Báo cáo LAB05
IT3323 Mã lớp 161269
Bài 5 : Type checking part I

Họ và tên: Dương Công Thuyết

MSSV: 20225932

I. Mô tả hàm compileConstant trong parser.c và ứng dụng của hàm đó trong kiểm tra kiểu

1. Chức năng

Hàm compileConstant() có nhiệm vụ phân tích cú pháp và kiểm tra kiểu của hằng số có dấu (signed constants) trong phần khai báo CONST của chương trình KPL.

2. Cơ chế hoạt động

Hàm xử lý bốn trường hợp chính:

Trường hợp 1: Hằng số có dấu cộng (+)

```
case SB_PLUS:
   eat(SB_PLUS);
   constValue = compileConstant2();
   if (constValue->type != TP_INT)
     error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);
   break;
```

Loại bỏ dấu +

Gọi compileConstant2() để lấy giá trị hằng số

Kiểm tra kiểu: So sánh constValue->type != TP_INT

Nếu không phải integer → báo lỗi ERR_TYPE_INCONSISTENCY

Trường hợp 2: Hằng số có dấu trừ (-)

```
case SB_MINUS:
   eat(SB_MINUS);
   constValue = compileConstant2();
   if (constValue->type != TP_INT)
     error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);
   constValue->intValue = -constValue->intValue;
```

```
    break;
```

Loại bỏ dấu -

Gọi compileConstant2() để lấy giá trị

Kiểm tra kiểu: So sánh constValue->type != TP_INT

Nếu đúng là integer → đảo dấu giá trị: constValue->intValue = -constValue->intValue

Nếu không → báo lỗi ERR_TYPE_INCONSISTENCY

Trường hợp 3: Hằng ký tự

```
case TK_CHAR:
    eat(TK_CHAR);
    constValue = makeCharConstant(currentToken->string[0]);
    break;
```

Nhận diện token ký tự 'X'

Tạo hằng số ký tự: makeCharConstant(currentToken->string[0])

Không cần kiểm tra kiểu vì ký tự không được phép có dấu

Trường hợp 4: Hằng số không dấu

```
default:
    constValue = compileConstant2();
    break;
```

Gọi compileConstant2() để xử lý số nguyên hoặc định danh hằng

Hàm con sẽ xử lý các trường hợp: số nguyên (TK_NUMBER) hoặc tên hằng (TK_IDENT)

3. Ứng dụng trong kiểm tra kiểu

Quy tắc kiểm tra

Hàm áp dụng quy tắc: Chỉ hằng số kiểu INTEGER mới được phép có dấu +/-

Vị trí sử dụng

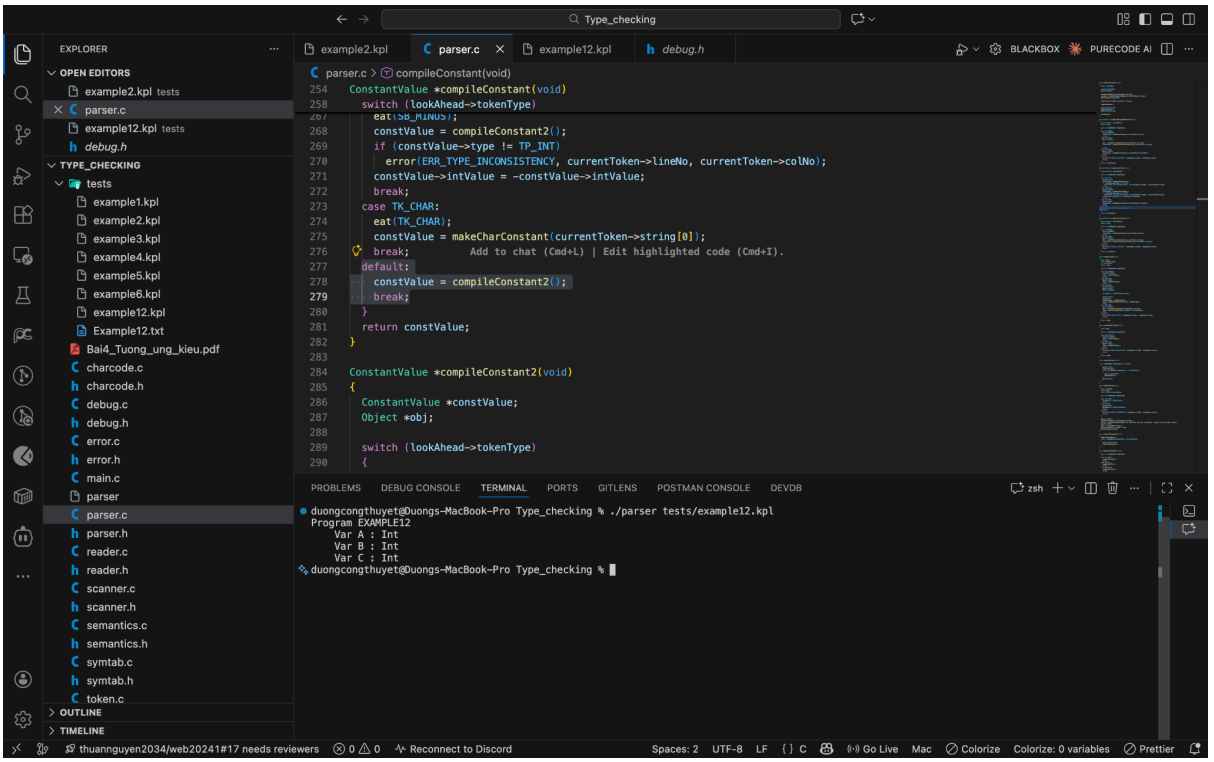
Được gọi trong hàm compileBlock() khi xử lý khai báo hằng số:

Ý nghĩa

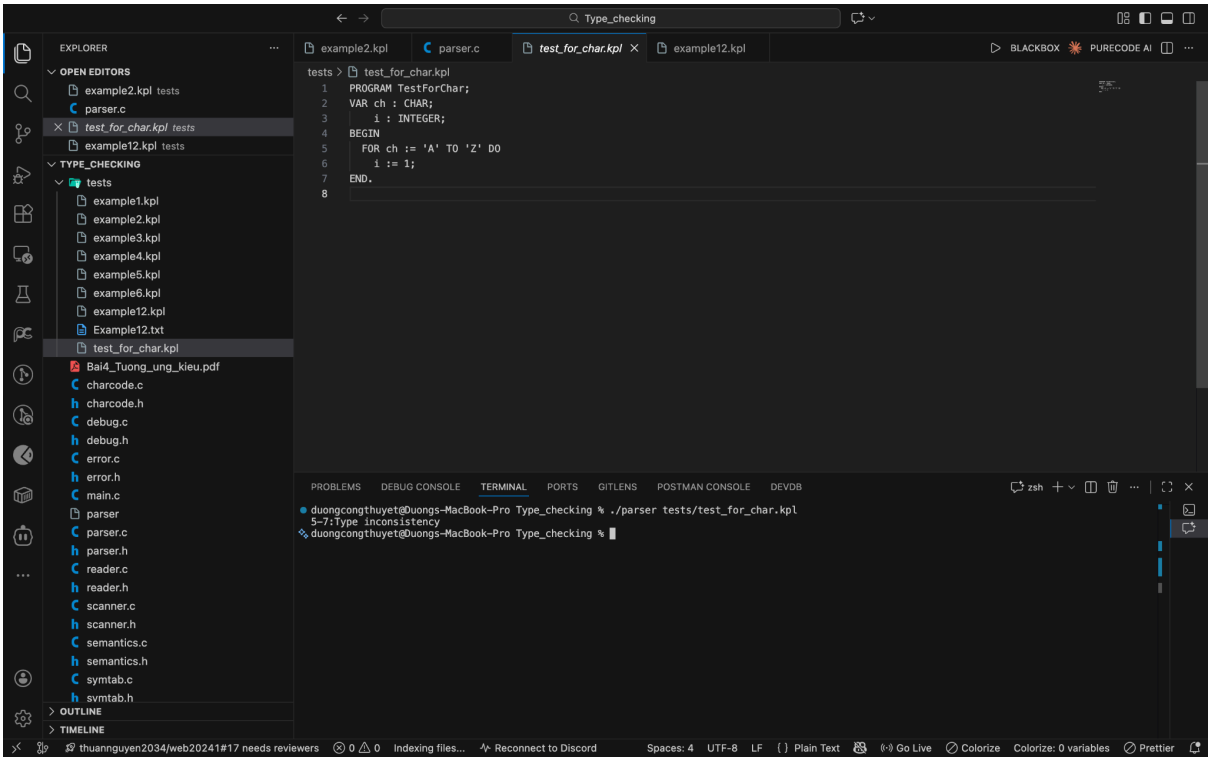
Hàm đảm bảo tính nhất quán về kiểu dữ liệu ngay từ giai đoạn khai báo hằng số, ngăn chặn các lỗi như áp dụng toán tử số học (+/-) cho kiểu ký tự - một thao tác

không hợp lệ trong ngôn ngữ KPL. Việc kiểm tra này được thực hiện thông qua điều kiện constValue->type != TP_INT và báo lỗi ngay lập tức khi phát hiện vi phạm.

II. Kết quả thực hiện với EXAMPLE12.KPL không có lỗi



III. Kiểm tra khả năng điều khiển lệnh for bằng biến ký tự



Chương trình không cho phép sử dụng biến ký tự trong vòng lặp FOR.

Nguyên nhân

Trong hàm parser.c có ba lần kiểm tra kiểu:

```
void compileForSt(void)

{

 Type *varType;

 Type *type1;

 Type *type2;



 eat(KW_FOR);

 eat(TK_IDENT);



 // check if the identifier is a variable

 varType = checkDeclaredVariable(currentToken->string)->varAttrs->type;

 checkIntType(varType);         // ← Kiểm tra 1: Biến phải là INTEGER



 eat(SB_ASSIGN);

 type1 = compileExpression();

 checkIntType(type1);           // ← Kiểm tra 2: Giá trị bắt đầu phải là INTEGER



 eat(KW_TO);

 type2 = compileExpression();

 checkIntType(type2);           // ← Kiểm tra 3: Giá trị kết thúc phải là INTEGER



 eat(KW_DO);

 compileStatement();

}
```

Kết luận

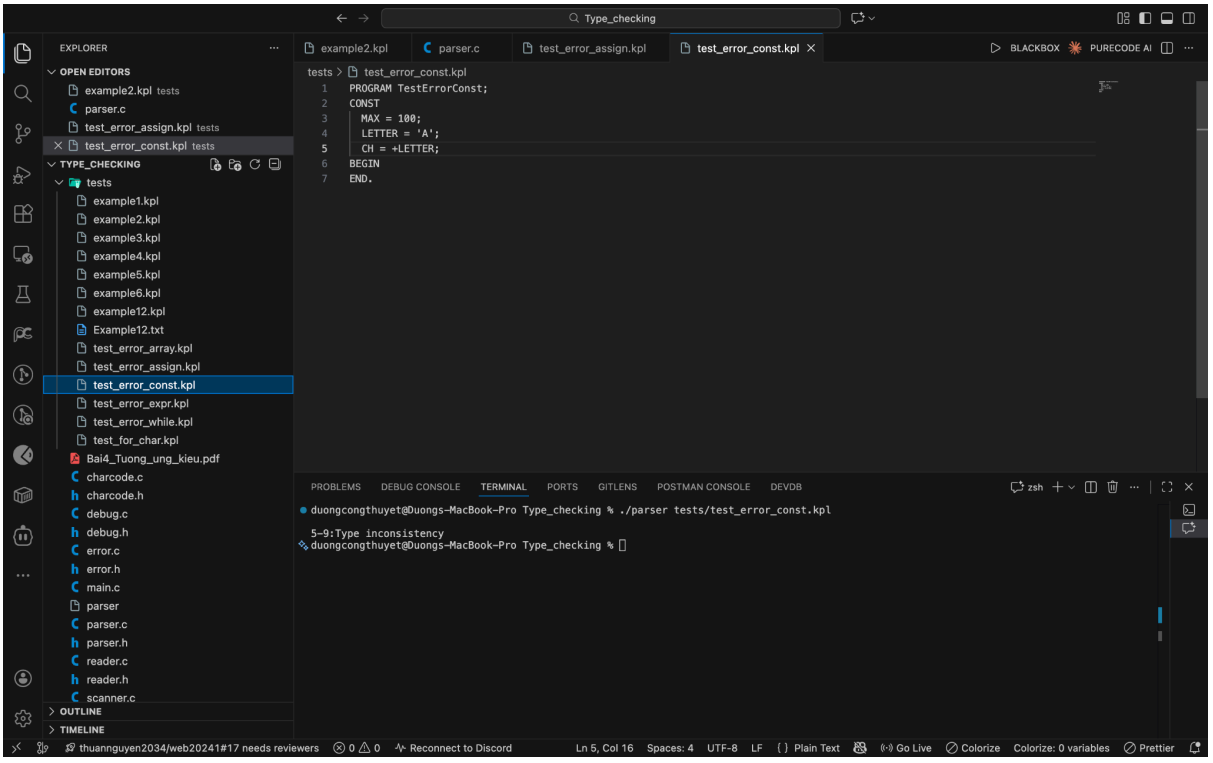
Vòng lặp FOR trong ngôn ngữ KPL bắt buộc:

- Biến điều khiển phải là kiểu INTEGER
- Giá trị bắt đầu phải là kiểu INTEGER
- Giá trị kết thúc phải là kiểu INTEGER

=> Không thể sử dụng biến ký tự CHAR để điều khiển vòng lặp FOR.

IV. Thay đổi trong các ví dụ đã cho để gây các lỗi:

1. Không tương ứng kiểu khi khai báo hằng



Code test:

```
CH = +LETTER;    (* LETTER là hằng ký tự *)
```

Kết quả: 5-9:Type inconsistency

TẠI SAO BỊ LỖI:

Dấu + chỉ được phép đứng trước hằng số INTEGER, không được dùng với CHAR.

LETTER có kiểu CHAR nên không thể có dấu +.

HÀM XỬ LÝ:

• compileConstant() (parser.c, dòng 267-268)

- Gặp dấu +, gọi compileConstant2() lấy giá trị

- Kiểm tra: if (constValue->type != TP_INT)

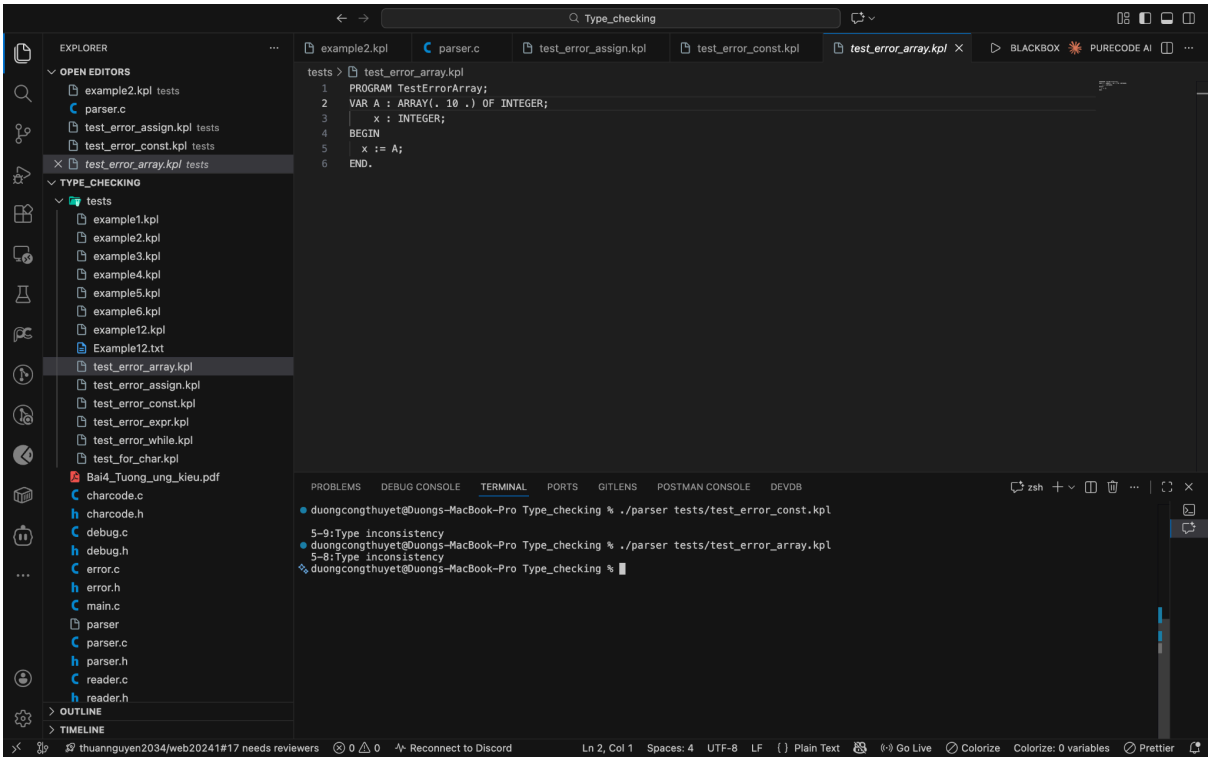
- Kết quả: constValue->type = TP_CHAR ≠ TP_INT → Báo lỗi

• error() (error.c)

- In ra: "Type inconsistency"

- Dừng chương trình

2. Dùng sai số chiều của mảng so với khai báo



Code test:

```
VAR A : ARRAY(. 10 .) OF INTEGER;
    x : INTEGER;
x := A;    (* Gán toàn bộ mảng cho biến integer *)
```

Kết quả: 5-8:Type inconsistency

TẠI SAO BỊ LỖI:

Không thể gán mảng cho biến integer. Phải dùng chỉ số: x := A(. 1 .)

HÀM XỬ LÝ:

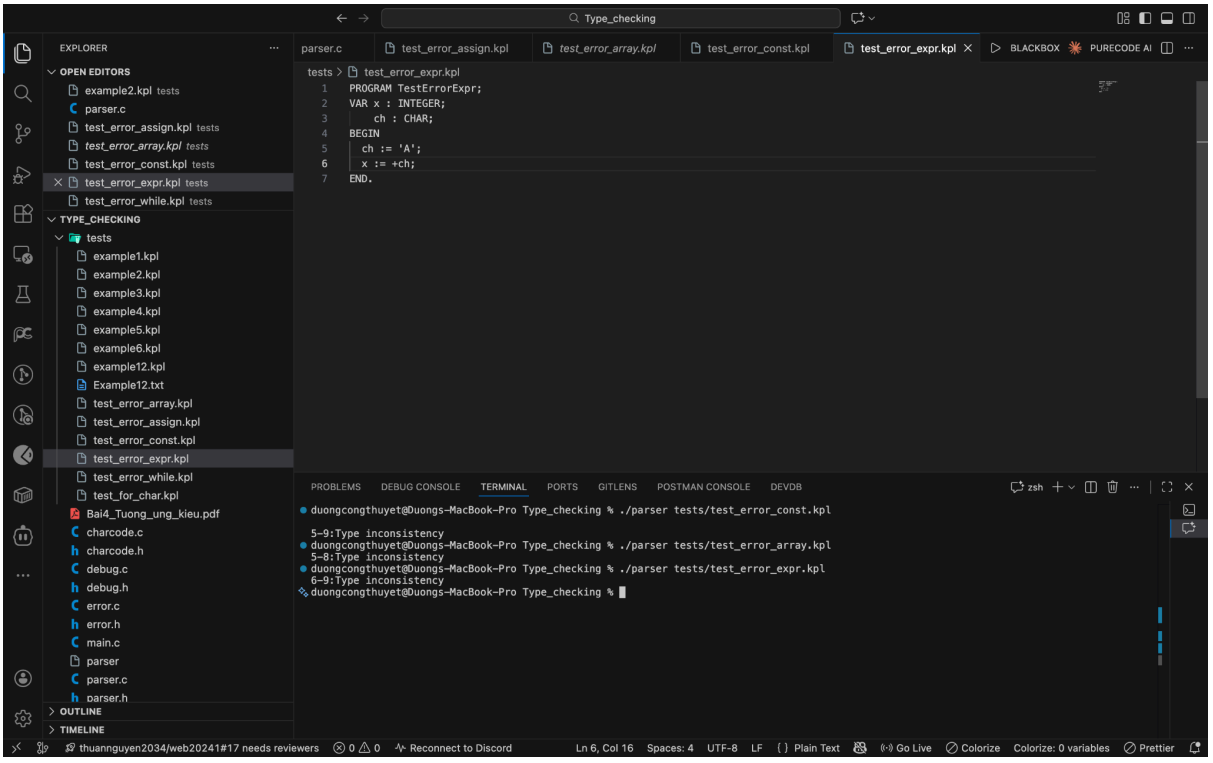
• compileAssignSt() (parser.c, dòng 489-498)

- Lấy kiểu vế trái: x → TP_INT
- Lấy kiểu vế phải: A → TP_ARRAY
- Gọi checkTypeEquality(TP_INT, TP_ARRAY)

• checkTypeEquality() (semantics.c, dòng 149-152)

- Gọi compareType(type1, type2)
- compareType() so sánh TP_INT vs TP_ARRAY → trả về 0 (không bằng)
- Báo lỗi ERR_TYPE_INCONSISTENCY

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



Code test:

```
VAR ch : CHAR;
x := +ch;    (* Dấu + với biến ký tự *)
```

Kết quả: 6-9:Type inconsistency

TẠI SAO BỊ LỖI:

Toán tử đơn ngôi + chỉ áp dụng cho kiểu INTEGER, không được dùng với CHAR.

HÀM XỬ LÝ:

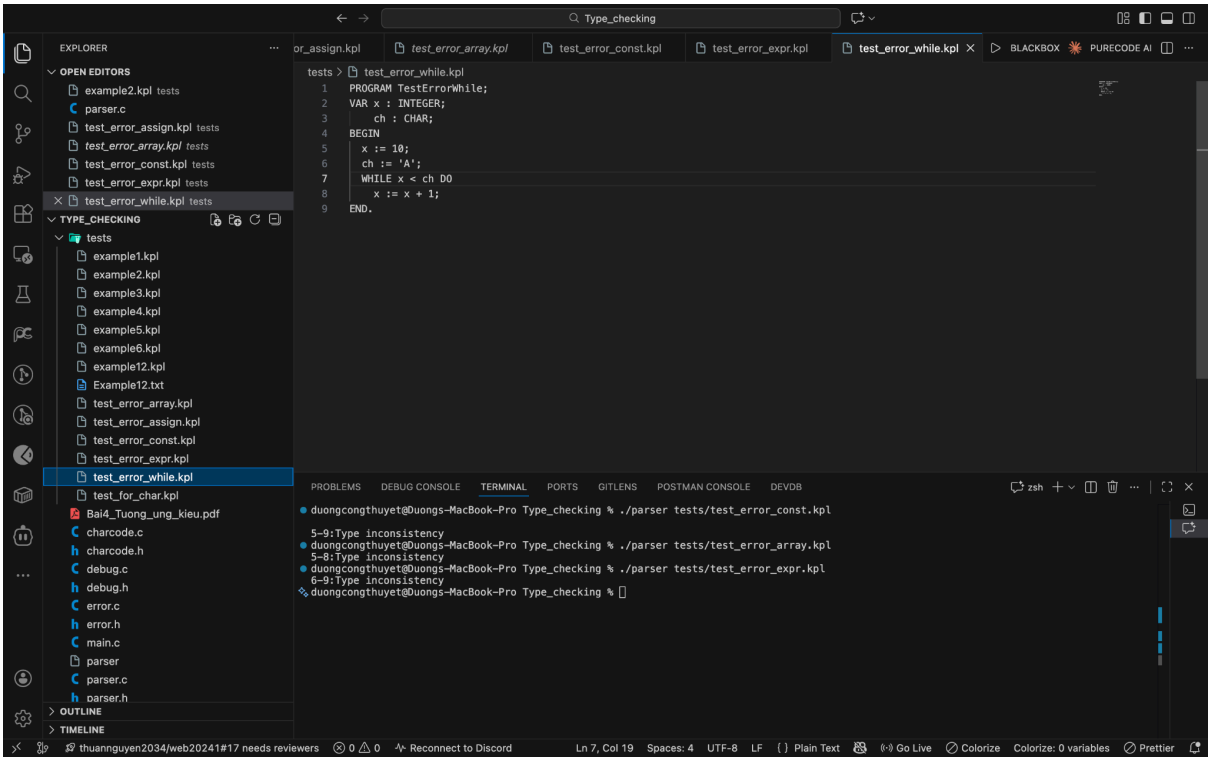
• compileExpression() (parser.c, dòng 677-693)

- Gặp dấu +, gọi compileExpression2()
- Nhận được kiểu của ch → TP_CHAR
- Gọi checkIntType(TP_CHAR)

• checkIntType() (semantics.c, dòng 116-120)

- Kiểm tra: if (type->typeClass == TP_INT)
- Kết quả: TP_CHAR ≠ TP_INT → Báo lỗi ERR_TYPE_INCONSISTENCY

4. Không tương ứng kiểu trong lệnh gán, lệnh while



Code test:

```
VAR x : INTEGER;
    ch : CHAR;
WHILE x < ch DO    (* So sánh INTEGER với CHAR *)
```

Kết quả: 7-13:Type inconsistency

TẠI SAO BỊ LỖI:

Trong điều kiện so sánh, hai vế phải cùng kiểu. Không được so sánh INTEGER với CHAR.

HÀM XỬ LÝ:

• compileCondition() (parser.c, dòng 649-675)

- Lấy kiểu vế trái: x → TP_INT
- Kiểm tra là kiểu cơ bản → OK
- Lấy kiểu vế phải: ch → TP_CHAR
- Gọi checkTypeEquality(TP_INT, TP_CHAR)

• checkTypeEquality() (semantics.c, dòng 149-152)

- compareType(TP_INT, TP_CHAR) → trả về 0
- Báo lỗi ERR_TYPE_INCONSISTENCY