler gặp lời gọi CALL P(I + J) với biểu thức I + J, hàm compileArgument nhận diện đây là tham biến và kiểm tra token đầu tiên là TK_IDENT (token I). Compiler parse token I, sau đó kiểm tra token tiếp theo. Thay vì gặp SB_LPAR (cho function call) hoặc các token kết thúc như SB_RPAR hay SB_COMMA, compiler phát hiện lookAhead->tokenType == SB_PLUS. Điều này cho thấy đây không phải là lvalue đơn lẻ mà là một phần của biểu thức. Do đó, điều kiện kiểm tra toán tử là true, và compiler báo lỗi ngay lập tức. **Hàm phát hiện lỗi** là compileArgument, báo lỗi ERR_TYPE_INCONSISTENCY với thông báo "Type inconsistency" vì biểu thức không thể được truyền cho tham biến.

4.7.3 Kết quả



The screenshot shows a code editor with a KPL program and a terminal window displaying the semantic error output.

```
tests > test_error11.kpl
1 PROGRAM TEST11;
2 (* Lỗi: Truyền tham biến bằng một biểu thức *)
3 VAR I : INTEGER;
4   J : INTEGER;
5
6 PROCEDURE P(VAR V : INTEGER); (* Tham biến INTEGER *)
7 BEGIN
8 END;
9
10 BEGIN
11   CALL P(I + J); (* Lỗi: Truyền biểu thức (I + J) cho tham biến *)
12 END.
```

Problems Output Debug Console **Terminal** Ports

carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$./semantics ./tests/test_error11.kp
1
11-10:Type inconsistency
carbon@DESKTOP-C85RJUV:/mnt/c/CODE/Compiler_Construction/Lab05_Type_checking\$

Hình 8: Kết quả test_error11.kpl