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

BÀI THỰC HÀNH SỐ 5: KIỂM TRA KIỂU
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 hàm `compileConstant()` và ứng dụng trong kiểm tra kiểu

1.1 Vị trí và chức năng

File: `parser.c`
Hàm: `ConstantValue* compileConstant(void)`

Hàm `compileConstant()` có nhiệm vụ parse và kiểm tra kiểu của các hằng số trong chương trình KPL. Hàm này được gọi khi parser gặp một hằng số trong phần khai báo CONST hoặc trong các biểu thức.

1.2 Cấu trúc và luồng xử lý

Hàm xử lý các trường hợp sau:

1. **Hằng số có dấu +**: `+<constant>`
2. **Hằng số có dấu −**: `-<constant>`
3. **Hằng số ký tự**: `'<char>'`
4. **Hằng số không dấu**: `<constant>`

1.3 Code implementation

Listing 1: Hàm compileConstant() trong parser.c

```
ConstantValue* compileConstant(void) {
  ConstantValue* constValue;

  switch (lookAhead->tokenType) {
  case SB_PLUS:
    eat(SB_PLUS);
    constValue = compileConstant2();
    //TODO Check if type of the constant is integer
    if (constValue->type != TP_INT)
      error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
          currentToken->colNo);
    break;
  case SB_MINUS:
    eat(SB_MINUS);
    constValue = compileConstant2();
    //TODO Check if type of the constant is integer
    if (constValue->type != TP_INT)
      error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
          currentToken->colNo);
    constValue->intValue = - constValue->intValue;
    break;
  case TK_CHAR:
    eat(TK_CHAR);
```

```
22      constValue = makeCharConstant(currentToken->string[0]);
23      break;
24    default:
25      constValue = compileConstant2();
26      break;
27    }
28    return constValue;
29  }
```

1.4 Luồng xử lý chi tiết

Hàm `compileConstant()` được gọi khi parser gặp một hằng số trong chương trình. Luồng xử lý được mô tả chi tiết như sau:

1. **Bước 1: Kiểm tra token đầu tiên**
   - Hàm sử dụng `switch` để kiểm tra `lookAhead->tokenType`
   - Có 4 trường hợp chính: `SB_PLUS`, `SB_MINUS`, `TK_CHAR`, và default
2. **Bước 2: Xử lý hằng số có dấu +**
   - Gọi `eat(SB_PLUS)` để consume token dấu cộng
   - Gọi `compileConstant2()` để parse phần sau dấu cộng
   - `compileConstant2()` có thể trả về:
     - Integer constant (từ `TK_NUMBER`)
     - Char constant (từ `TK_CHAR`)
     - Constant identifier (từ `TK_IDENT`)
   - Kiểm tra type: `if (constValue->type != TP_INT)`
     - Nếu `TRUE`: hằng số không phải integer → báo lỗi `ERR_TYPE_INCONSISTENCY`
     - Vị trí lỗi: `currentToken->lineNo, currentToken->colNo` (vị trí của token sau dấu +)
3. **Bước 3: Xử lý hằng số có dấu -**
   - Tương tự như dấu `+`, nhưng sau khi kiểm tra type:
   - Nếu là integer: đảo dấu giá trị bằng `constValue->intValue = - constValue->intValue`
   - Nếu không phải integer: báo lỗi `ERR_TYPE_INCONSISTENCY`
4. **Bước 4: Xử lý hằng số ký tự**
   - Gọi `eat(TK_CHAR)` để consume token ký tự
   - Tạo char constant bằng `makeCharConstant(currentToken->string[0])`
   - Trả về ngay, không cần kiểm tra thêm (vì không có dấu +/-)
5. **Bước 5: Trường hợp mặc định**
   - Gọi `compileConstant2()` để xử lý hằng số không có dấu
   - Có thể là: số nguyên, ký tự, hoặc identifier của hằng số

Lưu ý quan trọng:

- Hàm `compileConstant2()` được gọi để parse phần sau dấu `+/-` hoặc hằng số không dấu
- Kiểm tra type được thực hiện **SAU** khi parse xong constant, dựa trên `constValue->type`
- Lỗi được báo tại vị trí của `currentToken` (token sau dấu +/-), không phải vị trí dấu

1.5 Ứng dụng trong kiểm tra kiểu

Hàm `compileConstant()` được sử dụng trong:

1.5.1 Khai báo hằng số (CONST)

Hàm `compileConstant()` được gọi trong `compileBlock()` khi parse khai báo CONST:

Listing 2: Ví dụ khai báo hằng số

```
CONST C1 = 10;       (* OK: integer *)
CONST C2 = +10;      (* OK: + voi integer *)
CONST C3 = -10;      (* OK: - voi integer *)
CONST C4 = 'a';      (* OK: char *)
CONST C5 = +'a';     (* ERR_TYPE_INCONSISTENCY *)
CONST C6 = -'a';     (* ERR_TYPE_INCONSISTENCY *)
```

Luồng xử lý trong compileBlock():

1. Parser gặp `CONST C5 = +'a';`
2. Gọi `compileConstant()` trong `compileBlock()`
3. Trong `compileConstant()`: gặp `SB_PLUS`
4. Gọi `compileConstant2()` → parse được `'a'` → trả về char constant
5. Kiểm tra: `constValue->type != TP_INT` → `TRUE` (vì là `TP_CHAR`)
6. Báo lỗi: `ERR_TYPE_INCONSISTENCY` tại vị trí token `'a'`

1.5.2 Quy tắc kiểm tra type

- **Quy tắc 1**: Dấu `+` và `-` chỉ được phép dùng với hằng số kiểu INTEGER
  - Lý do: Toán tử unary +/- chỉ có ý nghĩa với số, không có ý nghĩa với ký tự
  - Kiểm tra: `if (constValue->type != TP_INT)`
- **Quy tắc 2**: Nếu dùng dấu +/- với CHAR → báo lỗi `ERR_TYPE_INCONSISTENCY`
  - Lỗi được phát hiện ngay tại thời điểm parse constant
  - Vị trí lỗi: tại token sau dấu +/- (ví dụ: tại `'a'` trong `+'a'`)

- **Quy tắc 3**: Hằng số không có dấu có thể là INTEGER hoặc CHAR
  - Không cần kiểm tra type đặc biệt
  - Type được xác định tự động từ token (`TK_NUMBER` → INTEGER, `TK_CHAR` → CHAR)

1.6 Hàm hỗ trợ `compileConstant2()`

Hàm `compileConstant2()` được gọi để parse phần sau dấu `+/-` hoặc hằng số không dấu. Hàm này xử lý 3 loại constant:

Listing 3: Hàm compileConstant2() trong parser.c

```
ConstantValue* compileConstant2(void) {
  ConstantValue* constValue;
  Object* obj;

  switch (lookAhead->tokenType) {
  case TK_NUMBER:
    eat(TK_NUMBER);
    constValue = makeIntConstant(currentToken->value);
    break;
  case TK_IDENT:
    eat(TK_IDENT);
    obj = checkDeclaredConstant(currentToken->string);
    constValue = duplicateConstantValue(obj->constAttrs->value);
    break;
  case TK_CHAR:
    eat(TK_CHAR);
    constValue = makeCharConstant(currentToken->string[0]);
    break;
  default:
    error(ERR_INVALID_CONSTANT, lookAhead->lineNo, lookAhead->
       colNo);
    break;
  }
  return constValue;
}
```

Chức năng của từng trường hợp:

1. **TK_NUMBER**: Tạo integer constant từ giá trị số
   - `makeIntConstant(currentToken->value)` tạo `ConstantValue` với `type = TP_INT`
2. **TK_IDENT**: Lấy giá trị từ hằng số đã khai báo trước đó
   - `checkDeclaredConstant()` kiểm tra identifier có phải là constant không
   - `duplicateConstantValue()` sao chép giá trị constant (có thể là INT hoặc CHAR)
3. **TK_CHAR**: Tạo char constant từ ký tự

- `makeCharConstant(currentToken->string[0])` tạo `ConstantValue` với `type = TP_CHAR`

4. **Default**: Báo lỗi `ERR_INVALID_CONSTANT` nếu không phải một trong các trường hợp trên

2 Kết quả thực hiện với Example12.kpl

2.1 Mã nguồn Example12.kpl

Listing 4: Mã nguồn Example12.kpl

```
PROGRAM EXAMPLE12;
VAR A : INTEGER;
B:INTEGER;
C:INTEGER;
BEGIN
A:=10;
C:=0;
FOR B:=1 to A do
if B/2*2 !=0 then C:=C+B;
END.
```

Phân tích cấu trúc chương trình:

- **Khai báo biến**: 3 biến kiểu INTEGER (A, B, C)
- **Phép gán**: `A:=10`, `C:=0`
- **Vòng lặp FOR**: `FOR B:=1 to A do`
- **Điều kiện IF**: `if B/2*2 !=0 then`
- **Phép gán trong IF**: `C:=C+B`

2.2 Phân tích từng statement chi tiết

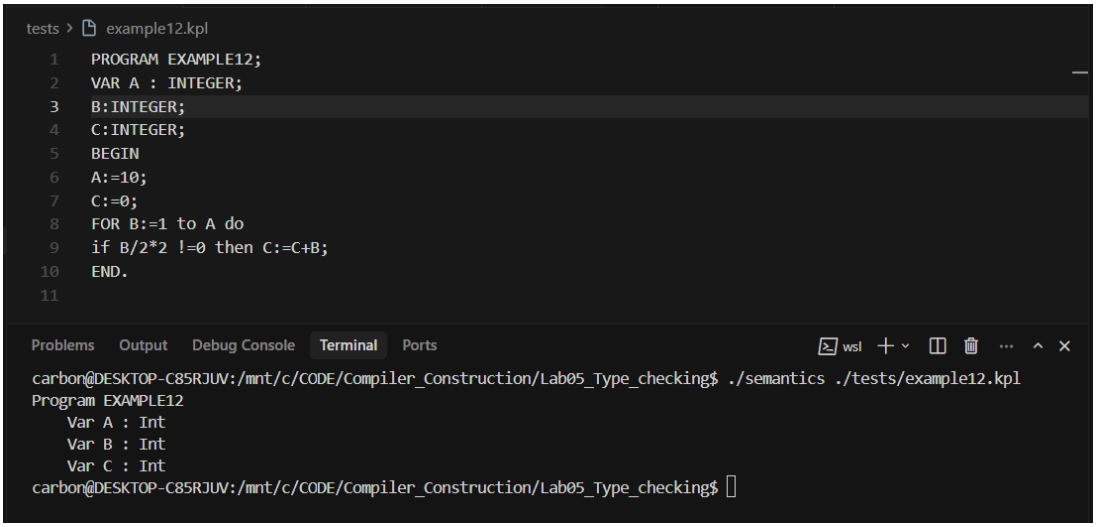
Bảng phân tích type checking cho từng statement với giải thích chi tiết về luồng kiểm tra:

Statement	Phân tích chi tiết
`VAR A : INTEGER`	• Gọi `compileType()` → trả về `intType` • Không có type checking ở đây, chỉ khai báo

`A:=10`	• `compileAssignSt()`: gọi `compileLValue()` → A là INTEGER • `compileExpression()`: `10` là `TK_NUMBER` → INTEGER • `checkTypeEquality(intType, intType)` → OK
`C:=0`	• Tương tự `A:=10` • `compileLValue()` → C: INTEGER • `compileExpression()` → `0`: INTEGER • Type khớp nhau
`FOR B:=1 to A do`	• `compileForSt()`: kiểm tra B là variable → OK • `checkIntType(B->varAttrs->type)` → OK (INTEGER) • `compileExpression()` cho `1` → INTEGER • `checkIntType(exprType1)` → OK • `compileExpression()` cho `A` → INTEGER • `checkIntType(exprType2)` → OK
`B/2*2`	• `compileTerm()`: gọi `compileFactor()` → B → INTEGER • Gọi `compileTerm2(INTEGER)`: gặp `SB_SLASH` • `checkIntType(INTEGER)` → OK (kiểm tra B) • `compileFactor()`: `2` → INTEGER • `checkIntType(INTEGER)` → OK (kiểm tra 2) • Gọi `compileTerm2(INTEGER)` đệ quy: gặp `SB_TIMES` • `checkIntType(INTEGER)` → OK (kiểm tra kết quả B/2) • `compileFactor()`: `2` → INTEGER • `checkIntType(INTEGER)` → OK (kiểm tra 2) • Kết quả cuối cùng: INTEGER

`B/2*2 !=0`	• `compileCondition()`: • `compileExpression()` → `B/2*2`: INTEGER • `checkBasicType(INTEGER)` → OK • Gặp `SB_NEQ` • `compileExpression()` → `0`: INTEGER • `checkBasicType(INTEGER)` → OK • `checkTypeEquality(INTEGER, INTEGER)` → OK
`C:=C+B`	• `compileAssignSt()`: • `compileLValue()` → C: INTEGER • `compileExpression()`: `C+B` – `compileExpression2()`: C → INTEGER – `compileExpression3()`: gặp `SB_PLUS` – `checkIntType(INTEGER)` → OK – B → INTEGER – `checkIntType(INTEGER)` → OK – Kết quả: INTEGER + INTEGER → INTEGER • `checkTypeEquality(INTEGER, INTEGER)` → OK

2.3 Kết quả chạy chương trình



Hình 1: Kết quả chạy Example12.kpl

Output:

```
Program EXAMPLE12
    Var A : Int
    Var B : Int
    Var C : Int
```

2.4 Kết luận

Chương trình `Example12.kpl` compile thành công, không có lỗi type checking. Tất cả các phép gán, biểu thức và điều kiện đều có type consistency đúng.

3 Kiểm tra khả năng điều khiển lệnh FOR bằng biến ký tự

3.1 Yêu cầu

Kiểm tra xem lệnh FOR có thể sử dụng biến kiểu CHAR hay không.

3.2 Test case

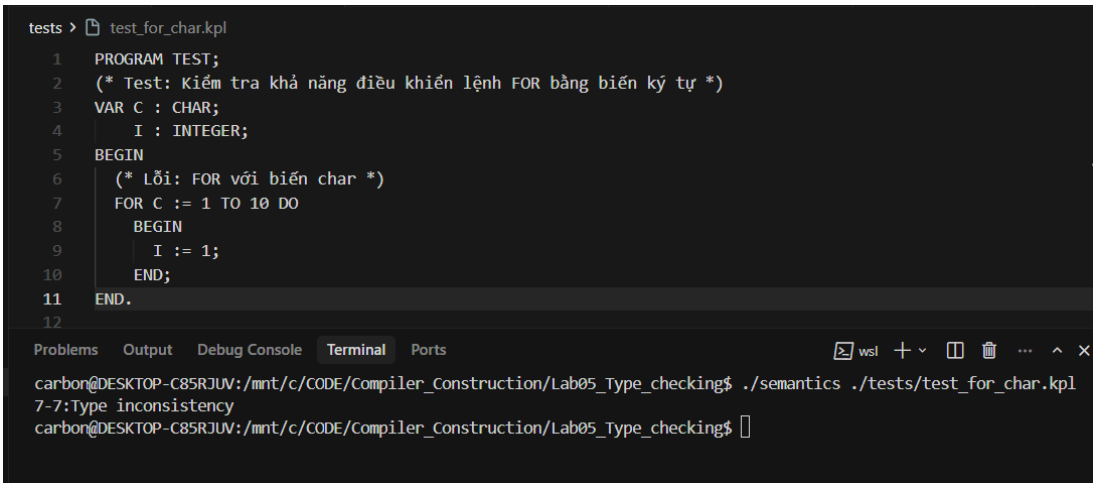
Tạo file `test_for_char.kpl` với nội dung:

Listing 5: Test case FOR với char variable

```
PROGRAM TEST;
(* Test: Kiem tra kha nang dieu khien lenh FOR bang bien ky tu *)
VAR C : CHAR;
    I : INTEGER;
```

```
BEGIN
  (* Loi: FOR voi bien char *)
  FOR C := 1 TO 10 DO
    BEGIN
      I := 1;
    END;
END.
```

3.3 Kết quả thực thi



Hình 2: Kết quả chạy test_for_char.kpl

3.4 Phân tích chi tiết

Theo đặc tả ngôn ngữ KPL, lệnh FOR chỉ chấp nhận:

- **Biến điều khiển**: phải là INTEGER
  - Lý do: Vòng lặp FOR đếm số nguyên, không thể đếm ký tự
  - Kiểm tra: `checkIntType(var->varAttrs->type)`
- **Giá trị bắt đầu**: phải là INTEGER
  - Kiểm tra: `checkIntType(exprType1)`
- **Giá trị kết thúc**: phải là INTEGER
  - Kiểm tra: `checkIntType(exprType2)`

Luồng xử lý khi gặp lỗi:

1. Parser gặp `FOR C := 1 TO 10 DO` với C là CHAR
2. Gọi `compileForSt()`
3. Gọi `checkIntType(var->varAttrs->type)`

- `var->varAttrs->type->typeClass = TP_CHAR`
- Hàm `checkIntType()` trong `semantics.c`:

```
void checkIntType(Type* type) {
  if ((type != NULL) && (type->typeClass == TP_INT))
    return;
  else error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
      currentToken->colNo);
}
```

- Điều kiện `type->typeClass == TP_INT` là `FALSE`
- Báo lỗi: `ERR_TYPE_INCONSISTENCY` tại vị trí token C

4. Compiler dừng lại và hiển thị lỗi

3.5 Code kiểm tra trong `compileForSt()`

Listing 6: Hàm compileForSt() với type checking trong parser.c

```
void compileForSt(void) {
  // TODO: Check type consistency of FOR's variable
  Object* var;
  Type* exprType1, *exprType2;

  eat(KW_FOR);
  eat(TK_IDENT);

  // check if the identifier is a variable
  var = checkDeclaredVariable(currentToken->string);
  checkIntType(var->varAttrs->type);

  eat(SB_ASSIGN);
  exprType1 = compileExpression();
  checkIntType(exprType1);

  eat(KW_TO);
  exprType2 = compileExpression();
  checkIntType(exprType2);

  eat(KW_DO);
  compileStatement();
}
```

Giải thích chi tiết từng bước:

1. **Bước 1**: Parse `FOR` và identifier của biến điều khiển
   - `eat(KW_FOR)` consume token `FOR`
   - `eat(TK_IDENT)` consume identifier (ví dụ: `C`)
2. **Bước 2**: Kiểm tra biến điều khiển

- `checkDeclaredVariable()` kiểm tra identifier có phải là variable không
  - Nếu không phải variable → báo lỗi `ERR_INVALID_VARIABLE`
  - Nếu chưa khai báo → báo lỗi `ERR_UNDECLARED_VARIABLE`
- `checkIntType(var->varAttrs->type)` kiểm tra type của biến
  - Hàm này gọi `checkIntType()` trong `semantics.c`
  - Nếu `type->typeClass != TP_INT` → báo lỗi `ERR_TYPE_INCONSISTENCY`
  - Đây là nơi phát hiện lỗi khi dùng biến CHAR trong FOR

3. **Bước 3**: Kiểm tra giá trị bắt đầu
   - `eat(SB_ASSIGN)` consume token `:=`
   - `compileExpression()` parse và kiểm tra type của expression
     - Có thể là: số, biến, biểu thức phức tạp
     - Trả về `Type*` của expression
   - `checkIntType(exprType1)` kiểm tra expression phải là INTEGER
     - Nếu là CHAR → báo lỗi `ERR_TYPE_INCONSISTENCY`
4. **Bước 4**: Kiểm tra giá trị kết thúc
   - Tương tự như giá trị bắt đầu
   - `checkIntType(exprType2)` đảm bảo giá trị kết thúc cũng là INTEGER
5. **Bước 5**: Parse statement trong vòng lặp
   - `eat(KW_DO)` consume token `DO`
   - `compileStatement()` parse statement (có thể là block, assignment, etc.)

3.6 Các trường hợp lỗi có thể gặp

1. **Lỗi 1: Biến điều khiển là CHAR**
   - Ví dụ: `FOR C := 1 TO 10 DO` (với C là CHAR)
   - Vị trí lỗi: Trong `compileForSt()`, tại lời gọi `checkIntType(var->varAttrs->type)`
   - Nguyên nhân: `checkIntType(var->varAttrs->type)` phát hiện C là CHAR
   - Lỗi: `ERR_TYPE_INCONSISTENCY` tại vị trí identifier C
2. **Lỗi 2: Giá trị bắt đầu là CHAR**
   - Ví dụ: `FOR I := 'a' TO 10 DO` (với I là INTEGER)
   - Vị trí lỗi: Trong `compileForSt()`, tại lời gọi `checkIntType(exprType1)`
   - Nguyên nhân: `compileExpression()` trả về `charType`, nhưng `checkIntType()` yêu cầu INTEGER
   - Lỗi: `ERR_TYPE_INCONSISTENCY` tại vị trí expression 'a'
3. **Lỗi 3: Giá trị kết thúc là CHAR**

- Ví dụ: `FOR I := 1 TO 'a' DO` (với I là INTEGER)
- Vị trí lỗi: Trong `compileForSt()`, tại lời gọi `checkIntType(exprType2)`
- Nguyên nhân: Tương tự lỗi 2, nhưng ở giá trị kết thúc
- Lỗi: `ERR_TYPE_INCONSISTENCY` tại vị trí expression 'a'

3.7 Kết luận

Lệnh FOR KHÔNG thể điều khiển bằng biến ký tự. Compiler sẽ phát hiện và báo lỗi `ERR_TYPE_INCONSISTENCY` ngay khi gặp biến CHAR trong lệnh FOR.

4 Thay đổi trong các ví dụ đã cho để gây các lỗi type checking

4.1 Lỗi 1: Không tương ứng kiểu khi khai báo hằng

4.1.1 Mô tả lỗi

Khi khai báo hằng số có dấu `+` hoặc `-`, hằng số đó phải là kiểu INTEGER. Nếu là CHAR sẽ gây lỗi `ERR_TYPE_INCONSISTENCY`.

4.1.2 Test case

File: `test_error1.kpl`

Listing 7: Test case lỗi khai báo hằng

```
PROGRAM TEST1;
(* Loi: Khong tuong ung kieu khi khai bao hang *)
CONST C1 = 'a';
      C2 = +'a';  (* Loi: + voi char *)
BEGIN
END.
```

4.1.3 Kết quả thực thi

Hình 3: Kết quả chạy test_error1.kpl

4.1.4 Phân tích chi tiết luồng xử lý

1. **Bước 1: Parser gặp khai báo CONST**
   - Trong `compileBlock()`, parser gặp `CONST C2 = +'a';`
   - Gọi `constValue = compileConstant();`
2. **Bước 2: Xử lý trong compileConstant()**
   - `switch (lookAhead->tokenType)`
   - Gặp `SB_PLUS` → vào case tương ứng
   - `eat(SB_PLUS)` → consume token `+`
   - `constValue = compileConstant2();`
     - Trong `compileConstant2()` gặp `TK_CHAR`
     - `eat(TK_CHAR)` và tạo char constant
     - `constValue->type = TP_CHAR`
     - Trả về `constValue` với type là CHAR
3. **Bước 3: Kiểm tra type**
   - Quay lại `compileConstant()`
   - `if (constValue->type != TP_INT)`
     - `constValue->type` = `TP_CHAR`
     - Điều kiện: `TP_CHAR != TP_INT` → `TRUE`
     - Vào block if và báo lỗi
   - `error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);`
     - `currentToken` là token `'a'` (sau khi eat trong `compileConstant2()`)
     - Vị trí lỗi: dòng 4, cột 14 (vị trí của `'a'`)
4. **Vị trí code kiểm tra:**
   - File: `parser.c`
   - Hàm: `compileConstant()`
   - Trong case `SB_PLUS`
5. **Code kiểm tra:**

```
case SB_PLUS:
  eat(SB_PLUS);
  constValue = compileConstant2();
  //TODO Check if type of the constant is integer
  if (constValue->type != TP_INT)
    error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
       currentToken->colNo);
  break;
```

4.1.5 Kết luận

Lỗi được phát hiện đúng tại vị trí hằng số char sau dấu `+`. Compiler báo lỗi `ERR_TYPE_INCONSISTENCY` vì dấu `+` chỉ được phép dùng với INTEGER.

4.2 Lỗi 2: Dùng sai số chiều của mảng so với khai báo

4.2.1 Mô tả lỗi

Khi truy cập phần tử mảng, số lượng chỉ số phải khớp với số chiều của mảng đã khai báo. Nếu dùng quá nhiều hoặc quá ít chỉ số sẽ gây lỗi.

4.2.2 Test case

File: `test_error2.kpl`

Listing 8: Test case lỗi số chiều mảng

```
PROGRAM TEST2;
(* Loi: Dung sai so chieu cua mang so voi khai bao *)
VAR A : ARRAY(. 10 .) OF INTEGER;
BEGIN
  A(.1.) := 10;
  A(.1.)(.2.) := 20;    (* Loi: A chi co 1 chieu, khong the dung 2
     chi so *)
END.
```

4.2.3 Kết quả thực thi

Hình 4: Kết quả chạy test_error2.kpl

4.2.4 Phân tích chi tiết luồng xử lý

1. **Bước 1: Parser gặp lvalue A(.1.)(.2.)**
   - Trong `compileAssignSt()`, gọi `compileLValue()`
   - `eat(TK_IDENT)` → parse `A`
   - `var = checkDeclaredLValueIdent("A")`

- Kiểm tra `var->varAttrs->type->typeClass == TP_ARRAY`
  - A là array 1 chiều → `typeClass = TP_ARRAY`
  - Gọi `compileIndexes(var->varAttrs->type)`

2. **Bước 2: Xử lý chỉ số đầu tiên trong compileIndexes()**
   - `while (lookAhead->tokenType == SB_LSEL)`
   - `eat(SB_LSEL)` → consume token `(.`
   - `indexType = compileExpression()` → parse `1`
     - `compileExpression()` trả về `intType`
   - `checkIntType(indexType)` → OK (1 là INTEGER)
   - Kiểm tra `arrayType == NULL || arrayType->typeClass != TP_ARRAY`
     - `arrayType` là array type của A → `typeClass = TP_ARRAY`, không phải NULL
     - Điều kiện: `NULL || TP_ARRAY != TP_ARRAY` → `FALSE || FALSE` → `FALSE`
     - Không vào block if, tiếp tục
   - `arrayType = arrayType->elementType`
     - `elementType` của A là `intType` (INTEGER)
     - Sau chỉ số đầu tiên, `arrayType` = `intType`
   - `eat(SB_RSEL)` → consume token `.)`

3. **Bước 3: Xử lý chỉ số thứ hai (gây lỗi)**
   - Quay lại vòng lặp while (vì vẫn còn token `SB_LSEL`)
   - `eat(SB_LSEL)` → consume token `(.` của chỉ số thứ hai
   - `indexType = compileExpression()` → parse `2`
     - `compileExpression()` trả về `intType`
   - `checkIntType(indexType)` → OK (2 là INTEGER)
   - Kiểm tra `arrayType == NULL || arrayType->typeClass != TP_ARRAY`
     - `arrayType` hiện tại = `intType` (sau chỉ số đầu tiên, đã được cập nhật thành `elementType`)
     - `intType->typeClass = TP_INT`
     - Điều kiện: `intType != NULL && TP_INT != TP_ARRAY` → `TRUE`
     - Vào block if và báo lỗi
   - `error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);`
     - `currentToken` là token sau khi parse expression `2` (sau `eat(TK_NUMBER)` trong `compileFactor()`)
     - Vị trí lỗi: tại vị trí của chỉ số thứ hai `2`
   - `eat(SB_RSEL)` và `return NULL` (thoát sớm do lỗi)

4. **Vị trí code kiểm tra:**
   - File: `parser.c`

- Hàm: `compileIndexes()`
- Trong vòng lặp while xử lý các chỉ số

5. **Code kiểm tra:**

```
Type* compileIndexes(Type* arrayType) {
  Type* indexType;
//TODO check type of indexes and elements
  while (lookAhead->tokenType == SB_LSEL) {
    eat(SB_LSEL);
    indexType = compileExpression();
    checkIntType(indexType);
    if (arrayType == NULL || arrayType->typeClass != TP_ARRAY
       ) {
      error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
         currentToken->colNo);
      eat(SB_RSEL);
      return NULL;
    }
    arrayType = arrayType->elementType;
    eat(SB_RSEL);
  }
  return arrayType;
}
```

4.2.5 Kết luận

Lỗi được phát hiện khi số lượng chỉ số không khớp với số chiều của mảng. Compiler kiểm tra sau mỗi chỉ số xem `arrayType` còn là array hay không.

4.3 Lỗi 3: Một biểu thức bắt đầu bằng + nhưng phần sau đó lại có kiểu ký tự

4.3.1 Mô tả lỗi

Toán tử unary `+` và `-` chỉ được phép dùng với biểu thức kiểu INTEGER, không được dùng với CHAR.

4.3.2 Test case

File: `test_error3.kpl`

Listing 9: Test case lỗi unary +/- với char

```
PROGRAM TEST3;
(* Loi: Mot bieu thuc bat dau bang + nhung phan sau do lai co kieu
    ky tu *)
VAR C : CHAR;
    I : INTEGER;
BEGIN
  I := +'a';      (* Loi: + voi char *)
END.
```

4.3.3 Kết quả thực thi

Hình 5: Kết quả chạy test_error3.kpl

4.3.4 Phân tích chi tiết luồng xử lý

1. **Bước 1: Parser gặp lệnh gán**
   - Trong `compileAssignSt()`, gọi `compileLValue()` → I: INTEGER
   - `eat(SB_ASSIGN)` → consume token `:=`
   - `exprType = compileExpression()`
2. **Bước 2: Xử lý trong compileExpression()**
   - `switch (lookAhead->tokenType)`
   - Gặp `SB_PLUS` → vào case tương ứng
   - `eat(SB_PLUS)` → consume token `+`
   - `type = compileExpression2()`
     - Trong `compileExpression2()` gọi `compileTerm()`
     - Trong `compileTerm()` gọi `compileFactor()`
     - Trong `compileFactor()` gặp `TK_CHAR`
     - `eat(TK_CHAR)` và trả về `charType`
     - Kết quả: `type = charType`
3. **Bước 3: Kiểm tra type**
   - Quay lại `compileExpression()`
   - `checkIntType(type)`
     - `type` = `charType`
     - Trong `checkIntType()` (semantics.c)
     - Điều kiện: `if ((type != NULL) && (type->typeClass == TP_INT))`
       * `type` = `charType` (không phải NULL)
       * `type->typeClass = TP_CHAR`

        * Điều kiện: `(charType != NULL) && (TP_CHAR == TP_INT)`
        * Kết quả: `TRUE && FALSE` → `FALSE`
        * Không vào block if, vào else và báo lỗi
    - `error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);`
        * `currentToken` là token sau khi parse expression `'a'`
        * Vị trí lỗi: dòng 6, cột 10 (vị trí của `'a'`)

4. **Vị trí code kiểm tra:**
    - File: `parser.c`
    - Hàm: `compileExpression()`
    - Trong case `SB_PLUS`

5. **Code kiểm tra:**

```
Type* compileExpression(void) {
  Type* type;
  //TODO check type of operands
  switch (lookAhead->tokenType) {
  case SB_PLUS:
    eat(SB_PLUS);
    type = compileExpression2();
    checkIntType(type);
    break;
  case SB_MINUS:
    eat(SB_MINUS);
    type = compileExpression2();
    checkIntType(type);
    break;
  default:
    type = compileExpression2();
  }
  return type;
}
```

4.3.5 Kết luận

Lỗi được phát hiện khi sử dụng unary `+` hoặc `-` với biểu thức kiểu CHAR. Compiler kiểm tra type ngay sau khi parse expression.

4.4 Lỗi 4: Không tương ứng kiểu trong lệnh gán

4.4.1 Mô tả lỗi

Trong lệnh gán, type của lvalue (biến bên trái) và expression (biểu thức bên phải) phải khớp nhau. Nếu không khớp, compiler sẽ báo lỗi `ERR_TYPE_INCONSISTENCY`.

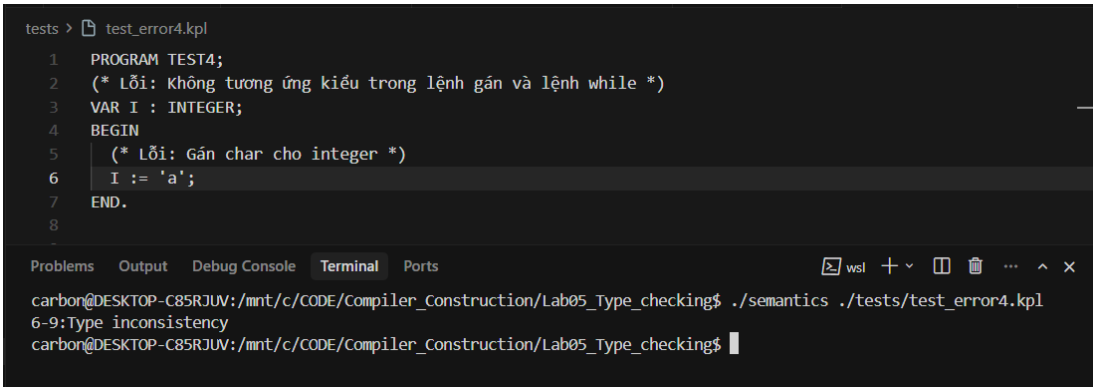
4.4.2 Test case

File: `test_error4.kpl`

Listing 10: Test case lỗi type trong gán

```
PROGRAM TEST4;
(* Loi: Khong tuong ung kieu trong lenh gan *)
VAR I : INTEGER;
BEGIN
  (* Loi: Gan char cho integer *)
  I := 'a';
END.
```

4.4.3 Kết quả thực thi



Hình 6: Kết quả chạy test_error4.kpl

4.4.4 Phân tích chi tiết luồng xử lý

a) Lệnh gán I := 'a':

1. **Bước 1: Parse lvalue**
   - Trong `compileAssignSt()`:
   - `lvalueType = compileLValue()`
     - `eat(TK_IDENT)` → parse `I`
     - `var = checkDeclaredLValueIdent("I")`
     - `var->varAttrs->type = intType`
     - `lvalueType = intType`
2. **Bước 2: Parse expression**
   - `eat(SB_ASSIGN)` → consume token `:=`
   - `exprType = compileExpression()`
     - Trong `compileExpression()` gọi `compileExpression2()`
     - Trong `compileExpression2()` gọi `compileTerm()`
     - Trong `compileTerm()` gọi `compileFactor()`

- Trong `compileFactor()` gặp `TK_CHAR`
- `eat(TK_CHAR)` và trả về `charType`
- `exprType = charType`

3. **Bước 3: Kiểm tra type equality**

- `checkTypeEquality(lvalueType, exprType)`
  - `lvalueType = intType`
  - `exprType = charType`
- Trong `checkTypeEquality()` (semantics.c)

```
void checkTypeEquality(Type* type1, Type* type2) {
  if (compareType(type1, type2) == 0)
    error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo,
       currentToken->colNo);
}
```

- `compareType(intType, charType)` được gọi
  - Trong `compareType()`: `intType->typeClass = TP_INT`, `charType->typeClass = TP_CHAR`
  - Điều kiện: `TP_INT == TP_CHAR` → `FALSE`
  - Hàm trả về **0** (không bằng nhau)
- Trong `checkTypeEquality()`: `if (compareType(intType, charType) == 0)`
  - Điều kiện: `0 == 0` → `TRUE`
  - Vào block if và báo lỗi
- `error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);`
  - `currentToken` là token sau khi parse expression `'a'`
  - Vị trí lỗi: dòng 6, cột 10 (vị trí của `'a'`)

4. **Vị trí code kiểm tra:**

- File: `parser.c`
- Hàm: `compileAssignSt()`
- Sau khi parse xong lvalue và expression

5. **Code kiểm tra:**

```
void compileAssignSt(void) {
  // TODO: parse the assignment and check type consistency
  Type* lvalueType;
  Type* exprType;

  lvalueType = compileLValue();
  eat(SB_ASSIGN);
  exprType = compileExpression();
  checkTypeEquality(lvalueType, exprType);
}
```

Lưu ý về hàm checkTypeEquality():

- Hàm này kiểm tra hai type có bằng nhau không bằng cách gọi `compareType()`
- `compareType()` trả về:
  - **1** nếu hai type bằng nhau
  - **0** nếu hai type không bằng nhau
- Logic kiểm tra: `if (compareType(type1, type2) == 0)`
  - Nếu `compareType()` trả về 0 (types không bằng nhau) → báo lỗi `ERR_TYPE_INCONSISTENCY`
  - Nếu `compareType()` trả về 1 (types bằng nhau) → không báo lỗi
- Mục đích: Đảm bảo type của lvalue và expression phải khớp nhau trong phép gán

4.4.5 Kết luận

Lỗi được phát hiện khi type của lvalue và expression trong phép gán không khớp nhau. Compiler kiểm tra type consistency ngay sau khi parse xong cả lvalue và expression, sử dụng hàm `checkTypeEquality()` để so sánh hai type. Nếu hai type không bằng nhau (theo hàm `compareType()`), compiler sẽ báo lỗi `ERR_TYPE_INCONSISTENCY` tại vị trí của expression.