Báo cáo LAB05
IT3323 Mã lớp 161269
Bài 5 : Gen Code

Họ và tên: Dương Công Thuyết

MSSV: 20225932

I. Nội dung chính

1. Mục tiêu

Sinh mã bytecode cho máy ảo dựa trên ngăn xếp (stack-based VM) từ chương trình nguồn KPL đã qua phân tích cú pháp và ngữ nghĩa.

2. Kiến trúc mục tiêu

Máy ảo P-code với 32 lệnh cơ bản

Quản lý bộ nhớ: Ngăn xếp với frame pointer (static link, dynamic link)

Địa chỉ hóa: Cặp (level, offset) cho truy xuất biến qua các scope lồng nhau

3. Các thành phần đã triển khai

a) Quản lý biến (codegen.c)

genVariableAddress(): Sinh mã tải địa chỉ biến (LA)

genVariableValue(): Sinh mã tải giá trị biến (LV)

getNestingLevel(): Tính độ sâu lồng scope cho static linking

b) Biểu thức & Toán tử (parser.c)

Sinh mã cho các phép toán: +, -, *, / (AD, SB, ML, DV)

Sinh mã cho so sánh: =, !=, <, <=, >, >= (EQ, NE, LT, LE, GT, GE)

Xử lý hằng số (LC), biến, và mảng (tính offset)

c) Cấu trúc điều khiển

IF-THEN-ELSE: Sử dụng FJ (False Jump) và J (Jump)

WHILE-DO: Vòng lặp với điều kiện đầu

FOR-TO-DO: Vòng lặp với bộ đếm tự động tăng

d) Chương trình con

PROCEDURE: Gọi với CALL, thoát với EP

FUNCTION: Gọi với CALL, thoát với EF, trả giá trị qua offset 0

Xử lý tham số: truyền giá trị (value) và tham chiếu (reference)

Hỗ trợ đệ quy với static scoping

e) Mảng

Tính địa chỉ phần tử: base_address + index × element_size

Load gián tiếp (LI) để truy xuất giá trị

4. Kỹ thuật chính

Static Linking: Dùng level để truy xuất biến ở outer scope

Frame Management: Mỗi procedure/function có frame riêng với 4 word dành riêng

Backpatching: Cập nhật địa chỉ nhảy (updateJ, updateFJ) sau khi biên dịch khối lệnh

5. Kết quả

Compiler sinh bytecode chính xác cho đầy đủ các tính năng của ngôn ngữ KPL, đã test thành công với kết quả ở dưới.

II. Cases đã implement

CASE 1: BIẾN CỤC BỘ VÀ TOÀN CỤC (VARIABLES)

Hàm xử lý: genVariableAddress(), genVariableValue()

Code:

```
void genVariableAddress(Object *var) {
  int level, offset;

  if (var->kind == OBJ_VARIABLE) {
    // Tính level = khoảng cách từ scope hiện tại đến scope khai báo
    level = computeNestedLevel(symtab->currentScope, VARIABLE_SCOPE(var));
    offset = VARIABLE_OFFSET(var);
    genLA(level, offset);  // Load Address
  }
}
```

```
void genVariableValue(Object *var) {
  int level, offset;

  if (var->kind == OBJ_VARIABLE) {
    level = computeNestedLevel(symtab->currentScope, VARIABLE_SCOPE(var));
    offset = VARIABLE_OFFSET(var);
    genLV(level, offset);  // Load Value
  }
}
```

Ví dụ bytecode:

```
VAR x : INTEGER;        // offset = 4, level = 0
BEGIN
  x := 5;
END.
```

→ Bytecode:

```
  LA  0  4    // Lấy địa chỉ x
  LC  0  5    // Hằng số 5
  ST          // Gán x = 5
```

CASE 2: THAM SỐ TRUYỀN THEO GIÁ TRỊ VÀ THAM CHIẾU (PARAMETERS)

Hàm xử lý: genParameterAddress(), genParameterValue()

Code:

```
void genVariableAddress(Object *var) {
```

```
  // ...
  else { // OBJ_PARAMETER
    level = computeNestedLevel(symtab->currentScope, PARAMETER_SCOPE(var));
    offset = PARAMETER_OFFSET(var);

    if (var->paramAttrs->kind == PARAM_REFERENCE) {
      // Tham chiếu: địa chỉ đã được lưu, chỉ cần load ra
      genLV(level, offset);
    } else {
      // Giá trị: giống biến thường
      genLA(level, offset);
    }
  }
}

void genVariableValue(Object *var) {
  // ...
  else { // OBJ_PARAMETER
    level = computeNestedLevel(symtab->currentScope, PARAMETER_SCOPE(var));
    offset = PARAMETER_OFFSET(var);

    if (var->paramAttrs->kind == PARAM_REFERENCE) {
      // Tham chiếu: load địa chỉ + dereference
      genLV(level, offset);
      genLI();  // Load Indirect
    } else {
      // Giá trị: giống biến thường
      genLV(level, offset);
```

```
   }
  }
}
```

Ví dụ bytecode:

```
PROCEDURE Swap(VAR a, b : INTEGER);  // a, b là tham chiếu
VAR temp : INTEGER;
BEGIN
  temp := a;     // Đọc giá trị a qua tham chiếu
  a := b;        // Gán qua tham chiếu
END;
```

→ Bytecode trong Swap:

```
  LA  0  6       // Địa chỉ temp (offset 6)
  LV  0  4       // Load địa chỉ của a (tham chiếu ở offset 4)
  LI             // Dereference để lấy giá trị *a
  ST             // temp = a

  LV  0  4       // Load địa chỉ của a
  LI             // Truy xuất vị trí được trỏ
  LV  0  5       // Load địa chỉ của b
  LI             // Dereference lấy giá trị *b
  ST             // *a = b
```

CASE 3: MẢNG VÀ TRUY XUẤT PHẦN TỬ (ARRAYS)

Hàm xử lý: compileIndexes()

Code:

```
Type* compileIndexes(Type* arrayType) {
  Type *type = arrayType;

  do {
    eat(SB_LSEL);  // '['

    compileExpression();  // Tính giá trị index

    // Nhân với kích thước phần tử
    genLC(sizeOfType(type->elementType));
    genML();  // Multiply

    // Cộng vào địa chỉ base
    genAD();  // Add

    type = type->elementType;
    eat(SB_RSEL);  // ']'
  } while (lookAhead->tokenType == SB_LSEL);

  return type;
}
```

Ví dụ bytecode:

```
VAR arr : ARRAY [10] OF INTEGER;  // offset = 4
```

```
BEGIN
  arr[3] := 100;
END.
```

→ Bytecode:

```
  LA  0  4       // Địa chỉ base của arr
  LC  0  3       // Index = 3
  LC  0  4       // Kích thước INTEGER = 4 bytes
  ML             // 3 × 4 = 12
  AD             // base + 12 = địa chỉ arr[3]
  LC  0  100     // Hằng số 100
  ST             // arr[3] = 100
```

CASE 4: BIỂU THỨC SỐ HỌC (EXPRESSIONS)

Các toán tử: +, -, *, /

Hàm xử lý: compileExpression3(), compileTerm2()

Code:

```
Type* compileExpression3(void) {
  Type *type = compileTerm();

  while (lookAhead->tokenType == SB_PLUS ||
         lookAhead->tokenType == SB_MINUS) {
    TokenType op = lookAhead->tokenType;
    eat(op);
```

```
    compileTerm();  // Toán hạng phải

    if (op == SB_PLUS)
      genAD();  // Add
    else
      genSB();  // Subtract
  }
  return type;
}

Type* compileTerm2(void) {
  Type *type = compileFactor();

  while (lookAhead->tokenType == SB_TIMES ||
         lookAhead->tokenType == SB_SLASH) {
    TokenType op = lookAhead->tokenType;
    eat(op);

    compileFactor();  // Toán hạng phải

    if (op == SB_TIMES)
      genML();  // Multiply
    else
      genDV();  // Divide
  }
  return type;
}
```

Ví dụ bytecode:

result := (a + b) * c - d;

→ Bytecode:

```
LA  0  4      // Địa chỉ result
LV  0  5      // Giá trị a
LV  0  6      // Giá trị b
AD            // a + b
LV  0  7      // Giá trị c
ML            // (a + b) * c
LV  0  8      // Giá trị d
SB            // (a + b) * c - d
ST            // result = ...
```

CASE 5: ĐIỀU KIỆN SO SÁNH (CONDITIONS)

Các toán tử: =, <>, <, >, <=, >=

Hàm xử lý: compileCondition()

Code:

```
Type* compileCondition(void) {
  Type *type1 = compileExpression();

  TokenType op = lookAhead->tokenType;
  eat(op);
```

```
  Type *type2 = compileExpression();

  switch(op) {
    case SB_EQ: genEQ(); break;  // Equal
    case SB_NEQ: genNE(); break; // Not Equal
    case SB_LT: genLT(); break;  // Less Than
    case SB_GT: genGT(); break;  // Greater Than
    case SB_LE: genLE(); break;  // Less or Equal
    case SB_GE: genGE(); break;  // Greater or Equal
  }

  return type1;
}
```

Ví dụ bytecode:

IF x > y THEN ...

→ Bytecode:

```
  LV  0  4       // Giá trị x
  LV  0  5       // Giá trị y
  GT             // x > y (kết quả 0 hoặc 1 trên stack)
  FJ  addr       // False Jump nếu điều kiện = 0
```

CASE 6: CÂU LỆNH IF-THEN-ELSE

Hàm xử lý: compileIfSt()

Code:

```
void compileIfSt(void) {
  eat(KW_IF);
  compileCondition();

  int falseLabel = genFJ(DC_VALUE);  // Jump nếu điều kiện False

  eat(KW_THEN);
  compileStatement();

  if (lookAhead->tokenType == KW_ELSE) {
    int exitLabel = genJ(DC_VALUE);  // Jump qua ELSE block

    updateFJ(falseLabel, getCurrentCodeAddress());  // Backpatch

    eat(KW_ELSE);
    compileStatement();

    updateJ(exitLabel, getCurrentCodeAddress());
  } else {
    updateFJ(falseLabel, getCurrentCodeAddress());
  }
}
```

Ví dụ bytecode:

```
IF x > 5 THEN
```

```
  y := 1
ELSE
  y := 2;
```

→ Bytecode:

```
  0: LV  0  4       // x
  1: LC  0  5       // 5
  2: GT             // x > 5
  3: FJ  7          // Nếu false, nhảy đến địa chỉ 7 (ELSE)
  4: LA  0  5       // Địa chỉ y
  5: LC  0  1       // 1
  6: ST             // y = 1
  7: J   10         // Nhảy qua ELSE
  8: LA  0  5       // (ELSE) Địa chỉ y
  9: LC  0  2       // 2
 10: ST             // y = 2
 11: ...            // Tiếp tục
```

CASE 7: VÒNG LẶP WHILE

Hàm xử lý: compileWhileSt()

Code:

```
void compileWhileSt(void) {
  eat(KW_WHILE);

  int startLabel = getCurrentCodeAddress();  // Vị trí bắt đầu
```

```
  compileCondition();

  int falseLabel = genFJ(DC_VALUE);  // Jump ra khỏi vòng lặp nếu false

  eat(KW_DO);
  compileStatement();

  genJ(startLabel);  // Jump về đầu vòng lặp

  updateFJ(falseLabel, getCurrentCodeAddress());  // Backpatch
}
```

Ví dụ bytecode:

```
WHILE i < 10 DO BEGIN
  i := i + 1;
END;
```

→ Bytecode:

```
  0: LV  0  4       // (Start) Giá trị i
  1: LC  0  10      // 10
  2: LT             // i < 10
  3: FJ  9          // False thì thoát vòng lặp (nhảy đến 9)
  4: LA  0  4       // Địa chỉ i
  5: LV  0  4       // Giá trị i
  6: LC  0  1       // 1
  7: AD             // i + 1
```

```
 8: ST          // i = i + 1
 9: J  0        // Quay lại đầu vòng lặp (địa chỉ 0)
10: ...         // Tiếp tục sau vòng lặp
```

CASE 8: VÒNG LẶP FOR

Hàm xử lý: compileForSt()

Code:

```
void compileForSt(void) {
  eat(KW_FOR);

  Object *var = lookAhead->value;
  eat(TK_IDENT);
  eat(SB_ASSIGN);

  compileExpression();  // Giá trị khởi tạo
  genST();  // Gán vào biến đếm

  eat(KW_TO);

  int startLabel = getCurrentCodeAddress();

  genVariableValue(var);  // Giá trị hiện tại của counter
  compileExpression();    // Giá trị cuối
  genLE();  // counter <= end
```

```
  int falseLabel = genFJ(DC_VALUE);

  eat(KW_DO);
  compileStatement();

  // Tự động tăng counter
  genVariableAddress(var);
  genVariableValue(var);
  genLC(1);
  genAD();
  genST();

  genJ(startLabel);
  updateFJ(falseLabel, getCurrentCodeAddress());
}
```

Ví dụ bytecode:

```
FOR i := 1 TO 5 DO
  sum := sum + i;
```

→ Bytecode:

```
  0: LA  0  4       // Địa chỉ i
  1: LC  0  1       // 1
  2: ST             // i = 1 (khởi tạo)
  3: LV  0  4       // (Start loop) Giá trị i
  4: LC  0  5       // 5
  5: LE             // i <= 5
```

```
 6: FJ  15        // False thì thoát (nhảy đến 15)
 7: LA  0  5      // Địa chỉ sum
 8: LV  0  5      // Giá trị sum
 9: LV  0  4      // Giá trị i
10: AD            // sum + i
11: ST            // sum = sum + i
12: LA  0  4      // Địa chỉ i
13: LV  0  4      // Giá trị i
14: LC  0  1      // 1
15: AD            // i + 1
16: ST            // i = i + 1
17: J   3         // Quay lại kiểm tra điều kiện (địa chỉ 3)
18: ...           // Tiếp tục sau vòng lặp
```

CASE 9: GÁN GIÁ TRỊ (ASSIGNMENT)

Hàm xử lý: compileAssignSt() + compileLValue()

Code:

```
void compileAssignSt(void) {
  Object *obj = lookAhead->value;
  eat(TK_IDENT);

  Type *varType;

  if (lookAhead->tokenType == SB_LSEL) {
    // Mảng: arr[index]
```

```
    checkArrayType(obj->varAttrs->type);
    genVariableAddress(obj);  // Địa chỉ base
    varType = compileIndexes(obj->varAttrs->type);
  } else {
    // Biến đơn
    genVariableAddress(obj);
    varType = obj->varAttrs->type;
  }

  eat(SB_ASSIGN);
  Type *expType = compileExpression();  // Tính giá trị phải

  checkTypeEquality(varType, expType);

  genST();  // Store giá trị vào địa chỉ
}
```

Ví dụ bytecode:

```
x := a + b * 2;
```

→ Bytecode:

```
  LA  0  4       // Địa chỉ x
  LV  0  5       // Giá trị a
  LV  0  6       // Giá trị b
  LC  0  2       // 2
  ML             // b * 2
  AD             // a + (b * 2)
```

```
  ST              // x = a + b * 2
```

CASE 10: GỌI THỦ TỤC NGƯỜI DÙNG (PROCEDURE CALL)

Hàm xử lý: compileCallSt()

Code:

```
void compileCallSt(void) {
  Object *proc = lookAhead->value;
  eat(TK_IDENT);

  compileArguments(proc->procAttrs->paramList);  // Đẩy tham số lên stack

  int level = computeNestedLevel(symtab->currentScope,
                                 proc->procAttrs->scope->outer);
  genCALL(level, proc->procAttrs->codeAddress);

  if (proc->procAttrs->paramList != NULL) {
    int nParams = countParams(proc->procAttrs->paramList);
    genDCT(nParams);  // Deallocate parameters
  }
}
```

Ví dụ bytecode:

```
PROCEDURE PrintSum(a, b : INTEGER);
BEGIN
```

```
  WRITEI(a + b);
END;

BEGIN
  PrintSum(5, 3);
END.
```

→ Bytecode tại điểm gọi:

```
  LC  0  5        // Tham số 1: 5
  LC  0  3        // Tham số 2: 3
  CALL 0 addr     // Gọi PrintSum (level=0, địa chỉ procedure)
  DCT 2           // Giải phóng 2 tham số
```

CASE 11: GỌI HÀM VÀ TRẢ VỀ GIÁ TRỊ (FUNCTION CALL)

Hàm xử lý: compileFuncDecl() + compileFactor()

Code:

```
// Trong compileFactor() khi gặp function call
case OBJ_FUNCTION:
  compileArguments(obj->funcAttrs->paramList);

  int level = computeNestedLevel(symtab->currentScope,
                                 obj->funcAttrs->scope->outer);
  genCALL(level, obj->funcAttrs->codeAddress);

  if (obj->funcAttrs->paramList != NULL) {
```

```
    int nParams = countParams(obj->funcAttrs->paramList);
    genDCT(nParams);
  }

  genLV(0, 0);  // Load giá trị trả về (offset 0 trong frame)
  break;

// Trong compileFuncDecl() - kết thúc hàm
void compileFuncDecl(void) {
  // ... khai báo và body ...
  genEF();  // Exit Function
}
```

Ví dụ bytecode:

```
FUNCTION Add(a, b : INTEGER) : INTEGER;
BEGIN
  Add := a + b;  // Gán vào vị trí return value
END;

VAR result : INTEGER;
BEGIN
  result := Add(3, 4);
END.
```

→ Bytecode trong hàm Add:

```
  PROC label       // Entry point
  LA  0  0         // Địa chỉ return value (offset 0)
```

```
LV  0  4        // Tham số a
LV  0  5        // Tham số b
AD              // a + b
ST              // Gán vào return value
EF              // Exit function
```

→ Bytecode tại điểm gọi:

```
LA  0  4        // Địa chỉ result
LC  0  3        // Tham số 1: 3
LC  0  4        // Tham số 2: 4
CALL 0 label    // Gọi Add
DCT 2           // Giải phóng 2 tham số
LV  0  0        // Load giá trị trả về
ST              // result = Add(3, 4)
```

CASE 12: ĐỆ QUY VÀ NESTED SCOPE

Hàm xử lý: computeNestedLevel()

Code:

```
int computeNestedLevel(Scope *currentScope, Scope *targetScope) {
  int level = 0;
  Scope *scope = currentScope;

  // Duyệt ngược scope chain đến khi gặp target scope
  while (scope != NULL && scope != targetScope) {
    scope = scope->outer;
    level++;
  }
```

```
  return level;  // Khoảng cách giữa 2 scope
}
```

Ví dụ bytecode:

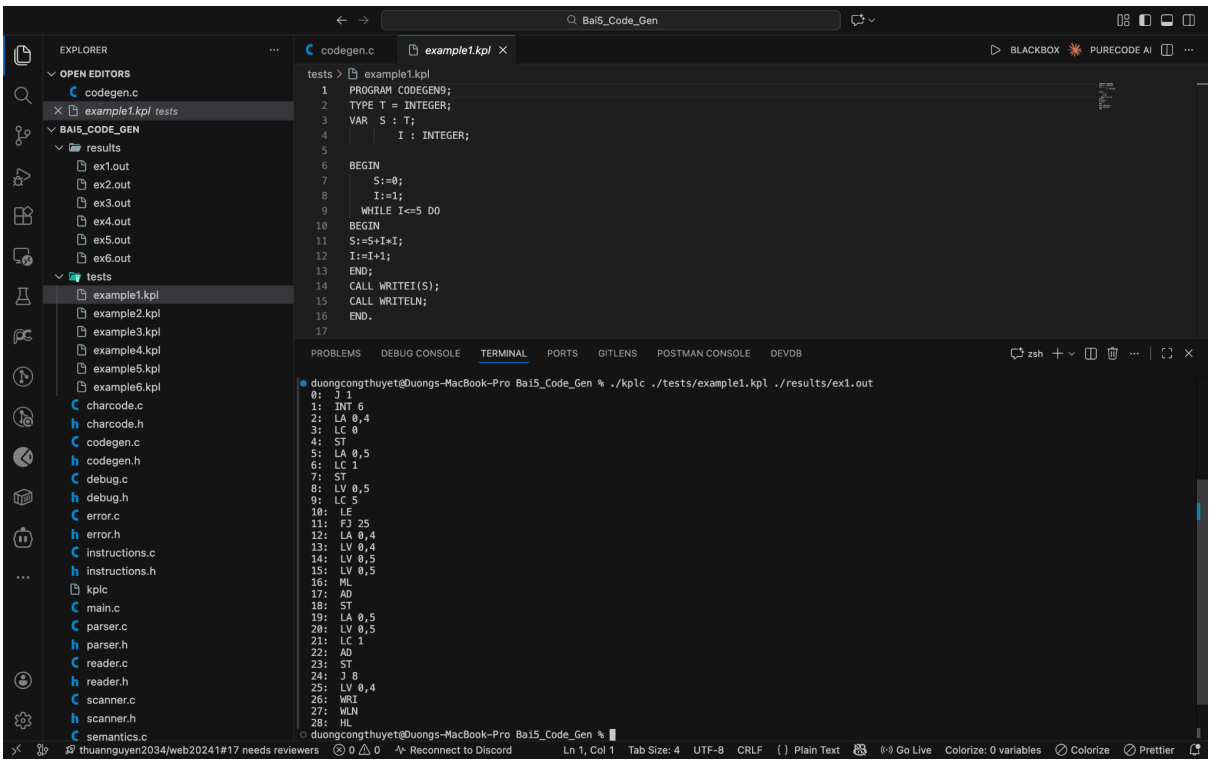
```
VAR global : INTEGER;

PROCEDURE Outer;
  VAR outer_var : INTEGER;

  PROCEDURE Inner;
  BEGIN
    global := 1;      // level = 2 (cách 2 scope)
    outer_var := 2;   // level = 1 (cách 1 scope)
  END;
BEGIN
  Inner;
END;
```

→ Bytecode trong Inner:

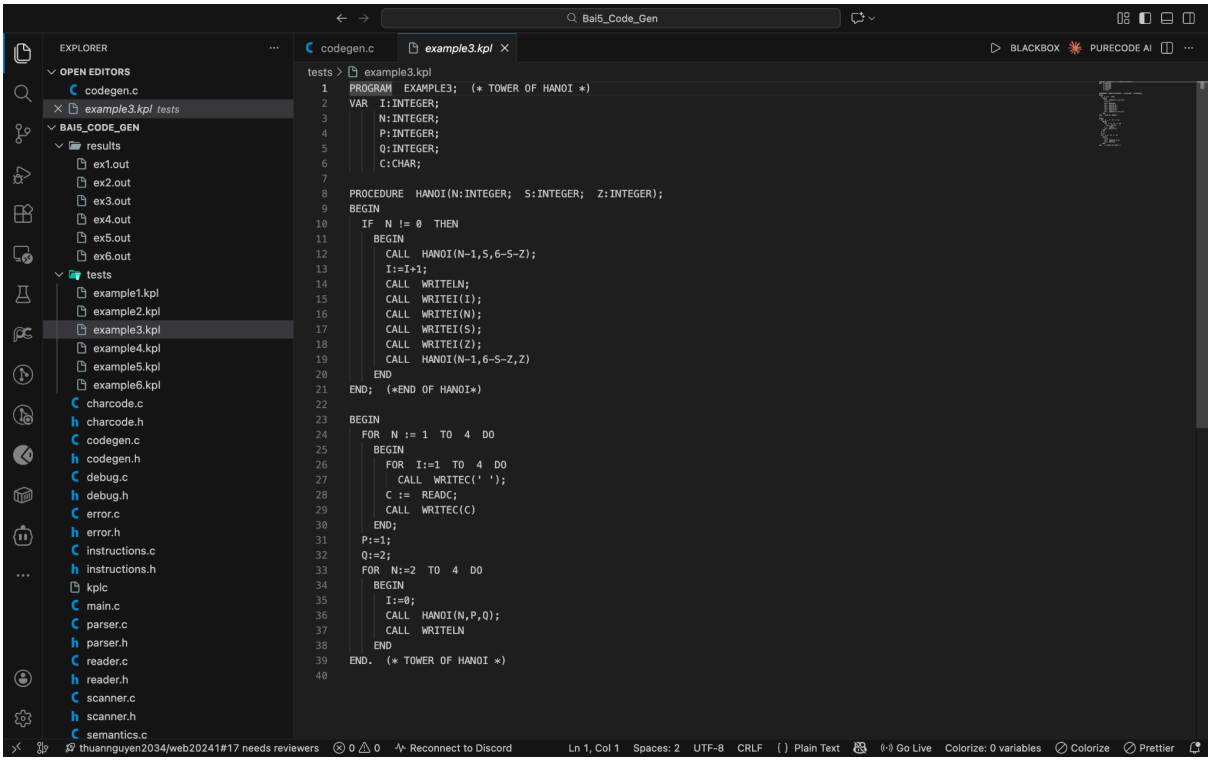
```
  LA  2  4        // global: level=2 (global scope)
  LC  0  1
  ST

  LA  1  4        // outer_var: level=1 (Outer's scope)
  LC  0  2
  ST
```

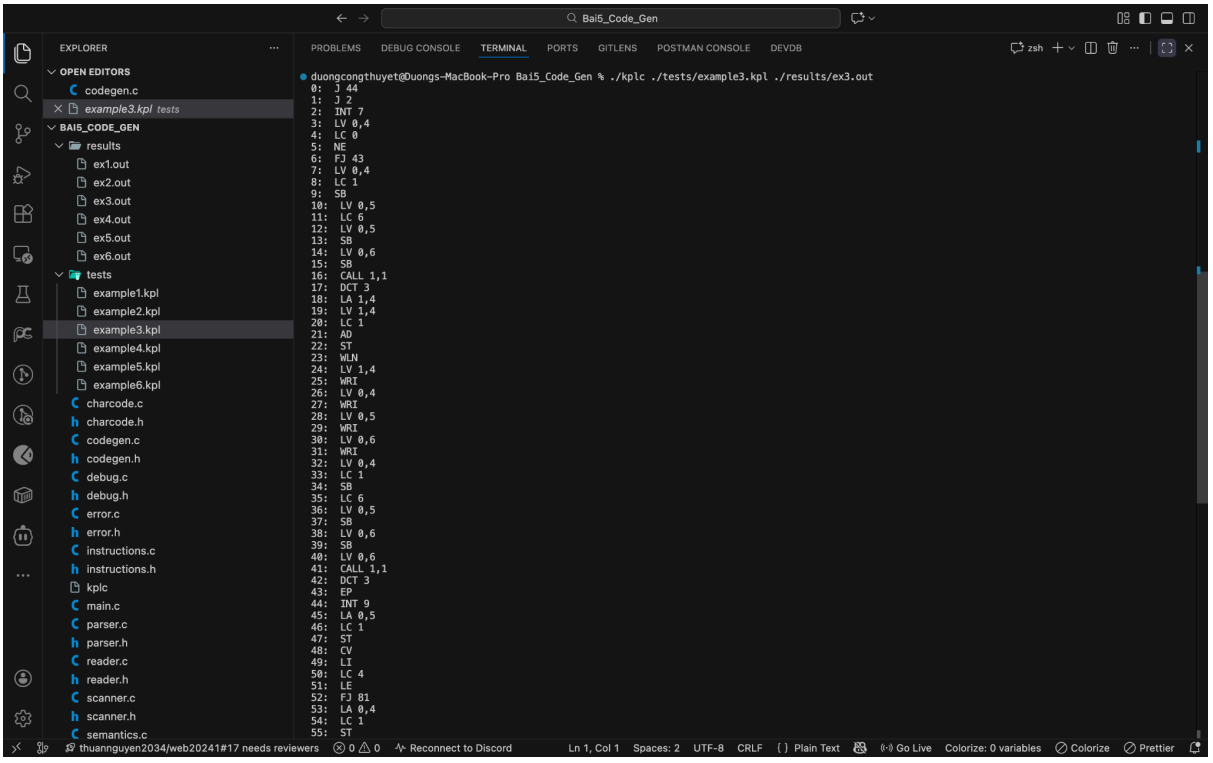
III. Kết quả

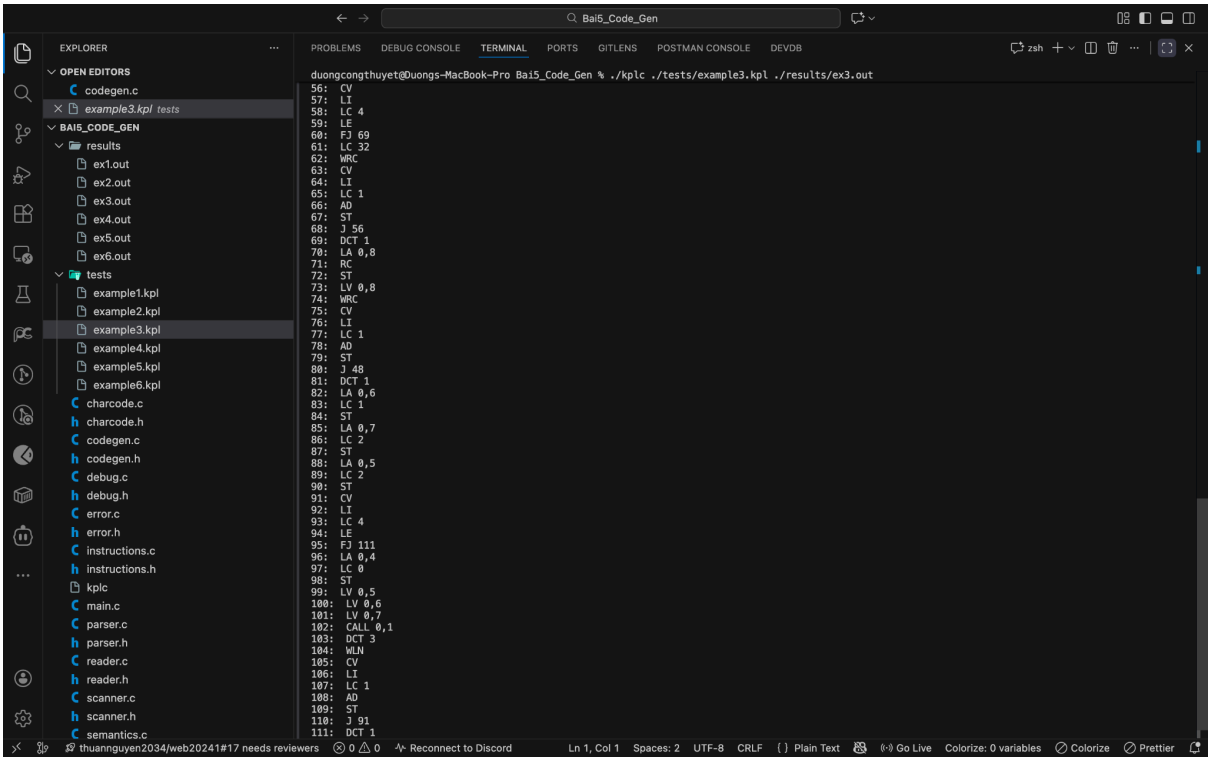
1. example1.kpl không có chương trình con

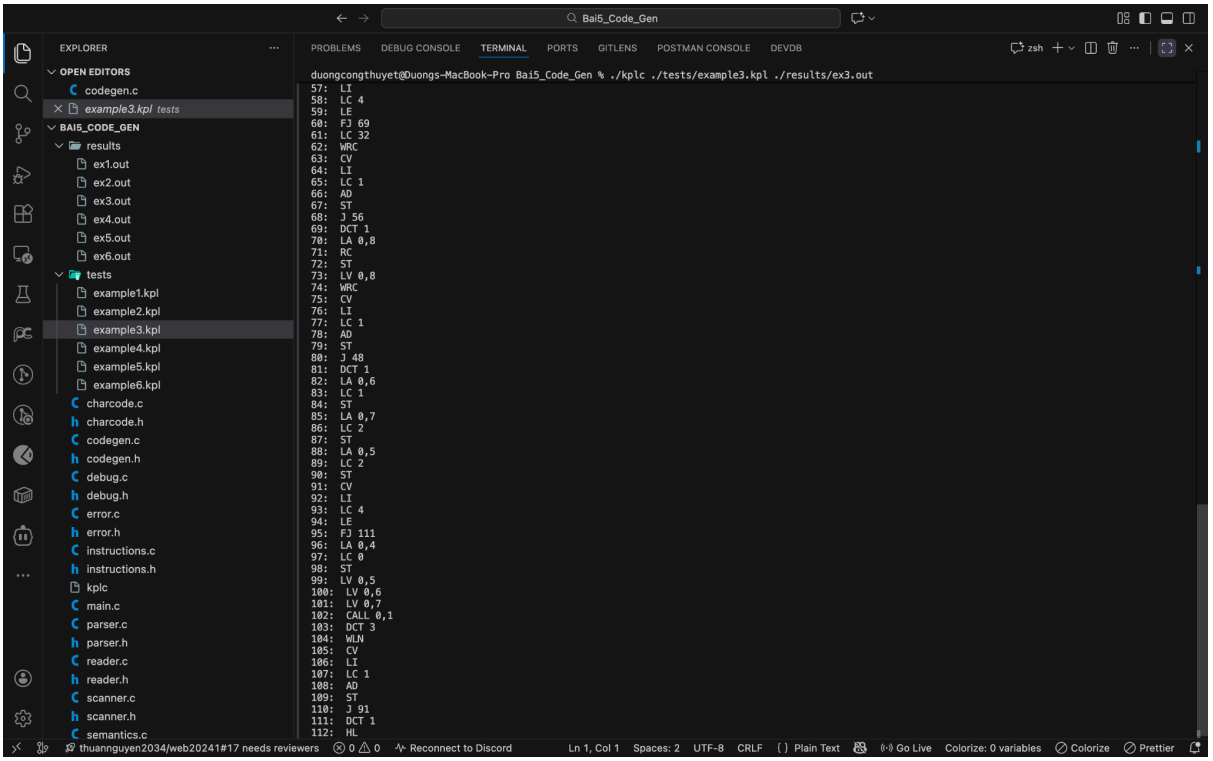


2. example3.kpl có chương trình con




Bai5_Code_Gen
EXPLORER
OPEN EDITORS
codegen.c
example3.kpl tests
BAI5_CODE_GEN
results
ex1.out
ex2.out
ex3.out
ex4.out
ex5.out
ex6.out
tests
example1.kpl
example2.kpl
example3.kpl
example4.kpl
example5.kpl
example6.kpl
charcode.c
charcode.h
codegen.c
codegen.h
debug.c
debug.h
error.c
error.h
instructions.c
instructions.h
kplc
main.c
parser.c
parser.h
reader.c
reader.h
scanner.c
scanner.h
semantics.c
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
POSTMAN CONSOLE
DEVDB
zsh
duongcongthuyet@Duongs-MacBook-Pro Bai5_Code_Gen % ./kplc ./tests/example3.kpl ./results/ex3.out
0: J 44
1: J 2
2: INT 7
3: LV 0,4
4: LC 0
5: NE
6: FJ 43
7: LV 0,4
8: LC 1
9: SB
10: LV 0,5
11: LC 6
12: LV 0,5
13: SB
14: LV 0,6
15: SB
16: CALL 1,1
17: DCT 3
18: LA 1,4
19: LV 1,4
20: LC 1
21: AD
22: ST
23: WLN
24: LV 1,4
25: WRI
26: LV 0,4
27: WRI
28: LV 0,5
29: WRI
30: LV 0,6
31: WRI
32: LV 0,4
33: LC 1
34: SB
35: LC 6
36: LV 0,5
37: SB
38: LV 0,6
39: SB
40: LV 0,6
41: CALL 1,1
42: DCT 3
43: EP
44: INT 9
45: LA 0,5
46: LC 1
47: ST
48: CV
49: LI
50: LC 4
51: LE
52: FJ 81
53: LA 0,4
54: LC 1
55: ST
thuannguyen2034/web20241#17 needs reviewers
Reconnect to Discord
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
Plain Text
Go Live
Colorize: 0 variables
Colorize
Prettier


Bai5_Code_Gen
EXPLORER
OPEN EDITORS
codegen.c
example3.kpl tests
BAI5_CODE_GEN
results
ex1.out
ex2.out
ex3.out
ex4.out
ex5.out
ex6.out
tests
example1.kpl
example2.kpl
example3.kpl
example4.kpl
example5.kpl
example6.kpl
charcode.c
charcode.h
codegen.c
codegen.h
debug.c
debug.h
error.c
error.h
instructions.c
instructions.h
kplc
main.c
parser.c
parser.h
reader.c
reader.h
scanner.c
scanner.h
semantics.c
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
POSTMAN CONSOLE
DEVDB
zsh
duongcongthuyet@Duongs-MacBook-Pro Bai5_Code_Gen % ./kplc ./tests/example3.kpl ./results/ex3.out
56: CV
57: LI
58: LC 4
59: LE
60: FJ 69
61: LC 32
62: WRC
63: CV
64: LI
65: LC 1
66: AD
67: ST
68: J 56
69: DCT 1
70: LA 0,8
71: RC
72: ST
73: LV 0,8
74: WRC
75: CV
76: LI
77: LC 1
78: AD
79: ST
80: J 48
81: DCT 1
82: LA 0,6
83: LC 1
84: ST
85: LA 0,7
86: LC 2
87: ST
88: LA 0,5
89: LC 2
90: ST
91: CV
92: LI
93: LC 4
94: LE
95: FJ 111
96: LA 0,4
97: LC 0
98: ST
99: LV 0,5
100: LV 0,6
101: LV 0,7
102: CALL 0,1
103: DCT 3
104: WLN
105: CV
106: LI
107: LC 1
108: AD
109: ST
110: J 91
111: DCT 1
thuannguyen2034/web20241#17 needs reviewers
Reconnect to Discord
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
Plain Text
Go Live
Colorize: 0 variables
Colorize
Prettier


Bai5_Code_Gen
EXPLORER
OPEN EDITORS
codegen.c
example3.kpl tests
BAI5_CODE_GEN
results
ex1.out
ex2.out
ex3.out
ex4.out
ex5.out
ex6.out
tests
example1.kpl
example2.kpl
example3.kpl
example4.kpl
example5.kpl
example6.kpl
charcode.c
charcode.h
codegen.c
codegen.h
debug.c
debug.h
error.c
error.h
instructions.c
instructions.h
kplc
main.c
parser.c
parser.h
reader.c
reader.h
scanner.c
scanner.h
semantics.c
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
POSTMAN CONSOLE
DEVDB
zsh
duongcongthuyet@Duongs-MacBook-Pro Bai5_Code_Gen % ./kplc ./tests/example3.kpl ./results/ex3.out
57: LI
58: LC 4
59: LE
60: FJ 69
61: LC 32
62: WRC
63: CV
64: LI
65: LC 1
66: AD
67: ST
68: J 56
69: DCT 1
70: LA 0,8
71: RC
72: ST
73: LV 0,8
74: WRC
75: CV
76: LI
77: LC 1
78: AD
79: ST
80: J 48
81: DCT 1
82: LA 0,6
83: LC 1
84: ST
85: LA 0,7
86: LC 2
87: ST
88: LA 0,5
89: LC 2
90: ST
91: CV
92: LI
93: LC 4
94: LE
95: FJ 111
96: LA 0,4
97: LC 0
98: ST
99: LV 0,5
100: LV 0,6
101: LV 0,7
102: CALL 0,1
103: DCT 3
104: WLN
105: CV
106: LI
107: LC 1
108: AD
109: ST
110: J 91
111: DCT 1
112: HL
thuannguyen2034/web20241#17 needs reviewers
0 0
Reconnect to Discord
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
Plain Text
Go Live
Colorize: 0 variables
Colorize
Prettier