

Báo cáo Thực hành Xây dựng chương trình dịch

Bài 2: Phân tích cú pháp

Họ và tên: Lê Hải Yến

MSSV: 20225780

I. Giải thích logic code

1. Kiến trúc tổng quan của Parser

Parser (bộ phân tích cú pháp) là thành phần thứ hai trong quy trình biên dịch, làm việc ngay sau Scanner. Trong khi Scanner chỉ nhận diện các token riêng lẻ, Parser có nhiệm vụ cao hơn: kiểm tra xem chuỗi các token đó có tuân theo ngữ pháp của ngôn ngữ KPL hay không.

Parser hoạt động theo phương pháp phân tích cú pháp đệ quy xuống (Recursive Descent Parsing) với lookahead 1 token. Điều này có nghĩa là tại mỗi thời điểm, Parser nhìn trước 1 token để quyết định nên làm gì tiếp theo.

2. Cơ chế hoạt động

2.1. Token Management

Parser duy trì hai biến toàn cục quan trọng:

- `currentToken`: Đây là token đang được xử lý hiện tại. Nó giống như vị trí con trỏ cho biết Parser đang đứng ở đâu trong dòng code.
- `lookAhead`: Đây là token tiếp theo trong luồng. Nó cho phép Parser nhìn trước để đưa ra quyết định. Ví dụ, khi gặp `identifier`, Parser cần nhìn token tiếp theo để biết đây là lời gọi hàm (nếu theo sau là dấu ngoặc), truy cập mảng (nếu theo sau là dấu ngoặc vuông), hay chỉ là biến thường.

Hai hàm điều khiển token:

- `scan()` - Hàm “trượt cửa sổ”:
 - + Lưu `currentToken` vào biến tạm `tmp`
 - + Chuyển `lookAhead` thành `currentToken` mới
 - + Gọi Scanner để lấy token mới cho `lookAhead`
 - + Giải phóng bộ nhớ của token cũ (`tmp`)
- `eat()` - Hàm “tiêu thụ token”:
 - + Kiểm tra xem `lookAhead` có đúng loại token mong đợi không
 - + Nếu đúng: In token ra console (để debug), sau đó gọi `scan()` để tiến lên
 - + Nếu sai: Gọi `missingToken()` để báo lỗi cú pháp

2.2. Recursive Descent

Parser được tổ chức thành một hệ thống các hàm, mỗi hàm tương ứng với một quy tắc ngữ pháp. Các hàm này gọi nhau theo cấu trúc phân cấp, tạo thành một cây gọi hàm phản ánh cấu trúc ngữ pháp của chương trình. Mỗi lần gọi hàm con, Parser "đi sâu" vào một cấu trúc ngữ pháp cụ thể hơn. Khi hàm con return, Parser "quay lên" và tiếp tục xử lý phần còn lại.

3. Cấu trúc chương trình KPL

Cấu trúc theo thứ tự sau:

- Header: PROGRAM tên_chương_trình;
- Phần khai báo CONST (tùy chọn): Định nghĩa các hằng số
- Phần khai báo TYPE (tùy chọn): Định nghĩa các kiểu dữ liệu mới
- Phần khai báo VAR (tùy chọn): Khai báo biến
- Phần khai báo FUNCTION/PROCEDURE (tùy chọn): Định nghĩa hàm và thủ tục
- Thân chương trình: BEGIN ... END. (bắt buộc)

a. compileProgram() - Hàm khởi đầu

Đây là điểm vào chính của Parser. Nó kiểm tra cấu trúc tổng thể:

- In thông báo bắt đầu parse
- Bắt buộc có từ khóa PROGRAM
- Bắt buộc có tên chương trình (identifier)
- Bắt buộc có dấu chấm phẩy
- Gọi compileBlock() để xử lý phần thân
- Bắt buộc kết thúc bằng dấu chấm
- In thông báo hoàn thành

Nếu thiếu bất kỳ thành phần nào, Parser sẽ gọi missingToken() và dừng chương trình.

b. Hệ thống Block - Xử lý phân tầng

Block được chia thành 5 giai đoạn xử lý tuần tự, mỗi giai đoạn có một hàm riêng:

- compileBlock() - Giai đoạn 1: Xử lý CONST
 - + Kiểm tra có từ khóa CONST không
 - + Nếu có: Ăn CONST, gọi compileConstDecl() cho khai báo đầu tiên, gọi compileConstDecls() cho các khai báo tiếp theo
 - + Chuyển sang compileBlock2() dù có hay không CONST

Thiết kế này cho phép phần CONST là tùy chọn - nếu không có, Parser nhảy thẳng sang giai đoạn tiếp theo.

- compileBlock2() - Giai đoạn 2: Xử lý TYPE
 - + Kiểm tra từ khóa TYPE

- + Nếu có: Xử lý tương tự CONST
- + Chuyển sang compileBlock3()
- compileBlock3() - Giai đoạn 3: Xử lý VAR
 - + Kiểm tra từ khóa VAR
 - + Nếu có: Xử lý khai báo biến
 - + Chuyển sang compileBlock4()
- compileBlock4() - Giai đoạn 4: Xử lý FUNCTION và PROCEDURE
 - + In thông báo bắt đầu parse subroutines
 - + Vòng lặp while kiểm tra liên tục: while (lookAhead->tokenType == KW_FUNCTION || lookAhead->tokenType == KW_PROCEDURE)
 - + Mỗi lần gặp FUNCTION: gọi compileFuncDecl()
 - + Mỗi lần gặp PROCEDURE: gọi compileProcDecl()
 - + Lặp cho đến khi không còn function/procedure nào
 - + Chuyển sang compileBlock5()
 - + In thông báo hoàn thành

Vòng lặp while cho phép có nhiều function/procedure liên tiếp.

- compileBlock5() - Giai đoạn 5: Thân chương trình
 - + Bắt buộc có BEGIN
 - + Gọi compileStatements() xử lý các câu lệnh
 - + Bắt buộc có END

Đây là phần bắt buộc duy nhất trong Block - mọi chương trình KPL phải có BEGIN...END.

4. Xử lý khai báo

a. Khai báo hằng số (CONST)

- compileConstDecls(): Vòng lặp while (lookAhead->tokenType == TK_IDENT) cho phép xử lý nhiều khai báo hằng liên tiếp. Mỗi khai báo bắt đầu bằng identifier (tên hằng).
- compileConstDecl(): Xử lý một khai báo theo cú pháp tên = giá_trị;
 - + Ăn TK_IDENT (tên hằng)
 - + Ăn SB_EQ (dấu bằng)
 - + Gọi compileConstant() để xử lý giá trị
 - + Ăn SB_SEMICOLON
- compileConstant(): Xử lý giá trị hằng có thể có dấu
 - + Nếu gặp SB_PLUS hoặc SB_MINUS: Ăn dấu, gọi compileConstant2()
 - + Nếu gặp TK_CHAR: Ăn trực tiếp (ký tự không có dấu +/-)

- + Ngược lại: Gọi compileConstant2()
- compileConstant2(): Xử lý giá trị cụ thể
 - + Chấp nhận TK_NUMBER (số) hoặc TK_IDENT (tên hằng đã khai báo trước)
 - + Nếu không phải: Báo lỗi ERR_INVALID_CONSTANT
- compileUnsignedConstant(): Chỉ chấp nhận hằng không dấu
 - + TK_NUMBER, TK_IDENT, hoặc TK_CHAR

b. Khai báo kiểu (TYPE)

- compileTypeDecls() và compileTypeDecl(): Hoạt động tương tự như CONST

Cú pháp: tên_kiểu = định_nghĩa_kiểu;

- compileType(): Xử lý các loại kiểu dữ liệu
 - + KW_INTEGER: Kiểu số nguyên - chỉ cần ăn từ khóa
 - + KW_CHAR: Kiểu ký tự - ăn từ khóa
 - + KW_ARRAY: Kiểu mảng - phức tạp hơn
 - Cú pháp: ARRAY[số] OF kiểu_phần_tử
 - Ăn KW_ARRAY, SB_LSEL, TK_NUMBER, SB_RSEL, KW_OF
 - Gọi compileType() đệ quy để xử lý kiểu phần tử
 - Đệ quy cho phép mảng nhiều chiều: ARRAY[5] OF ARRAY[3] OF INTEGER

+ TK_IDENT: Kiểu tự định nghĩa (đã khai báo trước trong TYPE)

- compileBasicType(): Chỉ chấp nhận INTEGER hoặc CHAR
- Dùng cho kiểu trả về của function và kiểu tham số

c. Khai báo biến (VAR)

- compileVarDecls() và compileVarDecl(): Tương tự CONST/TYPE
 - + Cú pháp: tên_biến : kiểu_dữ_liệu;
 - + Gọi compileType() để xử lý kiểu (có thể là mảng, kiểu tự định nghĩa, v.v.)

d. Khai báo Function và Procedure

- compileFuncDecl() - Xử lý FUNCTION:
 - + Cú pháp: FUNCTION tên(params) : kiểu_trả_về; block;
 - + Đặc điểm: Có giá trị trả về (INTEGER hoặc CHAR)
 - + Trình tự:

- Ăn KW_FUNCTION và tên function
 - Gọi compileParams() - danh sách tham số có thể rỗng
 - Ăn SB_COLON
 - Gọi compileBasicType() - chỉ INTEGER hoặc CHAR
 - Ăn SB_SEMICOLON
 - Gọi compileBlock() đệ quy - function có thể chứa các khai báo con
 - Ăn SB_SEMICOLON kết thúc
- compileProcDecl() - Xử lý PROCEDURE:
 - + Cú pháp: PROCEDURE tên(params); block;
 - + Khác function: Không có kiểu trả về
 - + Trình tự tương tự nhưng bỏ qua phần : kiểu_trả_về

Cả hai đều gọi compileBlock() đệ quy, có nghĩa là function/procedure có thể chứa các khai báo CONST, TYPE, VAR, và cả function/procedure con (nested).

5. Xử lý tham số (Parameters)

- compileParams(): Xử lý danh sách tham số
 - + Kiểm tra có dấu SB_LPAR (mở ngoặc) không
 - + Nếu có: Ăn ngoặc, gọi compileParam() cho tham số đầu, gọi compileParams2() cho các tham số tiếp theo, ăn ngoặc đóng
 - + Nếu không: Không làm gì (danh sách rỗng - function/procedure không có tham số)
- compileParams2(): Xử lý các tham số tiếp theo
 - + Vòng lặp while (lookAhead->tokenType == SB_SEMICOLON) - tham số cách nhau bởi dấu chấm phẩy
 - + Mỗi lần: Ăn dấu chấm phẩy, gọi compileParam()
- compileParam(): Xử lý một tham số
 - + Hai loại tham số:
 - Tham trị (pass by value): tên : kiểu
Truyền giá trị, thay đổi trong hàm không ảnh hưởng ngoài
Bắt đầu bằng TK_IDENT
 - Tham chiếu (pass by reference): VAR tên : kiểu
Truyền địa chỉ, thay đổi trong hàm ảnh hưởng ra ngoài
Bắt đầu bằng KW_VAR
 - + Switch-case kiểm tra token đầu:
 - TK_IDENT: Ăn IDENT, ăn COLON, gọi compileBasicType()

- KW_VAR: Ăn VAR, ăn IDENT, ăn COLON, gọi compileBasicType()
- Khác: Báo lỗi ERR_INVALIDPARAM

6. Xử lý câu lệnh (Statements)

a. Cấu trúc xử lý câu lệnh

- compileStatements(): Xử lý chuỗi câu lệnh
 - + Gọi compileStatement() cho câu lệnh đầu
 - + Gọi compileStatements2() cho các câu lệnh tiếp theo
- compileStatements2(): Xử lý câu lệnh tiếp theo
 - + Kiểm tra có SB_SEMICOLON (dấu phân cách câu lệnh) không
 - + Nếu có: Ăn dấu chấm phẩy, gọi compileStatement(), gọi compileStatements2() đệ quy
 - + Nếu không: Dừng (hết câu lệnh)
- compileStatement(): Switch-case phân loại câu lệnh
 - + TK_IDENT: Câu lệnh gán → gọi compileAssignSt()
 - + KW_CALL: Gọi procedure → gọi compileCallSt()
 - + KW_BEGIN: Nhóm lệnh → gọi compileGroupSt()
 - + KW_IF: Câu lệnh điều kiện → gọi compileIfSt()
 - + KW_WHILE: Vòng lặp while → gọi compileWhileSt()
 - + KW_FOR: Vòng lặp for → gọi compileForSt()
 - + SB_SEMICOLON, KW_END, KW_ELSE: Câu lệnh rỗng (empty statement) - không làm gì
 - + Khác: Báo lỗi ERR_INVALIDSTATEMENT

b. Các loại câu lệnh cụ thể

- compileAssignSt() - Câu lệnh gán:
 - + Cú pháp: biến := biểu_thức hoặc mảng[index] := biểu_thức
 - + Ăn TK_IDENT (tên biến)
 - + Gọi compileIndexes() - xử lý chỉ số mảng nếu có
 - + Ăn SB_ASSIGN (dấu :=)
 - + Gọi compileExpression() - vế phải có thể là biểu thức phức tạp
- compileCallSt() - Gọi procedure:
 - + Cú pháp: CALL tên_procedure(đối_số)
 - + Ăn KW_CALL và tên procedure
 - + Gọi compileArguments() - danh sách đối số có thể rỗng
- compileGroupSt() - Nhóm lệnh:
 - + Cú pháp: BEGIN câu_lệnh_1; câu_lệnh_2; ... END
 - + Ăn BEGIN

- + Gọi compileStatements() - xử lý các câu lệnh bên trong
- + Ăn END
- compileIfSt() - Câu lệnh điều kiện:
 - + Cú pháp: IF điều_kiện THEN câu_lệnh [ELSE câu_lệnh]
 - + Ăn IF
 - + Gọi compileCondition() - xử lý điều kiện (biểu thức so sánh)
 - + Ăn THEN
 - + Gọi compileStatement() - câu lệnh khi điều kiện đúng
 - + Kiểm tra có ELSE không:
 - Nếu có: Gọi compileElseSt()
 - Nếu không: Bỏ qua (ELSE là tùy chọn)
- compileWhileSt() - Vòng lặp while:
 - + Cú pháp: WHILE điều_kiện DO câu_lệnh
 - + Ăn WHILE
 - + Gọi compileCondition()
 - + Ăn DO
 - + Gọi compileStatement()
- compileForSt() - Vòng lặp for:
 - + Cú pháp: FOR biến := giá_trị_đầu TO giá_trị_cuối DO câu_lệnh
 - + Ăn FOR, biến đếm, dấu :=
 - + Gọi compileExpression() cho giá trị bắt đầu
 - + Ăn TO
 - + Gọi compileExpression() cho giá trị kết thúc
 - + Ăn DO
 - + Gọi compileStatement()

7. Xử lý đối số (Arguments)

- a. compileArguments(): Xử lý danh sách đối số khi gọi hàm/procedure
 - Cú pháp: (đối_số_1, đối_số_2, ...) hoặc ()
 - Kiểm tra có SB_LPAR không
 - Nếu có:
 - + Ăn SB_LPAR
 - + Kiểm tra ngay SB_RPAR (danh sách rỗng): Nếu có thì ăn và return
 - + Ngược lại: Gọi compileExpression() cho đối số đầu, gọi compileArguments2() cho các đối số tiếp, ăn SB_RPAR
- b. compileArguments2(): Xử lý đối số tiếp theo
 - Vòng lặp while (lookAhead->tokenType == SB_COMMA)
 - Mỗi lần: Ăn dấu phẩy, gọi compileExpression()

- Mỗi đối số là một biểu thức (expression), không chỉ là giá trị đơn giản.

8. Xử lý điều kiện (Condition)

- compileCondition(): Xử lý biểu thức so sánh
 - + Gọi compileExpression() - về trái
 - + Gọi compileCondition2() - toán tử so sánh và về phải
- compileCondition2(): Switch-case xử lý 6 toán tử so sánh
 - + SB_EQ (=): Bằng
 - + SB_NEQ (!=): Khác
 - + SB_LE (<=): Nhỏ hơn hoặc bằng
 - + SB_LT (<): Nhỏ hơn
 - + SB_GE (>=): Lớn hơn hoặc bằng
 - + SB_GT (>): Lớn hơn
- Mỗi trường hợp: Ăn toán tử, gọi compileExpression() cho về phải.
- Nếu không phải 6 toán tử: Báo lỗi ERR_INVALIDCOMPparator.

9. Xử lý biểu thức (Expression)

a. Cấu trúc phân cấp biểu thức

- Biểu thức trong KPL có cấu trúc phân cấp 3 tầng theo thứ tự ưu tiên toán tử:
 - + Expression = [+|-] Term ([+|-] Term)*
 - + Term = Factor ([*|/] Factor)*
 - + Factor = số | ký_tự | biến | mảng[index] | hàm(args) | (expression)
- Cấu trúc này phản ánh thứ tự ưu tiên toán tử:
 - + Factor: Mức cao nhất - các giá trị cơ bản
 - + Term: Mức trung - phép nhân/chia (ưu tiên cao hơn cộng/trừ)
 - + Expression: Mức thấp nhất - phép cộng/trừ

b. Hệ thống bắt lỗi 3 cấp độ

- ERR_INVALIDEXPRESSION: Chỉ bắt ở token ĐẦU TIÊN của biểu thức
 - + Kiểm tra: Token đầu có hợp lệ để bắt đầu expression không?
 - + Token hợp lệ: +, -, identifier, số, ký tự, dấu (
 - + Ví dụ lỗi: x := WHILE → WHILE không thể bắt đầu expression
- ERR_INVALIDTERM: Chỉ bắt SAU toán tử * hoặc /
 - + Kiểm tra: Sau *, / có factor hợp lệ không?
 - + Ví dụ lỗi: x := 5 * FOR → FOR không phải factor
- ERR_INVALIDFACTOR: Bắt TẤT CẢ trường hợp còn lại
 - + Kiểm tra: Token có phải factor hợp lệ không?
 - + Ví dụ lỗi: x := (WHILE) → WHILE không phải factor

c. Các hàm xử lý biểu thức

- compileExpression() - Kiểm tra token đầu:
 - + In thông báo bắt đầu parse
 - + Switch-case kiểm tra lookAhead:
 - Nếu là SB_PLUS, SB_MINUS, TK_IDENT, TK_NUMBER, TK_CHAR, SB_LPAR: Gọi compileExpression2()
 - Ngược lại: Báo lỗi ERR_INVALIDEXPRESSION
 - + In thông báo hoàn thành
- compileExpression2() - Xử lý dấu +/- đầu biểu thức:
 - + Switch-case:
 - SB_PLUS: Ăn +, gọi compileTerm(), gọi compileExpression3()
 - SB_MINUS: Ăn -, gọi compileTerm(), gọi compileExpression3()
 - Khác: Gọi compileTerm(), gọi compileExpression3()
- compileExpression3() - Xử lý chuỗi +/- tiếp theo:
 - + Switch-case:
 - SB_PLUS: Ăn +, gọi compileTerm(), gọi compileExpression3() đệ quy
 - SB_MINUS: Ăn -, gọi compileTerm(), gọi compileExpression3() đệ quy
 - Token thuộc FOLLOW set: Break (dừng lại)
 - Khác: Báo lỗi ERR_INVALIDFACTOR
- FOLLOW set của Expression: Tập hợp các token có thể xuất hiện sau expression
 - + KW_TO (trong FOR)
 - + KW_DO (trong WHILE/FOR)
 - + SB_RPAR (đóng ngoặc)
 - + SB_COMMA (phân cách đối số)
 - + Các toán tử so sánh: =, !=, <, <=, >, >=
 - + SB_RSEL (đóng ngoặc vuông)
 - + SB_SEMICOLON (kết thúc câu lệnh)
 - + KW_END, KW_ELSE, KW_THEN

FOLLOW set giúp Parser biết khi nào dừng parse expression để không "ăn nhầm" token của cấu trúc khác.

d. Xử lý Term

- compileTerm(): Xử lý thành phần factor và phép nhân/chia
 - + Switch-case kiểm tra token đầu:
 - TK_IDENT, TK_NUMBER, TK_CHAR, SB_LPAR: Gọi compileFactor(), gọi compileTerm2()
 - Khác: Báo lỗi ERR_INVALIDFACTOR
- compileTerm2(): Xử lý chuỗi * hoặc /
 - + Switch-case:
 - SB_TIMES: Ăn *, kiểm tra token tiếp có phải factor không, gọi compileFactor(), gọi compileTerm2() đệ quy
 - SB_SLASH: Ăn /, kiểm tra tương tự
 - Token thuộc FOLLOW set của Term: Break
 - Khác: Báo lỗi ERR_INVALIDFACTOR
 - + FOLLOW set của Term = FOLLOW của Expression + {+, -}

e. Xử lý Factor

- compileFactor(): Xử lý thành phần cơ bản nhất
 - + Switch-case phân loại 4 loại factor:
 - TK_NUMBER: Số nguyên → Ăn trực tiếp
 - TK_CHAR: Ký tự → Ăn trực tiếp
 - TK_IDENT: Có thể là biến, mảng, hoặc hàm
 - + Ăn identifier
 - + Kiểm tra token tiếp:
 - SB_LSEL: Đây là mảng → Gọi compileIndexes()
 - SB_LPAR: Đây là hàm → Gọi compileArguments()
 - Khác: Đây là biến đơn → Không làm gì thêm
 - SB_LPAR: Biểu thức trong ngoặc
 - + Ăn (
 - + Gọi compileExpression() đệ quy
 - + Ăn)
 - Nếu không phải 4 loại trên: Báo lỗi ERR_INVALIDFACTOR

10. Xử lý chỉ số mảng (Indexes)

compileIndexes(): Xử lý một hoặc nhiều chỉ số

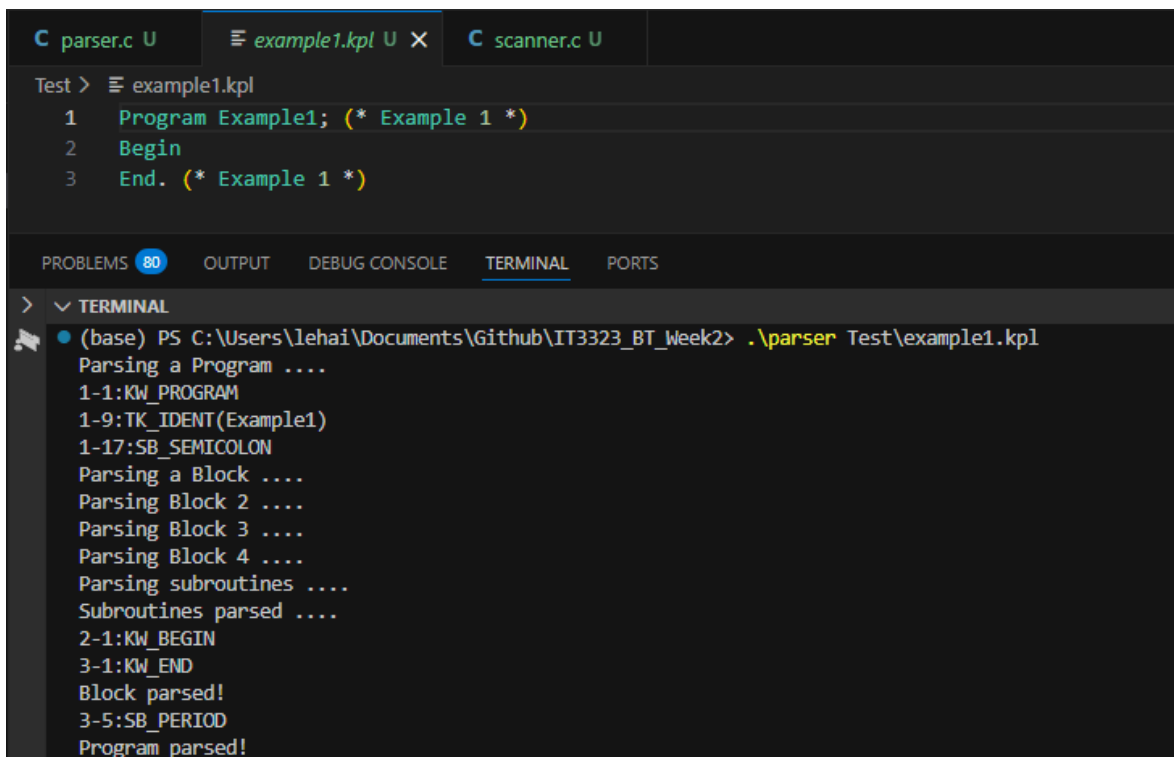
- Vòng lặp while (lookAhead->tokenType == SB_LSEL)
- Mỗi lần:
 - + Ăn SB_LSEL (I)

- + Gọi compileExpression() - chỉ số có thể là biểu thức phức tạp
- + Ăn SB_RSEL (])
- Dừng khi không còn gặp [
- Cho phép mảng nhiều chiều: matrix[i][j], cube[x][y][z]

11. Hàm khởi động Parser

- compile() - Hàm main của Parser:
 - + Tham số: fileName (tên file cần biên dịch)
 - + Trả về: IO_SUCCESS hoặc IO_ERROR
- Các bước:
 - + Gọi openInputStream() - mở file
 - Nếu lỗi: Return IO_ERROR ngay
 - + Khởi tạo token:
 - currentToken = NULL
 - lookAhead = getValidToken() (lấy token đầu từ Scanner)
 - + Gọi compileProgram() - parse toàn bộ chương trình
 - + Giải phóng bộ nhớ:
 - free(currentToken)
 - free(lookAhead)
 - + Đóng file: closeInputStream()
 - + Return IO_SUCCESS

II. Kết quả thực hiện với example1.kpl (Không lỗi)



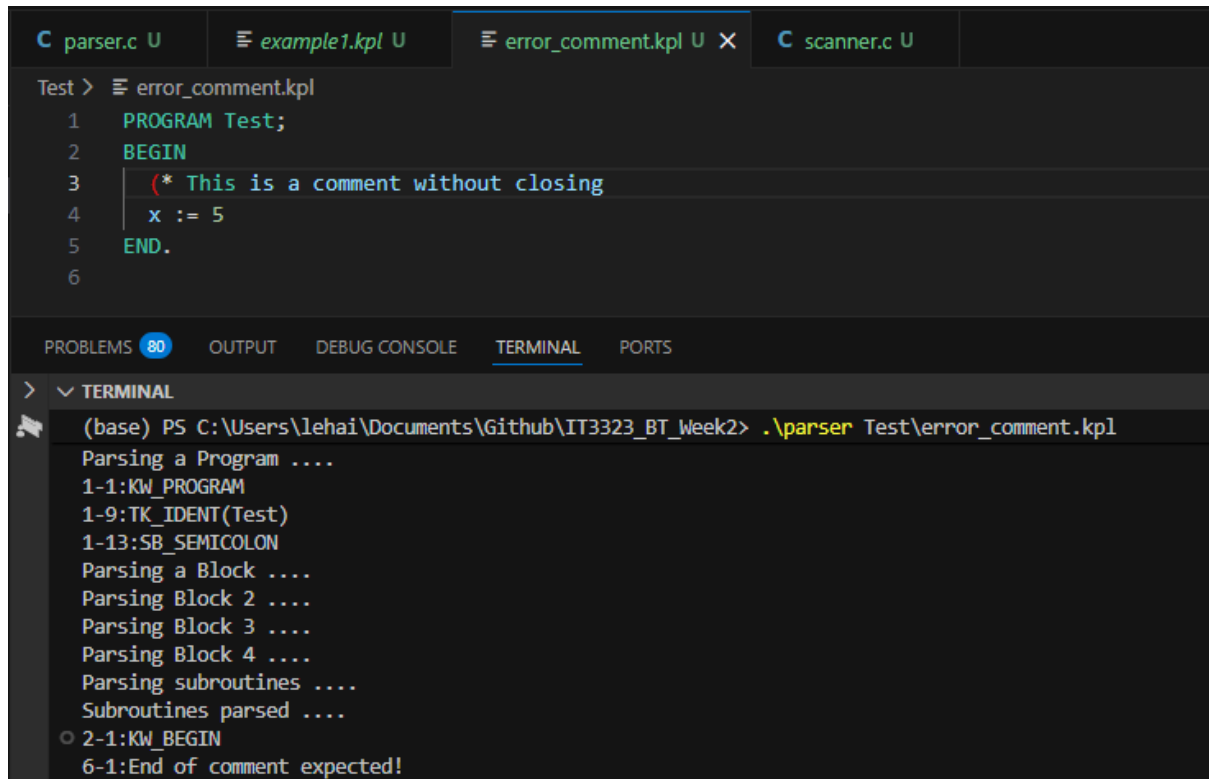
```

C parser.c U  example1.kpl U X  C scanner.c U
Test > example1.kpl
1 Program Example1; (* Example 1 *)
2 Begin
3 End. (* Example 1 *)

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> TERMINAL
• (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\example1.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Example1)
1-17:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
2-1:KW_BEGIN
3-1:KW_END
Block parsed!
3-5:SB_PERIOD
Program parsed!
  
```

III. Kết quả thực hiện với các trường hợp lỗi

1. Lỗi ERR_END_OF_COMMENT



The screenshot shows an IDE with four tabs: `parser.c U`, `example1.kpl U`, `error_comment.kpl U` (active), and `scanner.c U`. The active file contains the following code:

```
1 PROGRAM Test;
2 BEGIN
3   (* This is a comment without closing
4   x := 5
5 END.
6
```

The `TERMINAL` panel at the bottom shows the output of the parser:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_comment.kpl
Parsing a Program ....
1-1:KW PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
2-1:KW_BEGIN
6-1:End of comment expected!
```

- Đoạn code gây lỗi:
 - + Dòng 2-3: Mở comment bằng (*) nhưng không có *) đóng
 - + Scanner đọc hết file vẫn đang trong trạng thái comment
- Kết quả: 6-1:End of comment expected!
- Giải thích: Scanner phát hiện EOF (end of file) trong khi vẫn đang parse comment (state 37-38), dẫn đến lỗi tại dòng 6 cột 1.

2. Lỗi ERR_IDENT_TOO_LONG

The screenshot shows a code editor with several tabs: `parser.c`, `example1.kpl`, `error_comment.kpl`, `error_ident_long.kpl`, and `scanner.c`. The active tab is `error_ident_long.kpl`, which contains the following KPL code:

```
1 PROGRAM Test;
2 VAR
3   thisIsAVeryLongIdentifierNameThatExceedsTheMaximumLimitOf15Characters : INTEGER;
4 BEGIN
5 END.
```

The terminal output shows the parsing process:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_ident_long.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
3-3:Identification too long!
```

- Đoạn code gây lỗi: Dòng 3: Tên biến có 56 ký tự, vượt quá giới hạn `MAX_IDENT_LEN = 15`
- Kết quả: 3-3:Identification too long!
- Giải thích: Scanner đếm được 56 ký tự liên tiếp cho identifier (state 3), vượt quá giới hạn cho phép, báo lỗi tại vị trí bắt đầu identifier.

3. Lỗi `ERR_NUMBER_TOO_LONG`

The screenshot shows the same code editor with an additional tab `error_number_long.kpl`. The active tab is `error_number_long.kpl`, which contains the following KPL code:

```
1 PROGRAM Test;
2 CONST
3   HUGE = 99999999999999999999;
4 BEGIN
5 END.
```

The terminal output shows the parsing process:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_number_long.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
2-1:KW_CONST
3-3:TK_IDENT(HUGE)
3-8:SB_EQ
3-10:Value of integer number exceeds the range!
```

- Đoạn code gây lỗi: Dòng 3: Số 99999999999999999999 có 20 chữ số, vượt quá : Giới hạn 10 chữ số, Giá trị `INT_MAX` (2,147,483,647)
- Giải thích: Scanner (state 7) phát hiện số có > 10 chữ số hoặc vượt `INT_MAX`, báo lỗi tại vị trí bắt đầu số.

4. Lỗi `ERR_INVALIDCHARCONSTANT`

```
C parser.c U  error_char_invalid.kpl U X  C scanner.c U
Test > error_char_invalid.kpl
1  PROGRAM Test;
2  CONST
3    LETTER = 'AB';
4  BEGIN
5  END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> v TERMINAL
• (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_char_invalid.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
2-1:KW_CONST
3-3:TK_IDENT(LETTER)
3-10:SB_EQ
3-13:Invalid const char!
```

- Đoạn code gây lỗi: Dòng 3: Character constant 'AB' chứa 2 ký tự thay vì 1 ký tự duy nhất
- Giải thích: Scanner (state 31-34) yêu cầu char constant phải có đúng 1 ký tự giữa hai dấu nháy đơn. Phát hiện 'A' (OK) nhưng tiếp theo là 'B' thay vì ', dẫn đến lỗi.

5. Lỗi ERR_INVALID_SYMBOL

```
C parser.c U  error_char_invalid.kpl U  error_symbol.kpl U X  C scanner.c U
Test > error_symbol.kpl
1  PROGRAM Test;
2  VAR
3    x : INTEGER;
4  BEGIN
5    x := 5 @ 3
6  END.
7

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> v TERMINAL
• (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_symbol.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
3-3:TK_IDENT(x)
• 3-5:SB_COLON
• 3-7:KW_INTEGER
3-14:SB_SEMICOLON
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
4-1:KW_BEGIN
Parsing an assign statement ....
5-3:TK_IDENT(x)
5-6:SB_ASSIGN
Parsing an expression
5-8:TK_NUMBER(5)
5-10:Invalid symbol!
```

- Đoạn code gây lỗi:
 - + Dòng 5: Ký tự @ không thuộc bảng mã ký tự hợp lệ của KPL
 - + @ có mã ASCII không được định nghĩa trong charCodes[]
- Giải thích: Scanner gặp ký tự có charCode = CHAR_UNKNOWN (state 43), không thể phân loại thành bất kỳ token nào, báo lỗi ngay lập tức.

6. Lỗi ERR_INVALID_CONSTANT

```

C parser.c U  error_constant.kpl U X  scanner.c U
Test > error_constant.kpl
1  PROGRAM Test;
2  CONST
3  | X = END;
4  BEGIN
5  END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> ▼ TERMINAL
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_constant.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
2-1:KW_CONST
3-3:TK_IDENT(X)
3-5:SB_EQ
3-7:Invalid constant!
  
```

- Đoạn code gây lỗi:
 - + Dòng 3: Sử dụng keyword END làm giá trị constant
 - + Parser mong đợi TK_NUMBER, TK_IDENT (tên hằng khác), hoặc TK_CHAR
- Giải thích: Hàm compileConstant2() kiểm tra token, phát hiện KW_END không phải constant hợp lệ, gọi error(ERR_INVALID_CONSTANT).

7. Lỗi ERR_INVALIDTYPE

```
Test > error_type.kpl
1  PROGRAM Test;
2  TYPE
3    MyType = PROGRAM;
4  BEGIN
5  END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> ▼ TERMINAL
● (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_type.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
2-1:KW_TYPE
3-3:TK_IDENT(MyType)
3-10:SB_EQ
● 3-12:Invalid type!
```

- Đoạn code gây lỗi:
 - + Dòng 3: Sử dụng keyword PROGRAM làm định nghĩa type
 - + Parser chỉ chấp nhận: INTEGER, CHAR, ARRAY, hoặc identifier
- Giải thích: Hàm compileType() switch-case không match với KW_PROGRAM, rơi vào default case, báo lỗi ERR_INVALIDTYPE.

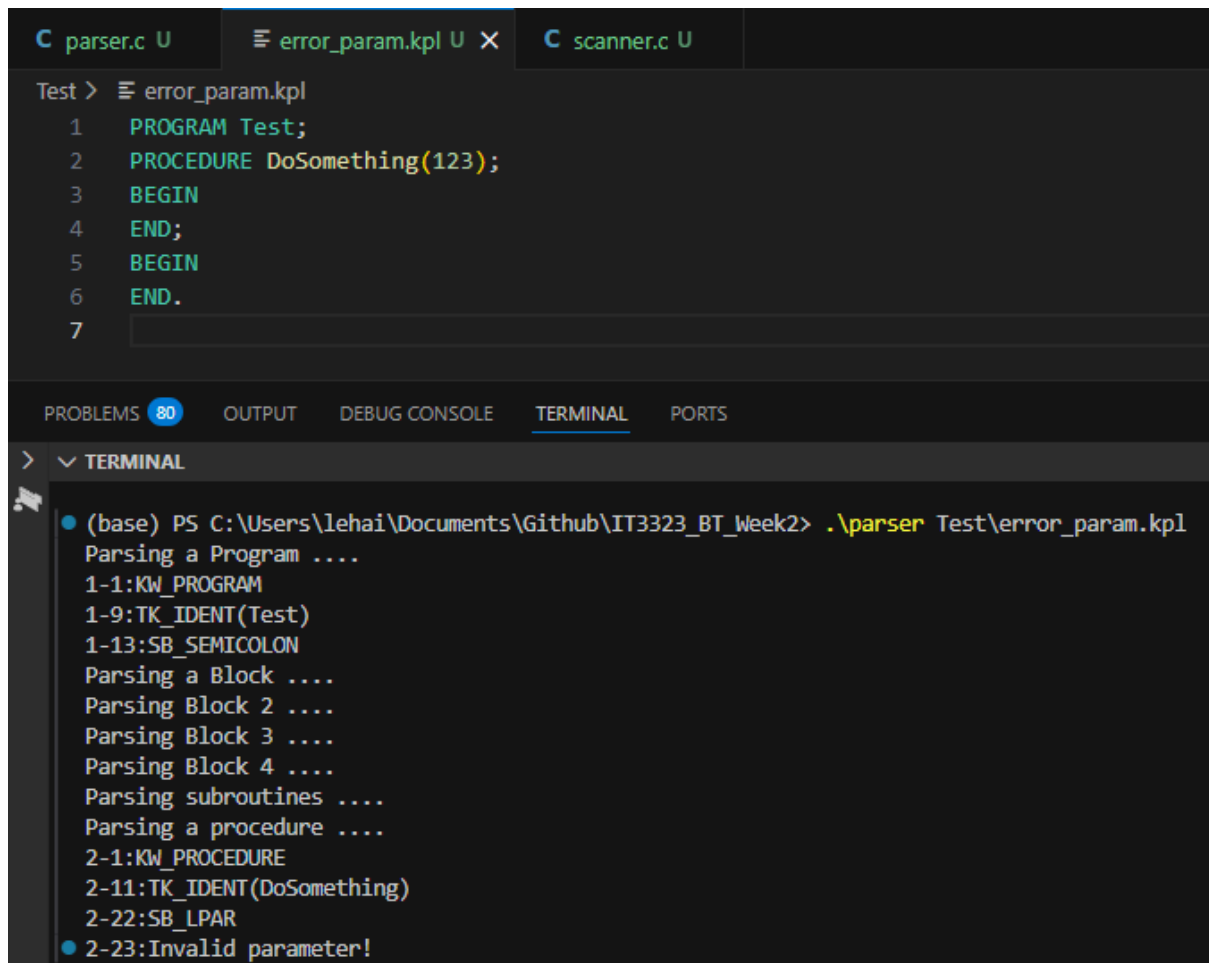
8. Lỗi ERR_INVALIDBASICTYPE

```
C parser.c U  error_basictype.kpl U X  C scanner.c U
Test > error_basictype.kpl
1  PROGRAM Test;
2  VAR
3      x : INTEGER;
4
5  FUNCTION Calculate(a: INTEGER): BOOLEAN;
6  BEGIN
7      Calculate := a
8  END;
9
10 BEGIN
11     x := Calculate(5);
12     CALL WriteI(x)
13 END.
```

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_basictype.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
3-5:TK_IDENT(x)
3-5:TK_IDENT(x)
3-5:TK_IDENT(x)
3-7:SB_COLON
3-5:TK_IDENT(x)
3-5:TK_IDENT(x)
3-7:SB_COLON
3-9:KW_INTEGER
3-16:SB_SEMICOLON
Parsing Block 4 ....
Parsing subroutines ....
Parsing a function ....
5-1:KW_FUNCTION
5-10:TK_IDENT(Calculate)
5-19:SB_LPAR
5-20:TK_IDENT(a)
5-21:SB_COLON
5-23:KW_INTEGER
5-30:SB_RPAR
5-31:SB_COLON
5-33:Invalid basic type!
```

- Đoạn code gây lỗi:
 - + Dòng : Kiểu trả về BOOLEAN không phải basic type
 - + KPL chỉ hỗ trợ INTEGER và CHAR làm kiểu trả về function
- Giải thích: Hàm compileBasicType() trong compileFuncDecl() chỉ chấp nhận KW_INTEGER hoặc KW_CHAR, phát hiện TK_IDENT(BOOLEAN), báo lỗi.

9. Lỗi ERR_INVALIDPARAM



The screenshot shows a code editor with three tabs: `parser.c`, `error_param.kpl`, and `scanner.c`. The `error_param.kpl` tab is active, displaying the following code:

```
1 PROGRAM Test;
2 PROCEDURE DoSomething(123);
3 BEGIN
4 END;
5 BEGIN
6 END.
7
```

Below the code editor, the `TERMINAL` tab is active, showing the output of the command `.\parser Test\error_param.kpl`:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_param.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
Parsing Block 4 ....
Parsing subroutines ....
Parsing a procedure ....
2-1:KW_PROCEDURE
2-11:TK_IDENT(DoSomething)
2-22:SB_LPAR
2-23:Invalid parameter!
```

- Đoạn code gây lỗi:
 - + Dòng 2: Parameter 123 là số thay vì identifier
 - + Parser yêu cầu tham số phải là: tên : kiểu hoặc VAR tên : kiểu
- Giải thích: Hàm `compileParam()` switch-case kiểm tra token đầu, `TK_NUMBER` không match với `TK_IDENT` hoặc `KW_VAR`, rơi vào default, báo `ERR_INVALIDPARAM`.

10. Lỗi ERR_INVALIDSTATEMENT

The screenshot shows a Visual Studio Code editor with four tabs: `parser.c`, `error_param.kpl`, `error_statement.kpl`, and `scanner.c`. The `error_statement.kpl` tab is active, displaying the following code:

```
1 PROGRAM Test;
2 VAR x : INTEGER;
3 BEGIN
4     x := 5;
5     TYPE
6 END.
```

Below the code editor, the `TERMINAL` tab is active, showing the output of the command `.\parser Test\error_statement.kpl`. The output is as follows:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_statement.kpl
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
2-5:TK_IDENT(x)
2-7:SB_COLON
2-9:KW_INTEGER
2-16:SB_SEMICOLON
Parsing Block 4 ....
● Parsing subroutines ....
Subroutines parsed ....
3-1:KW_BEGIN
Parsing an assign statement ....
4-3:TK_IDENT(x)
4-6:SB_ASSIGN
Parsing an expression
4-8:TK_NUMBER(5)
Expression parsed
Assign statement parsed ....
4-9:SB_SEMICOLON
5-3:Invalid statement!
```

- Đoạn code gây lỗi:
 - + Dòng 5: Keyword TYPE xuất hiện trong phần thân BEGIN...END
 - + Đây là vị trí chỉ chấp nhận statement (gán, CALL, IF, WHILE, FOR, BEGIN)
- Giải thích: Hàm `compileStatement()` switch-case không match với `KW_TYPE` (không phải token bắt đầu statement nào), rơi vào default, báo `ERR_INVALIDSTATEMENT`.

11. Lỗi `ERR_INVALIDARGUMENTS`

The screenshot shows an IDE with several tabs: `parser.c`, `error_param.kpl`, `error_statement.kpl`, `error_arguments.kpl`, and `scanner.c`. The active file is `error_arguments.kpl`, which contains the following KPL code:

```
1 PROGRAM Test;
2 VAR
3   x : INTEGER;
4
5 FUNCTION Add(a: INTEGER; b: INTEGER): INTEGER;
6 BEGIN
7   Add := a + b
8 END;
9
10 BEGIN
11   x := Add(5, );
12   CALL WriteI(x)
13 END.
```

The terminal window shows the output of the command `.\parser Test\error_arguments.kpl`:

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_arguments.kpl
8-4:SB_SEMICOLON
Function parsed ....
Subroutines parsed ....
10-1:KW_BEGIN
Parsing an assign statement ....
11-5:TK_IDENT(x)
11-8:SB_ASSIGN
Parsing an expression
11-10:TK_IDENT(Add)
11-13:SB_LPAR
Parsing an expression
11-14:TK_NUMBER(5)
Expression parsed
11-15:SB_COMMA
11-17:Invalid arguments!
```

- Đoạn code gây lỗi:
 - + Thiếu argument thứ 2 trong `Add(5,)`
 - + Sau dấu phẩy phải có một expression, nhưng gặp `)`
- Giải thích: Hàm `compileArguments2()` sau khi ăn dấu phẩy, kiểm tra token tiếp không phải token hợp lệ bắt đầu expression (gặp `SB_RPAR`), báo `ERR_INVALIDARGUMENTS`.

12. Lỗi `ERR_INVALIDCOMPARATOR`

```
Test > error_comparator.kpl
1 PROGRAM Test;
2 VAR
3     x : INTEGER;
4     y : INTEGER;
5 BEGIN
6     x := 5;
7     y := 10;
8     IF x + y THEN
9         CALL WriteI(x)
10    ELSE
11        CALL WriteI(y)
12 END.
```

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS

> ▾ TERMINAL

```
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_comparator.kpl
7-5:TK_IDENT(y)
7-8:SB_ASSIGN
Parsing an expression
7-10:TK_NUMBER(10)
Expression parsed
Assign statement parsed ....
7-12:SB_SEMICOLON
Parsing an if statement ....
8-5:KW_IF
Parsing an expression
8-8:TK_IDENT(x)
8-10:SB_PLUS
8-12:TK_IDENT(y)
Expression parsed
8-14:Invalid comparator!
```

- Đoạn code gây lỗi:
 - + Điều kiện IF thiếu toán tử so sánh
 - + Có $x + y$ (expression) nhưng không có $=$ / $!=$ / $<$ / $<=$ / $>$ / $>=$ theo sau
- Giải thích: Hàm `compileCondition2()` sau khi parse expression đầu, kiểm tra token tiếp (`KW_THEN`) không phải toán tử so sánh, rơi vào default case, báo `ERR_INVALIDCOMPARATOR`.

13. Lỗi `ERR_INVALIDEXPRESSION`

```
Test > error_expression.kpl
1 PROGRAM Test;
2 VAR x : INTEGER;
3 BEGIN
4   x := WHILE
5 END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> v TERMINAL
(base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_comparator.kpl
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
2-5:TK_IDENT(x)
2-7:SB_COLON
2-9:KW_INTEGER
2-16:SB_SEMICOLON
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
3-1:KW_BEGIN
Parsing an assign statement ....
4-3:TK_IDENT(x)
4-6:SB_ASSIGN
Parsing an expression
4-8:Invalid expression!
```

- Đoạn code gây lỗi:
 - + Dòng 4: Keyword WHILE ở vị trí cần một expression
 - + WHILE không phải token hợp lệ bắt đầu expression (+, -, identifier, số, ký tự, dấu mở ngoặc)
- Giải thích: Hàm compileExpression() kiểm tra token đầu (lookAhead = KW_WHILE), không match với bất kỳ case nào trong switch, rơi vào default, báo ERR_INVALIDEXPRESSION.

14. Lỗi ERR_INVALIDTERM

```
Test > error_term.kpl
1 PROGRAM Test;
2 VAR x : INTEGER;
3 BEGIN
4   x := 5 * FOR
5 END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> TERMINAL
• (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_term.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(Test)
1-13:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
2-5:TK_IDENT(x)
2-7:SB_COLON
2-9:KW_INTEGER
2-16:SB_SEMICOLON
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
3-1:KW_BEGIN
Parsing an assign statement ....
4-3:TK_IDENT(x)
4-6:SB_ASSIGN
Parsing an expression
4-8:TK_NUMBER(5)
4-10:SB_TIMES
4-12:Invalid term!
```

- Đoạn code gây lỗi:
 - + Dòng 4: Sau toán tử * là keyword FOR thay vì factor hợp lệ
 - + Parser mong đợi: identifier, số, ký tự, hoặc dấu mở ngoặc
- Giải thích: Hàm compileTerm2() sau khi ăn SB_TIMES, kiểm tra token tiếp (KW_FOR) không phải factor hợp lệ, báo ERR_INVALIDTERM.

15.Lỗi ERR_INVALIDFACTOR

```
Test > error_factor.kpl
1 PROGRAM TestFactor;
2 VAR x : INTEGER;
3 BEGIN
4     x := - WHILE + 5 ;
5 END.
6

PROBLEMS 80 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> v TERMINAL
• (base) PS C:\Users\lehai\Documents\Github\IT3323_BT_Week2> .\parser Test\error_factor.kpl
Parsing a Program ....
1-1:KW_PROGRAM
1-9:TK_IDENT(TestFactor)
1-19:SB_SEMICOLON
Parsing a Block ....
Parsing Block 2 ....
Parsing Block 3 ....
2-1:KW_VAR
2-5:TK_IDENT(x)
2-7:SB_COLON
2-9:KW_INTEGER
2-16:SB_SEMICOLON
Parsing Block 4 ....
Parsing subroutines ....
Subroutines parsed ....
3-1:KW_BEGIN
Parsing an assign statement ....
4-5:TK_IDENT(x)
4-8:SB_ASSIGN
Parsing an expression
4-10:SB_MINUS
4-12:Invalid factor!
```

- Đoạn code gây lỗi:
x := - WHILE + 5;
- Giải thích: Sau dấu - Parser mong đợi một factor (số, ký tự, biến, hoặc (biểu thức)), nhưng lại gặp keyword WHILE, không phải factor hợp lệ. Vì vậy compileFactor() báo lỗi ERR_INVALIDFACTOR.