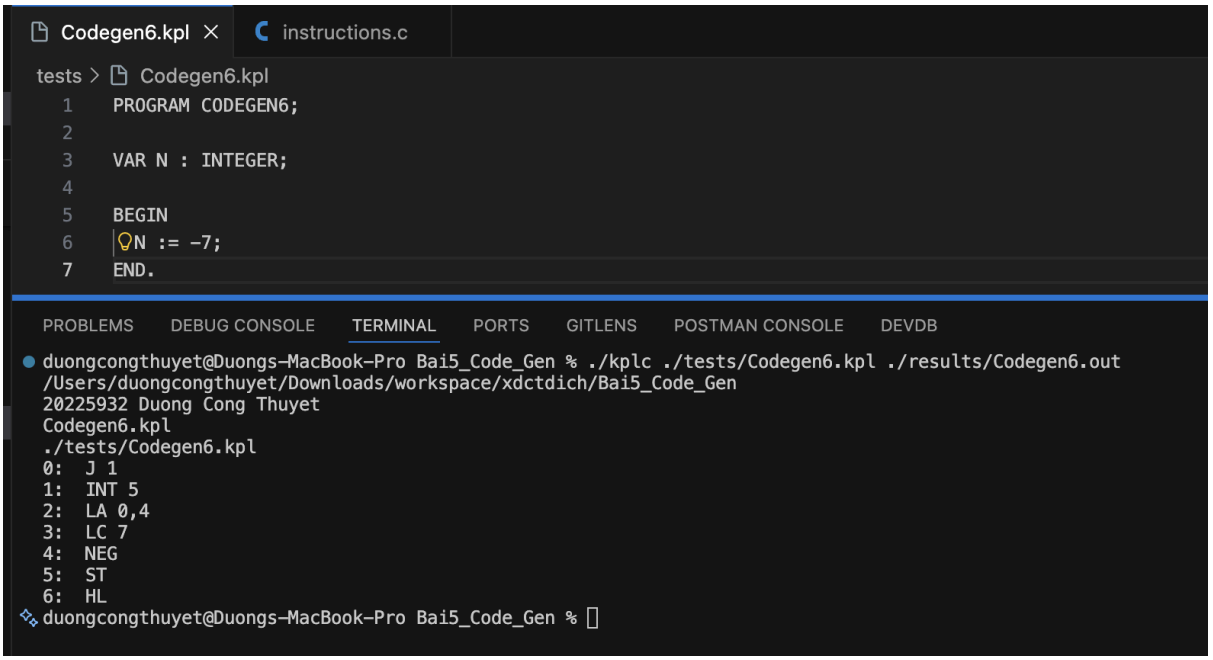
Bài làm lấy điểm giữa kỳ

Môn: Xây dựng chương trình dịch (phần thực hành)
Mã học phần: IT3323
Thời gian: 180 phút

Họ và tên: Dương Công Thuyết
MSSV: 20225932
Mã lớp BT/TH: 161269/161267
Ngành học: Công nghệ thông tin Việt – Nhật (IT-E6)

Câu 1: Thực hiện project CodeGen với file Codegen6.kpl đính kèm. Kết quả chạy được:



Cho biết đoạn mã nguồn nào sinh ra lệnh đích:

3: LC 7

4: NEG

Giải thích.

1. MÃ NGUỒN KPL

```
PROGRAM CODEGEN6;
VAR N : INTEGER;
BEGIN
  N := -7;     <-- Dòng sinh ra lệnh LC 7 và NEG
END.
```

2. MÃ ĐÍCH ĐƯỢC SINH RA

```
0:  J 1
1:  INT 5
2:  LA 0,4
3:  LC 7        <-- Load hằng số 7 lên stack
4:  NEG         <-- Đảo dấu giá trị trên đỉnh stack
5:  ST
6:  HL
```

3. PHÂN TÍCH LUỒNG SINH MÃ

Câu lệnh: N := -7;

Bước 1: Parser phân tích biểu thức -7

→ Phát hiện dấu trừ (unary minus) ở vị trí đầu biểu thức

→ File: parser.c, Hàm: compileExpression()

Bước 2: Xử lý hằng số 7

→ Parser nhận diện token TK_NUMBER với giá trị 7

→ File: parser.c, Hàm: compileFactor()

→ Gọi genLC(7) để sinh lệnh LC 7

Bước 3: Sinh lệnh đảo dấu

→ Parser gọi genNEG() để xử lý dấu trừ

→ File: codegen.c, Hàm: genNEG()

→ Sinh lệnh NEG

4. ĐOẠN MÃ NGUỒN CỤ THỂ

4.1. Xử lý dấu trừ

File: parser.c

Hàm: compileExpression()

```
case SB_MINUS:
   eat(SB_MINUS);
   type = compileExpression2();
   genNEG(); // gen ra lệnh NEG
   checkIntType(type);
   break;
```

Giải thích:

- Nhận diện dấu trừ (SB_MINUS) ở đầu biểu thức
- Compile biểu thức sau dấu trừ (số 7)
- Gọi genNEG() để sinh lệnh đảo dấu

4.2. Xử lý hằng số 7

File: parser.c

Hàm: compileFactor()

```
case TK_NUMBER:
   eat(TK_NUMBER);
   genLC(currentToken->value);   // sinh lệnh LC 7
   type = intType;
   break;
```

Giải thích:

- Nhận diện token NUMBER với giá trị 7
- Gọi genLC(7) để sinh lệnh load hằng số lên stack

4.3. Sinh lệnh NEG

File: codegen.c

Hàm: genNEG()

```
void genNEG(void)

{

 emitNEG(codeBlock);

}
```

Giải thích:

- Phát lệnh NEG vào code block để đảo dấu giá trị trên đỉnh stack

4.4. Phát lệnh NEG (instructions.c - dòng 65)

File: instructions.c

Hàm: emitNEG()

```
int emitNEG(CodeBlock *codeBlock) { return emitCode(codeBlock, OP_NEG, DC_VALUE,
DC_VALUE); }
```

Giải thích:

- Ghi lệnh NEG vào code block với opcode OP_NEG

5. LUỒNG THỰC THI CHI TIẾT

Câu lệnh: N := -7;

-->

1. compileAssignment() xử lý phép gán

-->

2. compileExpression() phát hiện SB_MINUS (dấu trừ)

-->

3. compileExpression2() → compileTerm() → compileFactor()

-->

4. compileFactor() nhận TK_NUMBER = 7

-->

5. genLC(7) → emitLC() → Sinh lệnh: LC 7

-->

6. Quay lại compileExpression(), gọi genNEG()

-->

7. genNEG() → emitNEG() → Sinh lệnh: NEG

-->

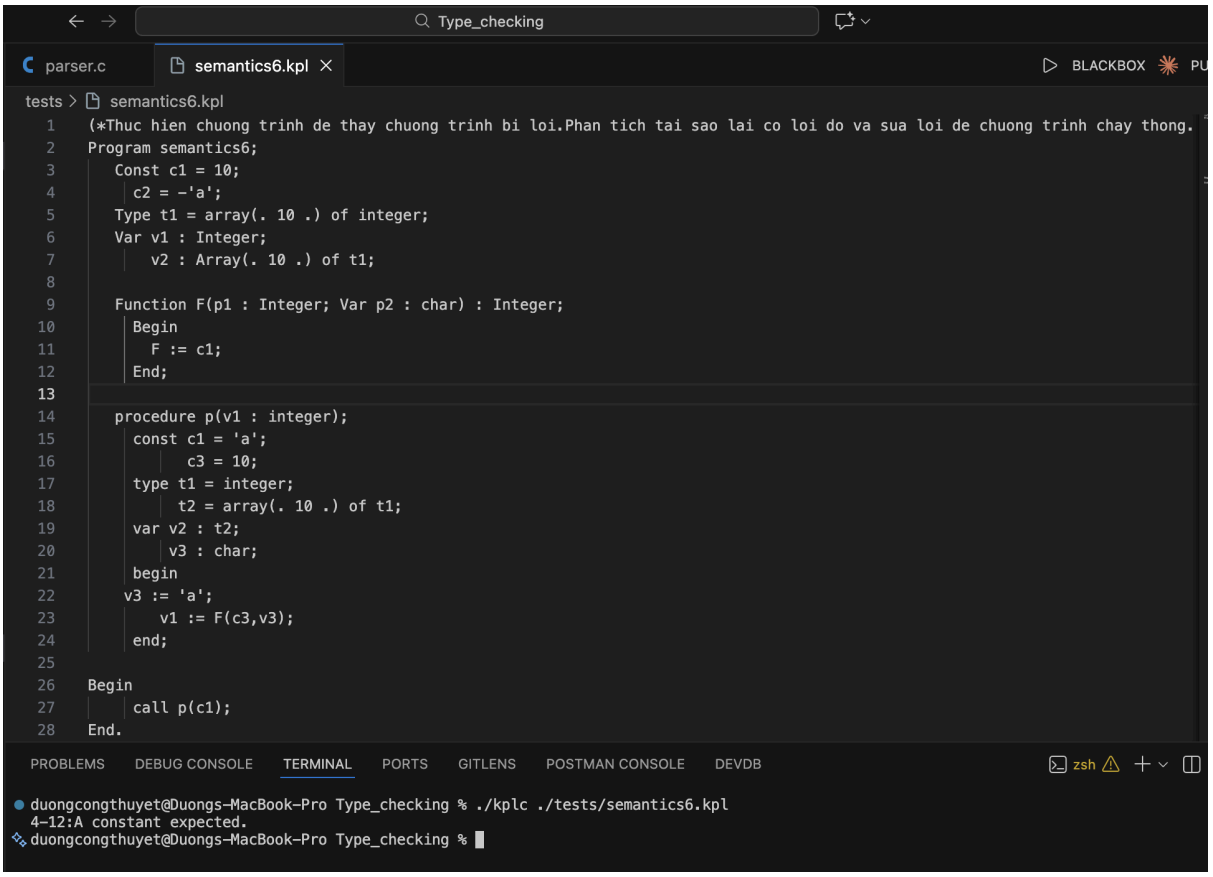
8. Stack: 7 → -7 (sau khi thực thi NEG)

-->

9. genST() sinh lệnh ST để lưu -7 vào biến N

Câu 2: Thực hiện bộ TypeChecking với file semantics6.kpl theo yêu cầu được ghi trong phần chú thích ở đầu file.

Màn hình kết quả chạy lỗi:



1. Lỗi nhận được: 4-12:A constant expected.
   Đây là lỗi cú pháp do vi phạm quy tắc khai báo hằng số trong ngôn ngữ KPL.
2. Đoạn code gây lỗi: c2 = -’a’;
3. Nguyên nhân gây lỗi:

Toán tử dấu trừ (-) chỉ có thể áp dụng cho hằng số kiểu INTEGER, không thể

áp dụng cho hằng số kiểu CHAR.

Trong khai báo: c2 = -'a'

- Giá trị 'a' có kiểu CHAR

- Toán tử "-" (âm) yêu cầu toán hạng phải là INTEGER

- Compiler phát hiện mâu thuẫn kiểu dữ liệu

4. Quy tắc kiểm tra kiểu:

Quy tắc: Trong khai báo hằng số, nếu có toán tử + hoặc - đứng trước giá trị,

thì giá trị đó PHẢI có kiểu INTEGER.

Cú pháp hợp lệ:

- Constant ::= ['+' | '-'] IntegerConstant

- Constant ::= CharConstant

Không hợp lệ:

- Constant ::= '-' CharConstant <-- Không hợp lệ

5. Hàm phát hiện lỗi:

Chuỗi gọi hàm: compileConstant() → compileConstant2() → error()

Hàm gọi lỗi: compileConstant2()

Đoạn code cụ thể:

```
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;

 default:

   error(ERR_INVALID_CONSTANT, lookAhead->lineNo, lookAhead->colNo); // Lỗi ở đây.

   break;

 }

 return constValue;

}
```

Giải thích:

- compileConstant2() chỉ chấp nhận TK_NUMBER (số) hoặc TK_IDENT (tên hằng)
- Khi gặp TK_CHAR ('a'), không khớp với bất kỳ case nào
- Rơi vào default case và gọi error(ERR_INVALID_CONSTANT, ...)
- Message lỗi: "A constant expected."

6. Luồng xử lý:

1: Parser gặp "c2 =" trong phần khai báo CONST

→ Gọi compileConstant()

2: compileConstant() phát hiện token SB_MINUS (dấu -)

→ Vào case SB_MINUS

→ eat(SB_MINUS) - ăn token dấu -

→ Gọi compileConstant2() để lấy giá trị sau dấu -

3: compileConstant2() đọc lookAhead token

→ Phát hiện TK_CHAR (ký tự 'a')

→ Kiểm tra switch cases:

- TK_NUMBER? KHÔNG
- TK_IDENT? KHÔNG
- Rơi vào default case

4: default case gọi error(ERR_INVALID_CONSTANT, ...)

→ Gọi đến hàm compileConstant2() trong parser.c

→ Báo lỗi: "A constant expected."

→ Dừng biên dịch

Lý do lỗi ở 3:

- compileConstant2() được thiết kế để xử lý:
  + Số nguyên (TK_NUMBER): như 10, -5, 100
  + Tên hằng (TK_IDENT): như c1, MAX
- KHÔNG xử lý ký tự (TK_CHAR): như 'a', 'b'
- Vì sau dấu - chỉ được phép là số hoặc tên hằng số nguyên

7. Cách khắc phục:

Bỏ dấu trừ, giữ nguyên kiểu CHAR

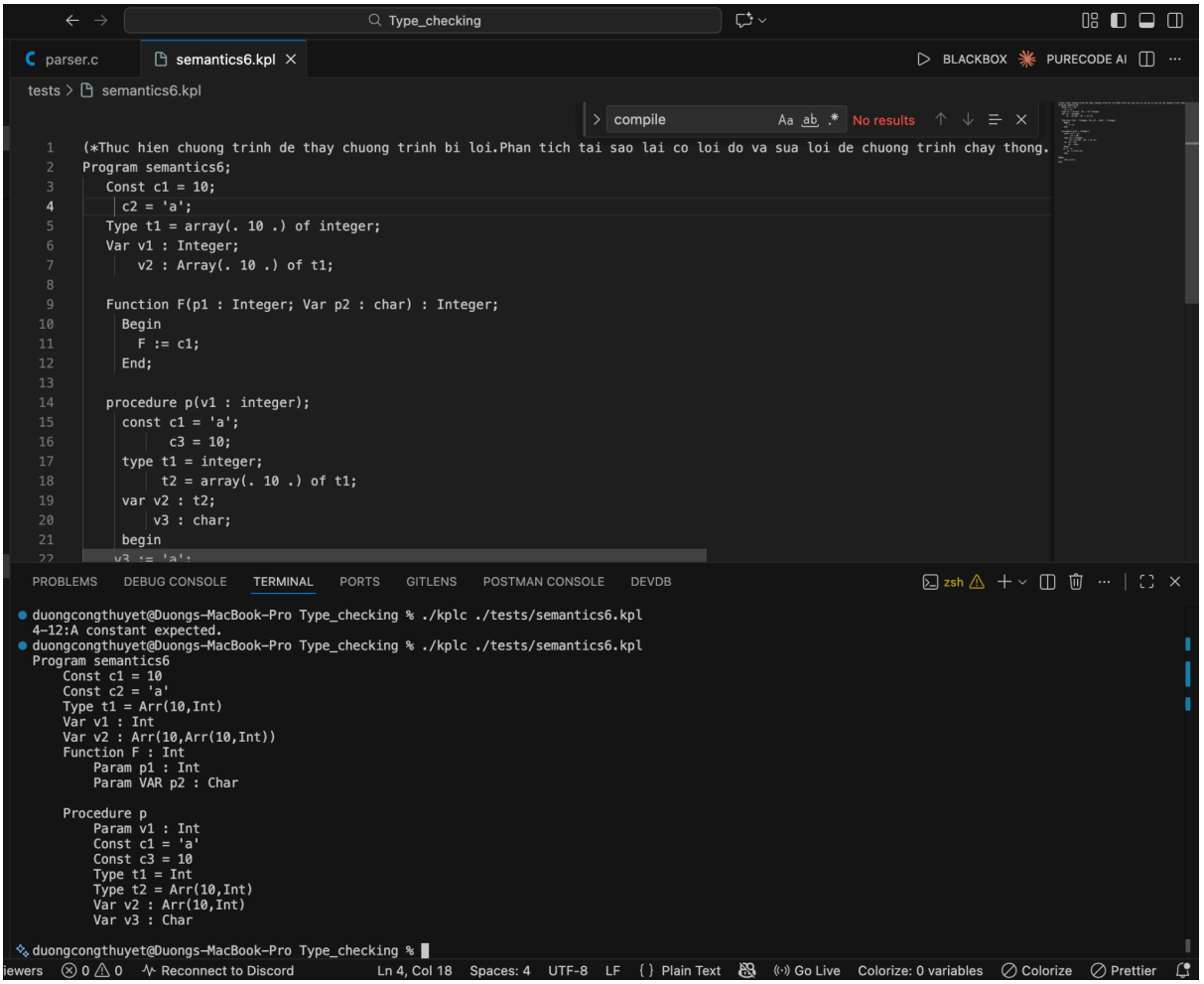
Code chỉnh sửa:

```
Const c1 = 10;
      c2 = 'a';
```

Giải thích:

- Khai báo c2 là hằng ký tự có giá trị 'a'
- Không sử dụng toán tử âm
- Phù hợp với cú pháp KPL cho hằng ký tự

Màn hình kết quả chạy thông:



<u>Câu 3:</u> Có thay đổi trong luật cú pháp trong KPL như sau:

<CallSt> ::= KW_CALL TK_IDENT <ActualParams>

<ActualParams>::= SB_LPAR <ActualParams1> SB_RPAR

<ActualParams1> ::= <Expression> <ActualParams2>

<ActualParams1> ::= e

<ActualParams2>::= SB_COMMA <Expression> <ActualParams2>

<ActualParams2>::= e

Thay đổi này có ý nghĩa như thế nào?

1. Nội dung thay đổi:

Cú pháp CŨ (Arguments - dấu ngoặc TÙY CHỌN):

<CallSt> ::= KW_CALL TK_IDENT

<CallSt> ::= KW_CALL TK_IDENT SB_LPAR <Arguments> SB_RPAR

<Arguments> ::= <Expression> <Arguments2>

<Arguments2> ::= SB_COMMA <Expression> <Arguments2>

<Arguments2> ::= e

Cú pháp MỚI (ActualParams - dấu ngoặc BẮT BUỘC):

<CallSt> ::= KW_CALL TK_IDENT <ActualParams>

<ActualParams> ::= SB_LPAR <ActualParams1> SB_RPAR

<ActualParams1> ::= <Expression> <ActualParams2>

<ActualParams1> ::= e

<ActualParams2> ::= SB_COMMA <Expression> <ActualParams2>

<ActualParams2> ::= e

2. Ý nghĩa thay đổi:

2.1.Thay đổi về cấu trúc:

CÚ PHÁP CŨ:

- Dấu ngoặc () là TÙY CHỌN

- Không có tham số: CALL proc (không có dấu ngoặc)

- Có tham số: CALL proc(x, y) (có dấu ngoặc)

- Khi có dấu (), BẮT BUỘC phải có ít nhất 1 tham số

CÚ PHÁP MỚI:

- Dấu ngoặc () là BẮT BUỘC
- Không có tham số: CALL proc() (có dấu ngoặc rỗng)
- Có tham số: CALL proc(x, y) (có dấu ngoặc)
- Dấu () rỗng được phép (ActualParams1 có production e)

2.2. Thay đổi về ngữ nghĩa:

CÚ PHÁP CŨ (code hiện tại):

```
CALL WriteLn;          // ĐÚNG - không có dấu ngoặc
CALL WriteI(n);        // ĐÚNG - có tham số trong ngoặc
CALL Proc();           // SAI - có () thì phải có ít nhất 1 tham số
```

CÚ PHÁP MỚI (yêu cầu thay đổi):

```
CALL WriteLn;          // SAI - thiếu dấu ngoặc ()
CALL WriteLn();        // ĐÚNG - dấu ngoặc bắt buộc, danh sách rỗng
CALL WriteI(n);        // ĐÚNG - có tham số
CALL Proc();           // ĐÚNG - () rỗng được phép
```

2.3.Lợi ích:

- Nhất quán: Mọi lời gọi procedure/function đều có dấu ()
- Rõ ràng: Dễ phân biệt lời gọi hàm với tên biến
- Linh hoạt: Hỗ trợ procedure/function không tham số với ()

Hãy thay đổi code của bộ parser để thực hiện sự thay đổi này. Thực hiện bộ parser với ví dụ parser6.kpl đính kèm. Phân tích những thay đổi khi chạy project mới.

Kết quả chạy

Code cũ:

Output:

Parsing a Program ....

1-1:KW_PROGRAM

1-10:TK_IDENT(PARSER6)

...

8-1:KW_PROCEDURE

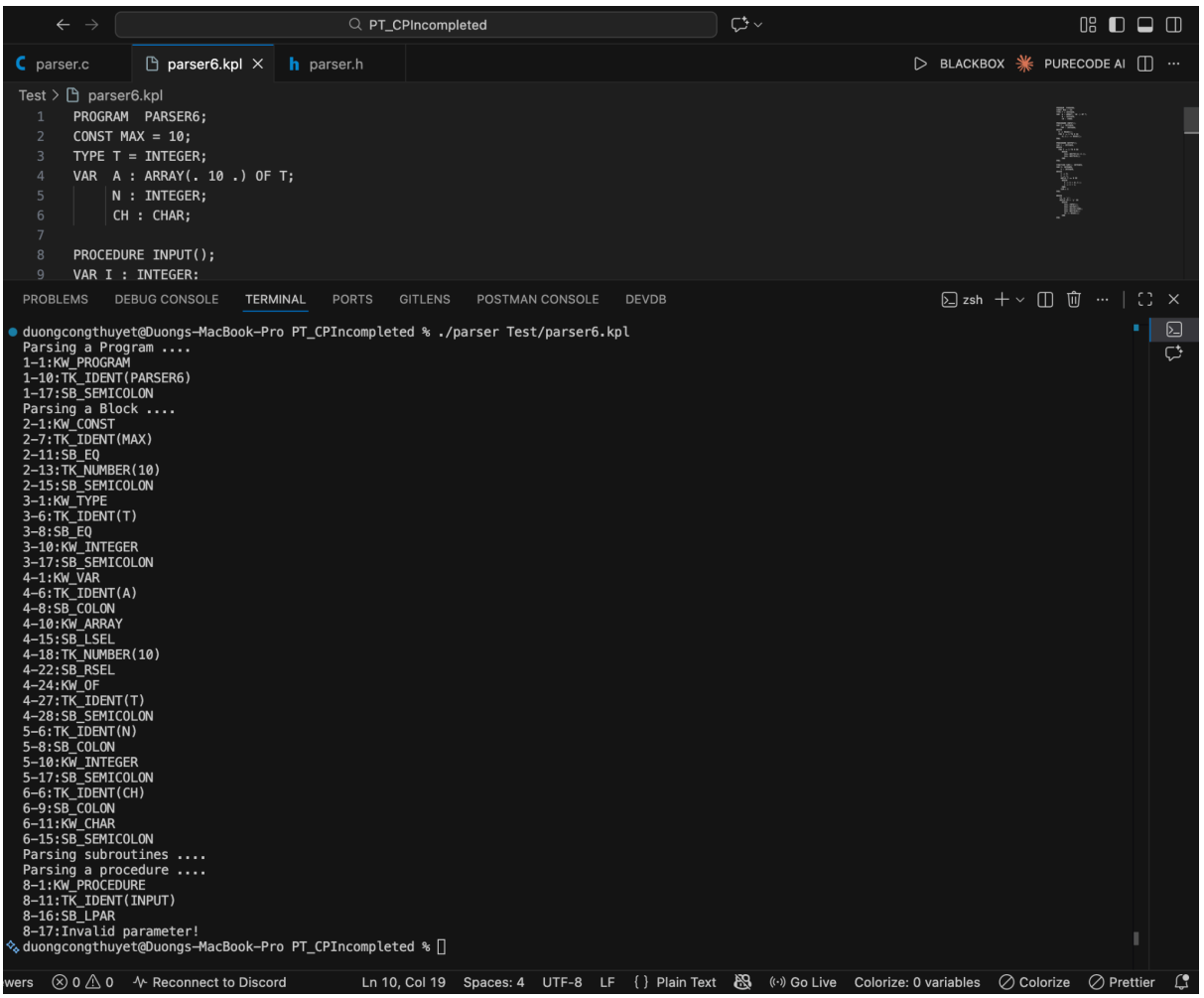
8-11:TK_IDENT(INPUT)

8-16:SB_LPAR

8-17:Invalid parameter!

Kết quả: FAIL tại dòng 8, cột 17

Lỗi: "Invalid parameter!"



Code mới:

Output:

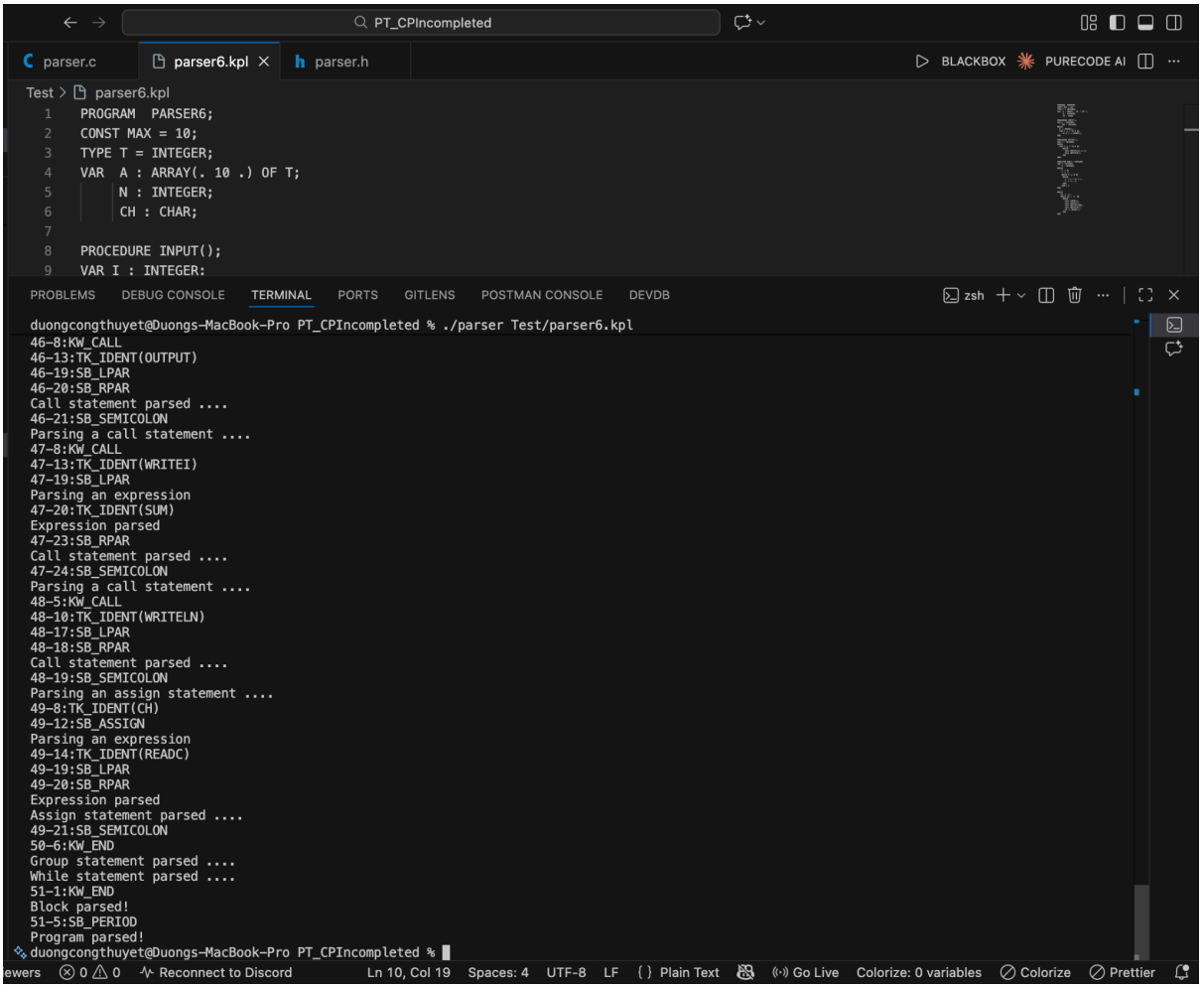
Parsing a Program ....

... (52 dòng parse thành công)

51-5:SB_PERIOD

Program parsed!

Kết quả: PASS - Parse hoàn toàn thành công!



Những thay đổi chính khi chạy:

1. PROCEDURE/FUNCTION không tham số:

Code cũ: PROCEDURE INPUT(); → Lỗi "Invalid parameter!"

Code mới: PROCEDURE INPUT(); → Parse thành công

2. Function call không tham số trong expression:

Code cũ: n := READI(); → Lỗi "Invalid factor!"

Code mới: n := READI(); → Parse thành công

3. CALL statement với () rỗng:

Code cũ: CALL INPUT(); → Lỗi "Invalid parameter!"

Code mới: CALL INPUT(); → Parse thành công

4. CALL statement không có ():

Code cũ: CALL WriteLn; → Parse thành công (dấu () tùy chọn)

Code mới: CALL WriteLn; → Lỗi "Missing '('" (dấu () bắt buộc)

Các case () rỗng trong file:

- Line 8: PROCEDURE INPUT();
- Line 12: n := READI();
- Line 14: A[I] := READI();
- Line 17: PROCEDURE OUTPUT();
- Line 23: CALL WRITELN();
- Line 27: FUNCTION SUM(): INTEGER;
- Line 45: CALL INPUT();
- Line 46: CALL OUTPUT();
- Line 48: CALL WRITELN();
- Line 49: CH := READC();

Code cũ: Fail tại case đầu tiên (line 8)

Code mới: Pass tất cả 10 cases + 42 dòng khác = 52 dòng

Copy nội dung những thay đổi vào bài làm. Ghi chú rõ là thay đổi ở hàm nào.

1: parser.h

Thêm 3 khai báo hàm mới

```
void compileActualParams(void);
void compileActualParams1(void);
void compileActualParams2(void);
```

2: parser.c - compileCallSt()

Sửa hàm compileCallSt()

Code cũ:

```
  void compileCallSt(void) {
    assert("Parsing a call statement ....");
    eat(KW_CALL);
    eat(TK_IDENT);
    if (lookAhead->tokenType == SB_LPAR) {
      eat(SB_LPAR);
      compileArguments();
      eat(SB_RPAR);
    }
    assert("Call statement parsed ....");
  }
```

Code mới:

```
  void compileCallSt(void) {
    assert("Parsing a call statement ....");
    eat(KW_CALL);
    eat(TK_IDENT);
    compileActualParams();  // Luôn gọi, không kiểm tra if
    assert("Call statement parsed ....");
  }
```

Giải thích:

- XÓA: if (lookAhead->tokenType == SB_LPAR) { ... }
- THÊM: compileActualParams(); // Bắt buộc có dấu ngoặc

3: parser.c - compileActualParams()

Thêm hàm mới:

```
  void compileActualParams(void) {
```

```
    eat(SB_LPAR);              // Bắt buộc có (
    compileActualParams1();    // Xử lý danh sách tham số
    eat(SB_RPAR);              // Bắt buộc có )
  }
```

Giải thích:

- Hàm này bắt buộc phải có dấu (và)
- Gọi compileActualParams1() để xử lý nội dung bên trong

4: parser.c - compileActualParams1()

Thêm hàm mới:

```
  void compileActualParams1(void) {
    switch (lookAhead->tokenType) {
    case TK_NUMBER:
    case TK_CHAR:
    case TK_IDENT:
    case SB_LPAR:
      // Có expression -> compile và gọi compileActualParams2
      compileExpression();
      compileActualParams2();
      break;
    case SB_RPAR:
      // Danh sách rỗng -> không làm gì (production e)
      break;
    default:
      error(ERR_INVALIDARGUMENTS, lookAhead->lineNo, lookAhead->colNo);
      break;
    }
  }
```

Giải thích:

- Nếu gặp token bắt đầu expression -> compile expression list
- Nếu gặp) -> danh sách rỗng, không làm gì (cho phép CALL proc())
- Token khác -> báo lỗi ERR_INVALIDARGUMENTS

5: parser.c - compileActualParams2()

Thêm hàm mới:

```
void compileActualParams2(void) {
  if (lookAhead->tokenType == SB_COMMA) {
    eat(SB_COMMA);
    if (lookAhead->tokenType == SB_RPAR) {
      error(ERR_INVALIDARGUMENTS, lookAhead->lineNo, lookAhead->colNo);
    }
    compileExpression();
    compileActualParams2();
  }
}
```

Giải thích:

- Nếu gặp dấu , -> đọc thêm expression và đệ quy
- Nếu sau dấu , là) -> lỗi (không cho phép trailing comma)
- Không có , -> kết thúc danh sách

6: parser.c - compileParams()

Sửa hàm compileParams():

Code cũ:

```
void compileParams(void) {
  compileParam();
  compileParams2();
```

```
}
```

Code mới:

```
void compileParams(void) {
  if (lookAhead->tokenType == SB_RPAR) {
    // Danh sách tham số rỗng trong khai báo procedure/function
    return;
  }
  compileParam();
  compileParams2();
}
```

Giải thích:

- THÊM: Kiểm tra SB_RPAR để chấp nhận procedure/function không tham số
- Ví dụ: PROCEDURE INPUT(); hoặc FUNCTION READI(): INTEGER;

7: parser.c - compileArguments()

Sửa hàm compileArguments()

Code cũ:

```
void compileArguments(void) {
  compileExpression();
  compileArguments2();
}
```

Code mới:

```
void compileArguments(void) {
  if (lookAhead->tokenType == SB_RPAR) {
    // Danh sách arguments rỗng trong function call
    return;
```

```
    }
    compileExpression();
    compileArguments2();
  }
```

Giải thích:

- THÊM: Kiểm tra SB_RPAR để chấp nhận function call không tham số
- Ví dụ: n := READI(); (function call trong expression)