BÁO CÁO THỰC HÀNH
XÂY DỰNG CHƯƠNG TRÌNH DỊCH
BÀI 4: QUẢN LÝ PHẠM VI

Họ và tên: Bùi Quang Hưng

Mã số sinh viên: 20225849

I. Nội dung chính về quản lý phạm vi

1. Mục đích chính

Quản lý phạm vi (Scope Management) là một phần thiết yếu trong quá trình phân tích ngữ nghĩa của compiler, đảm bảo tính đúng đắn của chương trình trước khi sinh mã. Mục đích chính của quản lý phạm vi trong project này bao gồm:

- Kiểm tra khai báo và tính duy nhất: Đảm bảo mọi identifier được sử dụng đều đã được khai báo trước trong phạm vi hiện tại hoặc các phạm vi bao quanh. Ngăn chặn khai báo trùng lặp trong cùng một phạm vi.
- Kiểm tra ngữ cảnh và kiểu dữ liệu: Kiểm tra identifier được sử dụng đúng ngữ cảnh, đảm bảo tính nhất quán về kiểu, kiểm tra LValue hợp lệ.
- Quản lý bằng bảng ký hiệu và phạm vi lồng nhau: Tạo và hủy phạm vi động: mỗi khối mới (program, procedure, function) tạo ra một phạm vi mới, tự động hủy khi thoát khối. Tra cứu identifier theo thứ tự phạm vi và lưu trữ đầy đủ thông tin.

2. Các hàm kiểm tra trong senmantics.c

Hàm lookupObject – Tìm kiếm đối tượng theo phạm vi

Hàm lookupObject thực hiện tìm kiếm một identifier trong các phạm vi lồng nhau, tuân theo quy tắc phạm vi gần nhất (nearest scope rule).

- Giai đoạn 1: Tìm kiếm trong chuỗi phạm vi lồng nhau
  - Bắt đầu từ phạm vi hiện tại (symtab -> currentScope)
  - Dùng vòng lặp while duyệt ngược qua con trỏ outer
  - Tại mỗi phạm vi gọi findObject(scope->objList, name) để quét danh sách đối tượng
  - Nếu tìm thấy, trả về ngay lập tức với đối tượng tìm được
  - Nếu không, tiếp tục với phạm vi ngoài (scope = scope->outer)
- Giai đoạn 2: Tìm kiếm trong phạm vi toàn cục
  - Kích hoạt khi vòng lặp kết thúc mà chưa tìm thấy (scope == NULL)
  - Gọi findObject(symtab->globalObjectList, name) để tìm trong danh sách toàn cục
  - globalObjectList chứa các hàm/thủ tục dựng sẵn: READC, READI, WRITEI, WRITEC, WRITELN

    - Được khởi tạo trong initSymTab() và có thể truy cập từ mọi nơi trong chương trình
- Giai đoạn 3: Xử lý không tìm thấy
    - Trả về NULL nếu cả hai giai đoạn thất bại
    - Báo hiệu đối tượng không tồn tại trong bất kỳ phạm vi nào
    - Các hàm gọi lookupObject sẽ xử lý lỗi ERR_UNDECLARED_* tương ứng

Các hàm kiểm tra đối tượng

- Hàm **checkFreshIdent:** Kiểm tra tên mới, kiểm tra định danh chưa bị trùng trong phạm vi hiện tại
- Hàm **checkDeclaredIdent:** Kiểm tra định danh tổng quát, kiểm tra định danh đã được khai báo, không phân biệt loại
- Hàm **checkDeclaredConstant:** Kiểm tra hằng số, kiểm tra định danh là hằng số đã khai báo
- Hàm **checkDeclaredType:** Kiểm tra kiểu người dùng, kiểm tra định danh là kiểu người dùng định nghĩa
- Hàm **checkDeclaredVariable:** Kiểm tra biến, kiểm tra định dạng là biến đã khai báo
- Hàm **checkDeclaredProcedure:** Kiểm tra thủ tục, kiểm tra định danh là thủ tục đã khai báo
- Hàm **checkDeclaredLValueIdent:** Kiểm tra vế trái phép gán, kiểm tra định danh có thể đứng vế trái (biến, tham số, tên hàm hiện tại)
- Hàm **checkDeclaredFunction:** Kiểm tra hàm, kiểm tra định danh là hàm đã khai báo

Các hàm kiểm tra kiểu dữ liệu

- Hàm checkIntType: Kiểm tra kiểu số nguyên
- Hàm checkCharType: Kiểm tra kiểu ký tự
- Hàm checkArrayType: Kiểm tra kiểu mảng
- Hàm checkBasicType: Kiểm tra kiểu cơ bản
- Hàm checkTypeEquality: Kiểm tra hai kiểu tương đương

Tích hợp vào file parser.c

- Gọi hàm kiểm tra tên mới (checkFreshIdent) tại 6 điểm:
    - compileBlock: Khai báo hằng - đảm bảo tên hằng chưa dùng trong phạm vi hiện tại
    - compileBlock2: Khai báo kiểu - ngăn trùng tên kiểu người dùng
    - compileBlock3: Khai báo biến - ngăn trùng tên biến
    - compileParam: Khai báo tham số - ngăn trùng tên tham số trong danh sách
    - compileFuncDecl: Khai báo hàm - ngăn trùng tên hàm
    - compileProDecl: Khai báo thủ tục - ngăn trùng tên thủ tục
- Kiểm tra tham chiếu tại nhiều điểm:

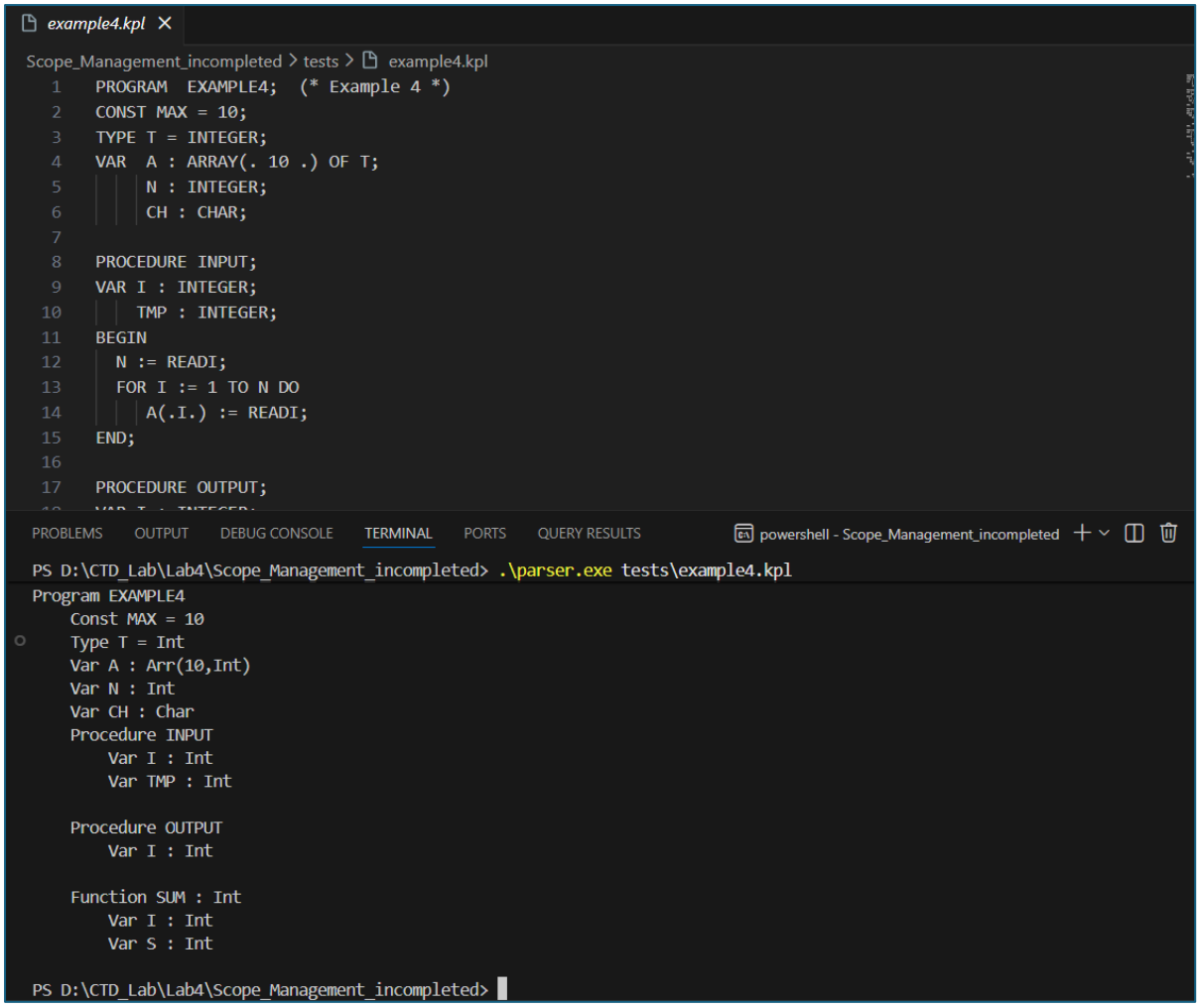
- compileUnsignedConstant, compileConstant2 → checkDeclaredConstant: Kiểm tra hằng đã khai báo khi dùng trong biểu thức hằng. Dùng duplicateConstantValue để sao chép giá trị
- compileType → checkDeclaredType: Kiểm tra kiểu người dùng định nghĩa. Dùng duplicateType để sao chép cấu trúc kiểu
- compileForSt → checkDeclaredVariable: Kiểm tra biến điều khiển vòng lặp FOR. Chấp nhận cả biến và tham số
- compileCallSt → checkDeclaredProcedure: Kiểm tra thủ tục trong lệnh CALL. Lấy paramList để kiểm tra đối số
- compileLValue → checkDeclaredLValueIdent: Kiểm tra vế trái phép gán. Xử lý 3 trường hợp: biến, tham số, tên hàm hiện tại
- compileFactor → checkDeclaredIdent: Kiểm tra định danh trong biểu thức. Sau đó phân loại theo kind để xử lý

Mô tả một hàm điển hình trong semantics.c

Hàm checkDeclaredLValueIdent là một hàm quan trọng để kiểm tra tính hợp lệ của vế trái trong phép gán.

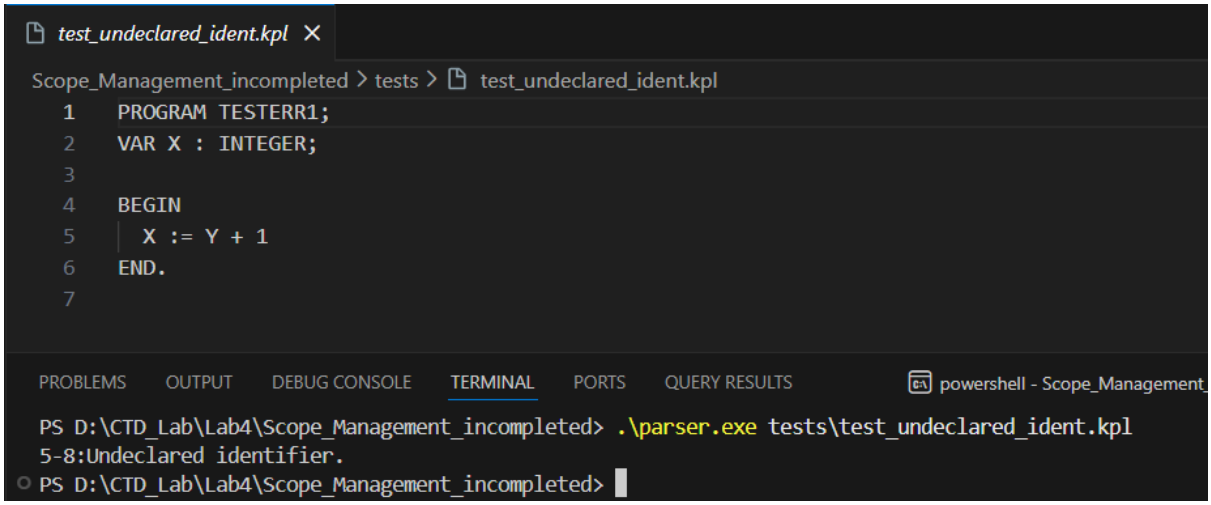
- Bước 1: Tìm kiếm đối tượng
  - Gọi lookupObject(name) để áp dụng luật phạm vi gần nhất
  - lookupObject tìm từ currentScope, qua outer, đến globalObjectList
  - Nếu trả về NULL → Đối tượng không tồn tại → Báo lỗi ERR_UNDECLARED_IDENT
- Bước 2: Kiểm tra các loại **lvalue** hợp lệ: so sánh obj -> kind với ba loại hợp lệ:
  - OBJ_VARIABLE: Biến thông thường, luôn có thể gán
  - OBJ_PARAMETER: Tham số của procedure/function, có thể gán
  - OBJ_FUNCTION: Tên hàm, chỉ hợp lệ trong thân hàm đó để gán giá trị trả về
- Bước 3: Xử lý các trường hợp không hợp lệ
  - Nếu obj->kind không thuộc ba loại trên (ví dụ: hằng, thủ tục, kiểu)
  - Báo lỗi ERR_INVALID_LVALUE với vị trí chính xác
  - Trả về obj (thực tế không bao giờ đến do error() đã thoát chương trình)

II. Kết quả thực hiện với file EXAMPLE4.KPL không có lỗi



III. Thực hiện chương trình với các ví dụ gây lỗi

1. Lỗi ERR_UNDECLARED_IDENT

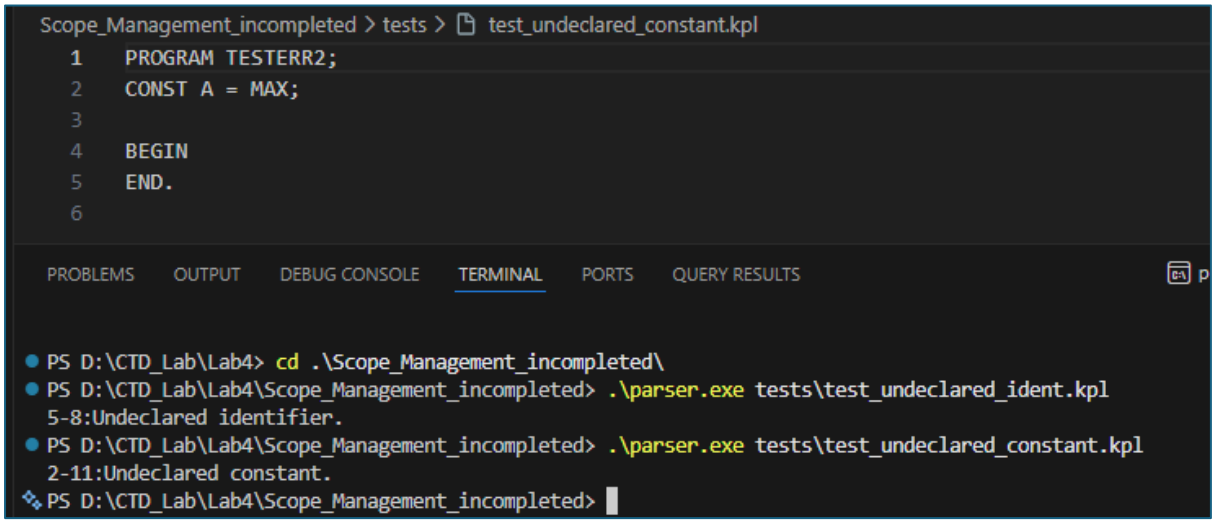


- **Dòng lỗi:** Dòng 5, cột 8 - identifier Y
- **Nguyên nhân:** Biến Y được sử dụng trong biểu thức Y + 1 nhưng chưa được khai báo. Chỉ có X được khai báo
- **Luồng xử lý:**

- Parser xử lý phép gán X := Y + 1 trong compileAssignSt()
- Gọi compileExpression() để xử lý vế phải Y + 1
- compileTerm() → compileFactor() gặp identifier Y
- compileFactor() gọi checkDeclaredIdent("Y")
- checkDeclaredIdent() gọi lookupObject("Y") → Không tìm thấy, trả về NULL
- Báo lỗi ERR_UNDECLARED_IDENT

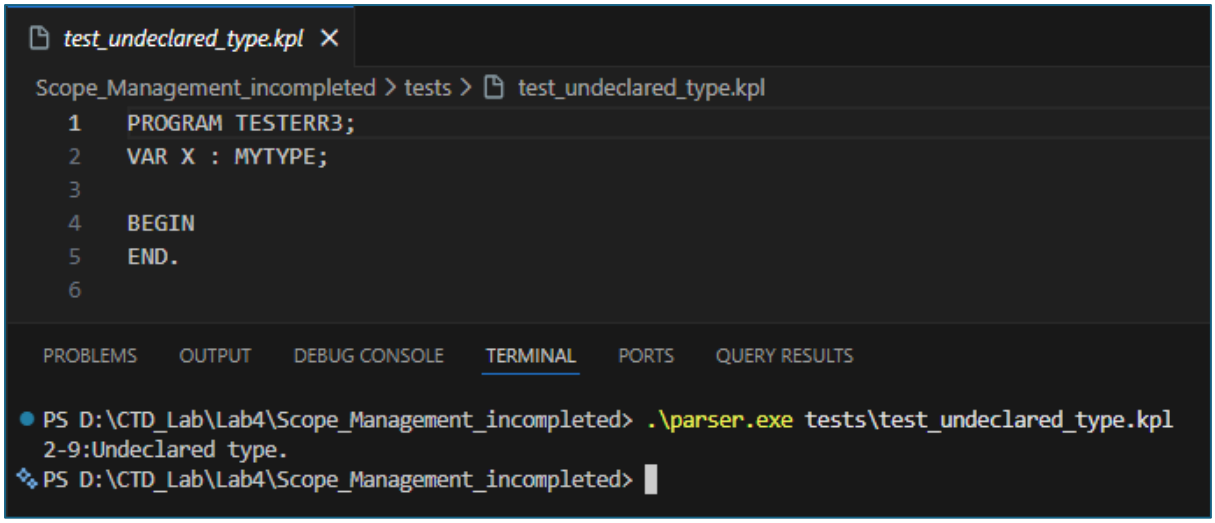
- **Hàm xử lý lỗi:** checkDeclaredIdent() trong semantics.c, được gọi từ compileFactor trong parser.c

2. Lỗi ERR_UNDECLARED_CONSTANT



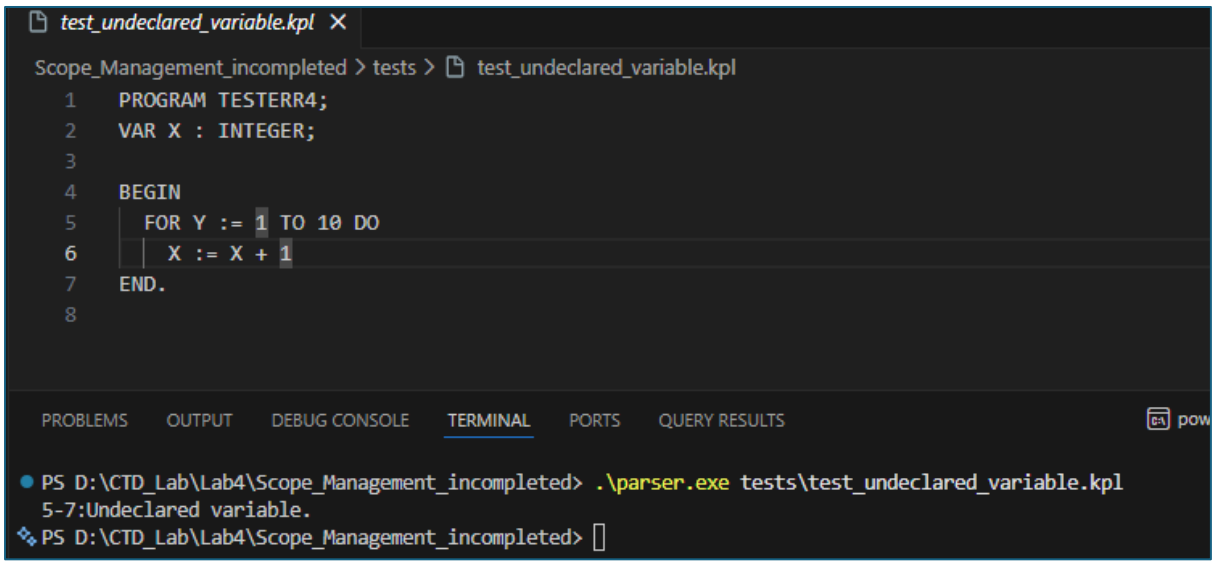
- **Dòng lỗi**: Dòng 2, cột 11 - identifier B
- **Nguyên nhân**: Khai báo hằng A = B, trong đó B phải là một hằng số đã được khai báo trước, nhưng B chưa tồn tại
- **Luồng xử lý**:
  - Parser xử lý compileBlock() → gặp CONST A = B
  - Gọi compileConstant() → compileConstant2() khi gặp identifier B
  - compileConstant2() gọi checkDeclaredConstant("B")
  - checkDeclaredConstant() gọi lookupObject("B") → Trả về NULL
  - Báo lỗi ERR_UNDECLARED_CONSTANT
- **Hàm xử lý:** checkDeclaredConstant() trong semantics.c, được gọi từ compileUnsignedConstant hoặc compileConstant2

3. Lỗi ERR_UNDECLARED_TYPE



- **Dòng lỗi**: Dòng 2, cột 9 - identifier MYTYPE
- **Nguyên nhân**: Biến X được khai báo với kiểu MYTYPE, nhưng MYTYPE chưa được định nghĩa trong phần TYPE
- **Luồng xử lý**:
  - Parser xử lý compileBlock3() → gặp VAR X : MYTYPE
  - Gọi compileType() để lấy kiểu của biến
  - compileType() gặp TK_IDENT, gọi checkDeclaredType("MYTYPE")
  - checkDeclaredType() gọi lookupObject("MYTYPE") → Trả về NULL
  - Báo lỗi ERR_UNDECLARED_TYPE
- **Hàm xử lý:** checkDeclaredType() trong semantics.c, được gọi từ compileType trong parser.c

4. Lỗi ERR_UNDECLARED_VARIABLE

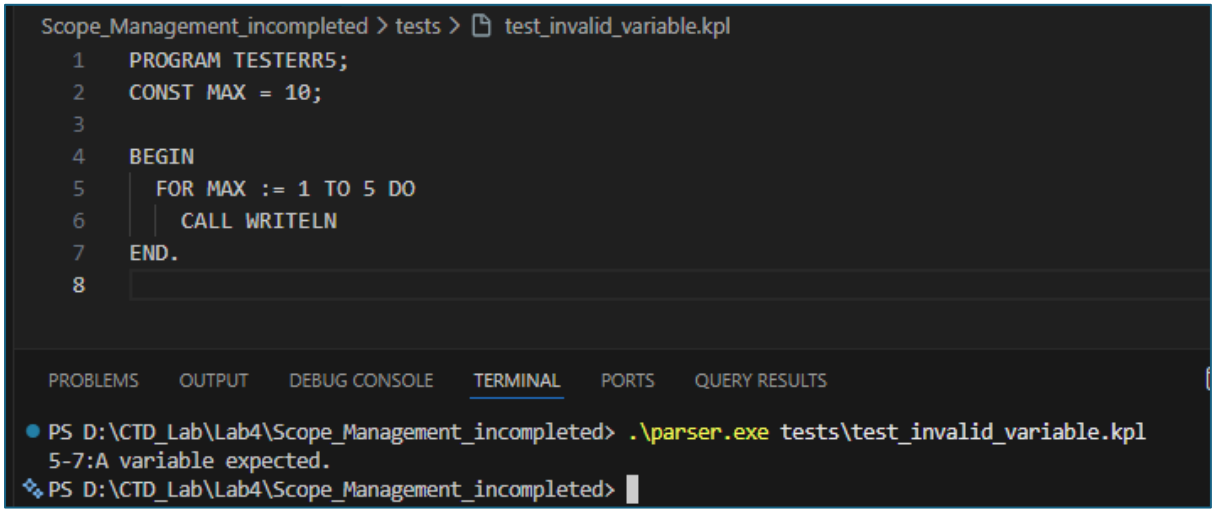


- **Dòng lỗi**: Dòng 5, cột 7 - identifier Y
- **Nguyên nhân**: Vòng lặp FOR sử dụng Y làm biến điều khiển, nhưng Y chưa được khai báo trong phần VAR. Chỉ có biến X được khai báo.
- **Luồng xử lý**:

- Parser gọi compileForSt() khi gặp từ khóa FOR ở dòng 5
- Gặp identifier Y sau FOR, gọi checkDeclaredVariable("Y")
- checkDeclaredVariable() gọi lookupObject("Y") để tìm trong các phạm vi
- lookupObject() tìm từ currentScope → outer → globalObjectList
- Không tìm thấy Y ở bất kỳ phạm vi nào, trả về NULL
- checkDeclaredVariable() phát hiện obj == NULL
- Báo lỗi ERR_UNDECLARED_VARIABLE tại dòng 5, cột 7

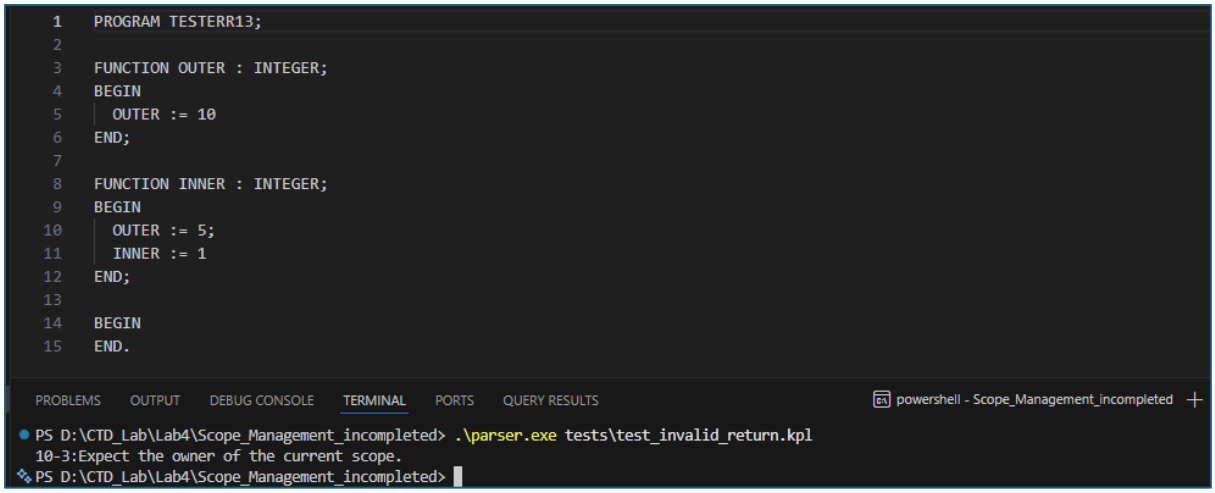
- **Hàm xử lý:** checkDeclaredVariable() trong semantics.c, được gọi từ compileForSt

5. Lỗi ERR_INVALID_VARIABLE



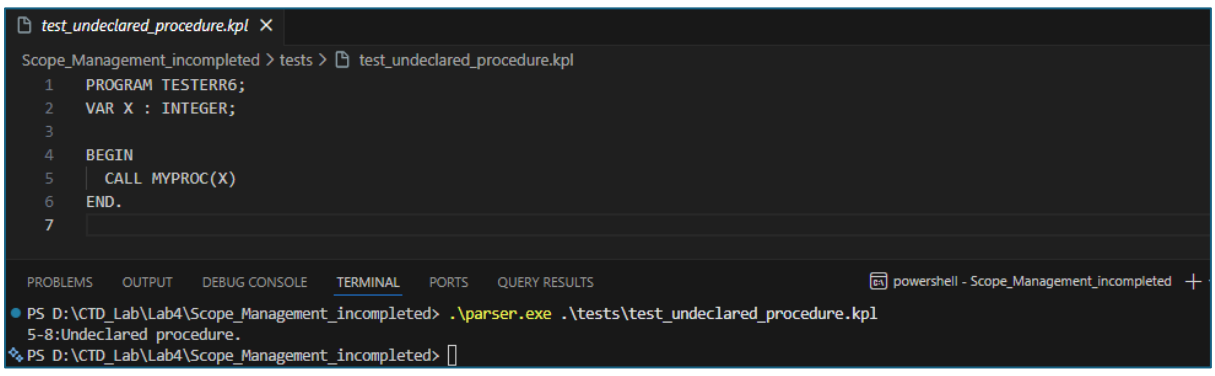
- **Dòng lỗi**: Dòng 5, cột 7 - identifier MAX
- **Nguyên nhân**: MAX được khai báo là hằng số (CONST MAX = 10), không phải biến. Vòng lặp FOR yêu cầu biến điều khiển phải là biến (OBJ_VARIABLE) vì giá trị của nó sẽ thay đổi trong mỗi vòng lặp. Hằng số có giá trị không đổi nên không thể làm biến điều khiển.
- **Luồng xử lý**:
  - Parser gọi compileForSt() khi gặp từ khóa FOR ở dòng 5
  - Gặp identifier MAX, gọi checkDeclaredVariable("MAX")
  - checkDeclaredVariable() gọi lookupObject("MAX")
  - lookupObject() tìm thấy MAX trong currentScope với kind = OBJ_CONSTANT
  - checkDeclaredVariable() kiểm tra obj->kind != OBJ_VARIABLE
  - Phát hiện MAX là OBJ_CONSTANT, không phải OBJ_VARIABLE
  - Báo lỗi ERR_INVALID_VARIABLE tại dòng 5, cột 7
- **Hàm xử lý:** checkDeclaredVariable() trong semantics.c, kiểm tra kind sau khi tìm thấy đối tượng

6. Lỗi ERR_INVALID_RETURN



- **Dòng lỗi**: Dòng 10, cột 3 - identifier OUTER
- **Nguyên nhân**: Trong thân function INNER, cố gắng gán giá trị cho OUTER (một function khác). Tên function chỉ có thể được gán giá trị trong chính thân function đó để trả về kết quả (return statement). Không thể gán giá trị cho function khác từ bên ngoài function đó.
- **Luồng xử lý**:
    - Parser xử lý phép gán OUTER := 5 trong thân function INNER (dòng 10)
    - compileAssignSt() gọi compileLValue() để xử lý vế trái
    - compileLValue() gọi checkDeclaredLValueIdent("OUTER")
    - checkDeclaredLValueIdent() gọi lookupObject("OUTER") → Tìm thấy OUTER với kind = OBJ_FUNCTION
    - Kiểm tra đặc biệt cho OBJ_FUNCTION:
        - Function chỉ là lvalue hợp lệ khi là owner của scope hiện tại
        - So sánh obj với symtab->currentScope->owner
        - obj = function OUTER
        - currentScope->owner = function INNER (scope hiện tại)
        - Không bằng nhau!
    - Báo lỗi ERR_INVALID_RETURN tại dòng 10, cột 3
- **Hàm xử lý: check**DeclaredLValueIdent() trong semantics.c, kiểm tra owner của scope

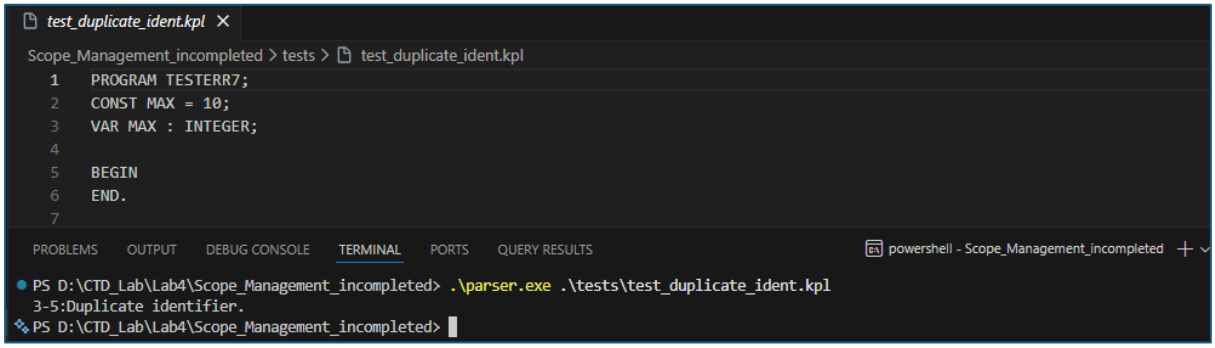
7. Lỗi ERR_UNDECLARED_PROCEDURE



- **Dòng lỗi**: Dòng 5, cột 8 - identifier MYPROC

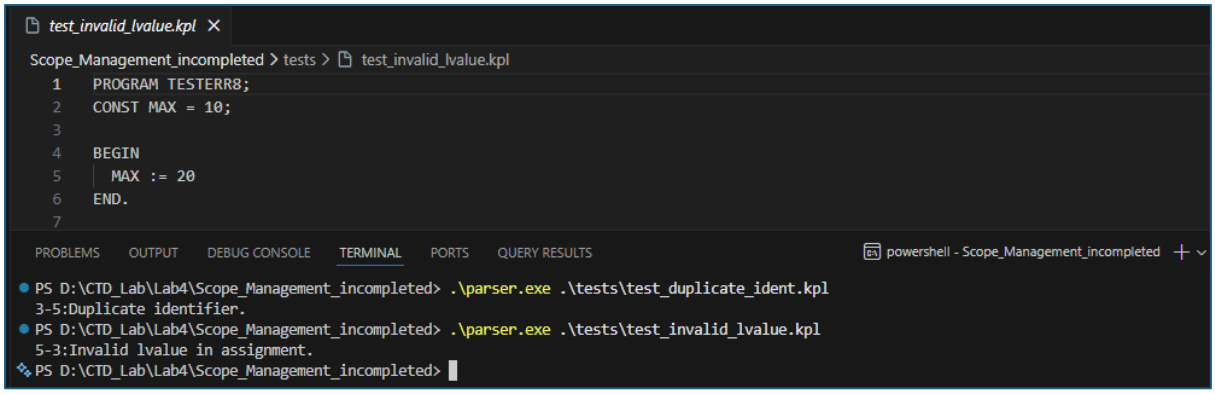
- **Nguyên nhân**: Lệnh CALL cố gắng gọi thủ tục MYPROC, nhưng MYPROC chưa được khai báo trong chương trình. Chỉ có biến X được khai báo.
- **Luồng xử lý**:
    - Parser gọi compileCallSt() khi gặp từ khóa CALL ở dòng 5
    - Gặp identifier MYPROC, gọi checkDeclaredProcedure("MYPROC")
    - checkDeclaredProcedure() gọi lookupObject("MYPROC")
    - lookupObject() tìm từ currentScope → outer → globalObjectList
    - Không tìm thấy MYPROC ở bất kỳ phạm vi nào, trả về NULL
    - checkDeclaredProcedure() phát hiện obj == NULL
    - Báo lỗi ERR_UNDECLARED_PROCEDURE tại dòng 5, cột 8
- **Hàm xử lý:** checkDeclaredProcedure() trong semantics.c, được gọi từ compileCallSt

8. Lỗi ERR_DUPLICATE_IDENT



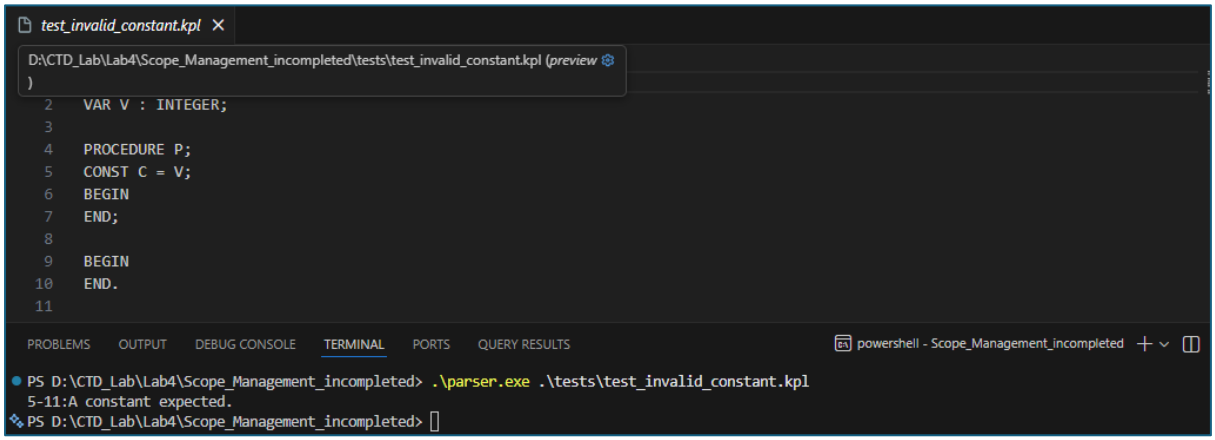
- **Dòng lỗi**: Dòng 3, cột 5 - identifier MAX
- **Nguyên nhân**: Identifier MAX được khai báo hai lần trong cùng phạm vi (program scope): lần đầu là hằng số (dòng 2), lần thứ hai là biến (dòng 3). Mỗi identifier chỉ được khai báo một lần trong cùng một phạm vi.
- **Luồng xử lý**:
    - **Dòng 2**: Parser xử lý CONST MAX = 10 trong compileBlock()
        - Gọi checkFreshIdent("MAX") để kiểm tra tên mới
        - findObject(currentScope->objList, "MAX") → Không tìm thấy → OK
        - Tạo constant object cho MAX
        - Gọi declareObject(constObj) → Thêm MAX vào currentScope->objList
    - **Dòng 3**: Parser xử lý VAR MAX : INTEGER trong compileBlock3()
        - Gọi checkFreshIdent("MAX") để kiểm tra tên mới
        - findObject(currentScope->objList, "MAX") → Tìm thấy MAX (đã khai báo ở dòng 2)
        - checkFreshIdent() phát hiện MAX đã tồn tại
        - Báo lỗi ERR_DUPLICATE_IDENT tại dòng 3, cột 5
- **Hàm xử lý:** checkFreshIdent() trong semantics.c

9. Lỗi ERR_INVALID_LVALUE



- **Dòng lỗi**: Dòng 5, cột 3 - identifier MAX
- **Nguyên nhân**: MAX là hằng số (CONST MAX = 10), không thể thay đổi giá trị. Chỉ có biến (variable), tham số (parameter), và tên hàm hiện tại (function) mới có thể xuất hiện ở vế trái của phép gán (lvalue).
- **Luồng xử lý**:
    - Parser xử lý phép gán MAX := 20 trong compileAssignSt() (dòng 5)
    - compileAssignSt() gọi compileLValue() để xử lý vế trái
    - compileLValue() gọi checkDeclaredLValueIdent("MAX")
    - checkDeclaredLValueIdent() gọi lookupObject("MAX") → Tìm thấy MAX
    - Kiểm tra obj->kind:
        - MAX có kind = OBJ_CONSTANT
        - Không phải OBJ_VARIABLE, OBJ_PARAMETER, hay OBJ_FUNCTION
    - Rơi vào case mặc định: không phải lvalue hợp lệ
    - Báo lỗi ERR_INVALID_LVALUE tại dòng 5, cột 3
- **Hàm xử lý:** checkDeclaredLValueIdent() trong semantics.c, kiểm tra 3 trường hợp hợp lệ

10. Lỗi ERR_INVALID_CONSTANT

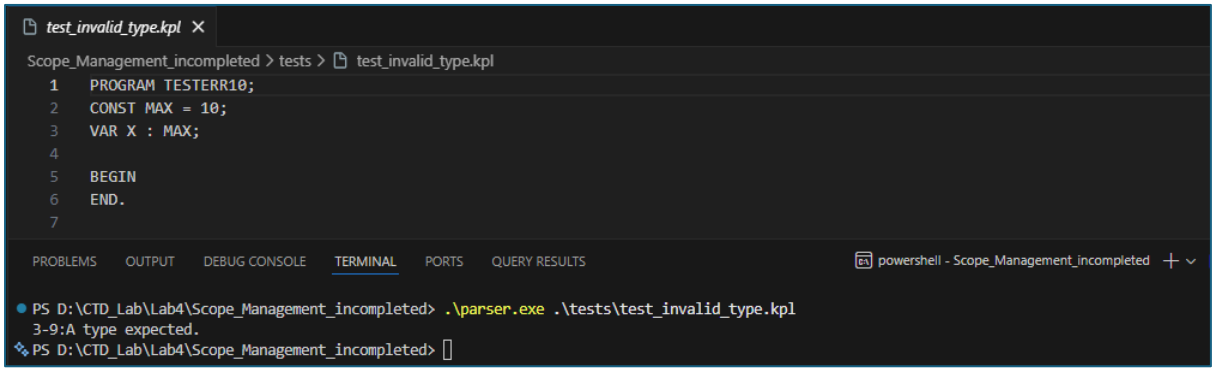


- **Dòng lỗi**: Dòng 5, cột 11 - identifier V
- **Nguyên nhân**: Khai báo CONST C = V, trong đó V là biến (VAR V : INTEGER), không phải hằng số. Giá trị của hằng phải là literal (số, ký tự) hoặc một hằng số khác

đã được định nghĩa trước. Không thể sử dụng biến để khởi tạo hằng vì giá trị biến có thể thay đổi trong runtime.

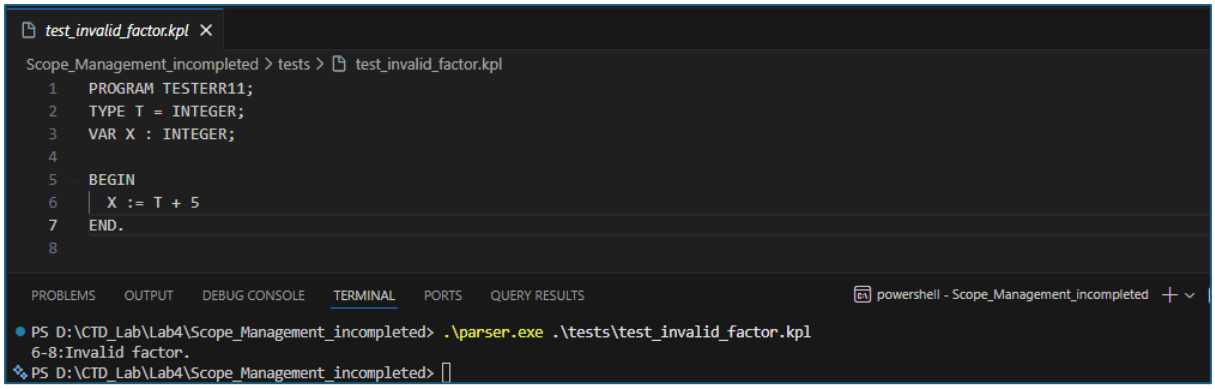
- **Luồng xử lý**:
  - Parser xử lý CONST C = V trong procedure P (dòng 5)
  - compileBlock() → compileConstant() khi gặp giá trị của hằng
  - compileConstant() → compileConstant2() khi gặp identifier V
  - compileConstant2() gọi checkDeclaredConstant("V")
  - checkDeclaredConstant() gọi lookupObject("V") → Tìm thấy V
  - Kiểm tra obj->kind:
    - V có kind = OBJ_VARIABLE
    - Không phải OBJ_CONSTANT
  - checkDeclaredConstant() phát hiện obj->kind != OBJ_CONSTANT
  - Báo lỗi ERR_INVALID_CONSTANT tại dòng 5, cột 11
- **Hàm xử lý:** checkDeclaredConstant() trong semantics.c, kiểm tra kind sau khi tìm thấy

11. Lỗi ERR_INVALID_TYPE



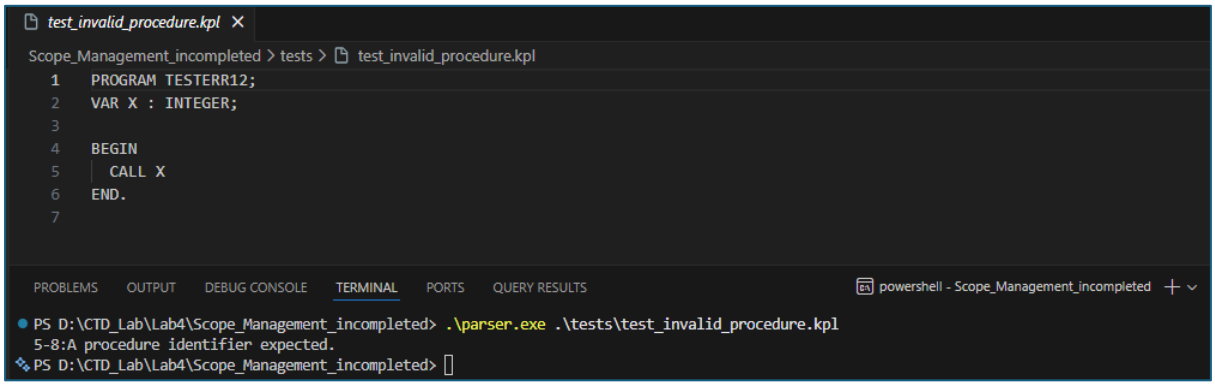
- **Dòng lỗi**: Dòng 3, cột 9 - identifier MAX
- **Nguyên nhân**: Khai báo VAR X : MAX, trong đó MAX là hằng số (CONST MAX = 10), không phải kiểu dữ liệu. Sau dấu : trong khai báo biến phải là một kiểu dữ liệu hợp lệ (INTEGER, CHAR, hoặc kiểu do người dùng định nghĩa bằng TYPE).
- **Luồng xử lý**:
  - Parser xử lý VAR X : MAX trong compileBlock3() (dòng 3)
  - Gặp dấu :, gọi compileType() để lấy kiểu của biến X
  - compileType() gặp identifier MAX, gọi checkDeclaredType("MAX")
  - checkDeclaredType() gọi lookupObject("MAX") → Tìm thấy MAX
  - Kiểm tra obj->kind:
    - MAX có kind = OBJ_CONSTANT
    - Không phải OBJ_TYPE
  - checkDeclaredType() phát hiện obj->kind != OBJ_TYPE
  - Báo lỗi ERR_INVALID_TYPE tại dòng 3, cột 9
- **Hàm xử lý:** checkDeclaredType() trong semantics.c

12. Lỗi ERR_INVALID_FACTOR



- **Dòng lỗi**: Dòng 6, cột 8 - identifier T
- **Nguyên nhân**: Biểu thức T + 5 sử dụng T (là tên kiểu TYPE T = INTEGER) như một toán hạng (factor). Tên kiểu không có giá trị, không thể xuất hiện trong biểu thức số học. Chỉ có constant, variable, parameter, và function call mới có thể là factor.
- **Luồng xử lý**:
    - Parser xử lý phép gán X := T + 5 trong compileAssignSt() (dòng 6)
    - Compile vế phải: compileExpression() → compileTerm() → compileFactor()
    - compileFactor() gặp identifier T, gọi checkDeclaredIdent("T")
    - checkDeclaredIdent() tìm thấy T với kind = OBJ_TYPE
    - Trong compileFactor(), kiểm tra obj->kind trong switch statement:
        - Case OBJ_CONSTANT, OBJ_VARIABLE, OBJ_PARAMETER, OBJ_FUNCTION → Hợp lệ
        - Case OBJ_TYPE → Không hợp lệ trong factor
    - Báo lỗi ERR_INVALID_FACTOR tại dòng 6, cột 8
- **Hàm xử lý:** compileFactor() trong parser.c, sau khi checkDeclaredIdent thành công

13. Lỗi ERR_INVALID_PROCEDURE



- **Dòng lỗi**: Dòng 5, cột 8 - identifier X
- **Nguyên nhân**: Lệnh CALL X cố gắng gọi X như một thủ tục, nhưng X là biến (VAR X : INTEGER), không phải thủ tục. Lệnh CALL chỉ có thể gọi procedure, không thể gọi variable, constant, type, hay function.
- **Luồng xử lý**:
    - Parser gọi compileCallSt() khi gặp từ khóa CALL (dòng 5)
    - Gặp identifier X, gọi checkDeclaredProcedure("X")

- checkDeclaredProcedure() gọi lookupObject("X") → Tìm thấy X
- Kiểm tra obj->kind:
  - X có kind = OBJ_VARIABLE
  - Không phải OBJ_PROCEDURE
- checkDeclaredProcedure() phát hiện obj->kind != OBJ_PROCEDURE
- Báo lỗi ERR_INVALID_PROCEDURE tại dòng 5, cột 8

- **Hàm xử lý:** checkDeclaredProcedure() trong semantics.c