

# Báo cáo Compiler tuần 3

## Vũ Trần Tuấn Minh - 20225891

### KẾT QUẢ THỰC HIỆN VỚI CÁC VÍ DỤ

#### EXAMPLE1.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example1.kpl
Program EXAMPLE1
```

**Giải thích:** Chỉ có tên chương trình mà không có khai báo nào.

#### EXAMPLE2.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example2.kpl
Program EXAMPLE2
    Var N : Int
    Function F : Int
        Param N : Int
```

**Giải thích:**

- Biến toàn cục **N** kiểu INTEGER
- Hàm **F** nhận tham số **N** kiểu INTEGER, trả về INTEGER
- Tham số **N** trong hàm F nằm trong scope của F, khác với biến **N** toàn cục

#### EXAMPLE3.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example3.kpl
Program EXAMPLE3
    Var I : Int
    Var N : Int
    Var P : Int
    Var Q : Int
    Var C : Char
    Procedure HANOI
        Param N : Int
        Param S : Int
        Param Z : Int
```

**Giải thích:**

- Nhiều biến toàn cục: I, N, P, Q kiểu INTEGER và C kiểu CHAR
- Thủ tục HANOI nhận 3 tham số: N, S, Z đều kiểu INTEGER
- Các tham số được đăng ký trong scope của thủ tục HANOI

---

## EXAMPLE4.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example4.kpl
Program EXAMPLE4
    Const MAX = 10
    Type T = Int
    Var A : Arr(10,Int)
    Var N : Int
    Var CH : Char
    Procedure INPUT
        Var I : Int
        Var TMP : Int

    Procedure OUTPUT
        Var I : Int

    Function SUM : Int
        Var I : Int
        Var S : Int
```

### Giải thích:

- Hằng số MAX = 10
- Kiểu T là alias của INTEGER
- Biến A là mảng 10 phần tử INTEGER
- Ba chương trình con: INPUT, OUTPUT (thủ tục), SUM (hàm)
- Mỗi chương trình con có biến cục bộ riêng
- Biến I xuất hiện trong nhiều scope khác nhau (không xung đột)

---

## EXAMPLE5.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example5.kpl
Program EXAMPLE5
    Const C = 1
    Type T = Char
    Function F : Char
        Param I : Int
        Const B = 1
        Type A = Arr(5,Char)
```

### Giải thích:

- Khai báo toàn cục: hằng C, kiểu T
- Hàm F có:

- Tham số lì kiểu INTEGER
  - Kiểu trả về CHAR
  - Hằng cục bộ B = 1
  - Kiểu cục bộ A là mảng 5 ký tự
- 

## EXAMPLE6.KPL

```
oc@oc:~/Downloads/compiler_t3/SymTab_Incompleted$ ./parser example6.kpl
Program EXAMPLE6
    Const C1 = 10
    Const C2 = 'a'
    Type T1 = Arr(10,Int)
    Var V1 : Int
    Var V2 : Arr(10,Arr(10,Int))
    Function F : Int
        Param P1 : Int
        Param VAR P2 : Char

    Procedure P
        Param V1 : Int
        Const C1 = 'a'
        Const C3 = 10
        Type T1 = Int
        Type T2 = Arr(10,Int)
        Var V2 : Arr(10,Int)
        Var V3 : Char
```

Giải thích:

- **Scope toàn cục:**
  - Hai hằng: C1 = 10 (int), C2 = 'a' (char)
  - Kiểu T1 là mảng 10 integer
  - V1 là integer
  - V2 là mảng 2 chiều 10x10 integer
- **Hàm F:**
  - Tham số P1 (tham trị) kiểu INTEGER
  - Tham số P2 (tham biến - VAR) kiểu CHAR
  - Trả về INTEGER
- **Thủ tục P:**
  - Tham số V1 (tham trị) - trùng tên với biến toàn cục nhưng khác scope
  - Khai báo lại C1 = 'a' (char) - che khuất (shadow) C1 toàn cục
  - Khai báo lại T1 = INTEGER - che khuất T1 toàn cục
  - Các khai báo cục bộ khác: C3, T2, V2, V3

## CÁC THÀNH PHẦN CHÍNH ĐÃ HOÀN THIỆN

Hàm **compileProgram()**

**Chức năng:** Tạo đối tượng chương trình và quản lý scope chương trình

```
void compileProgram(void) {
    Object* program;

    eat(KW_PROGRAM);
    eat(TK_IDENT);

    // Tạo đối tượng chương trình với tên từ token hiện tại
    program = createProgramObject(currentToken->string);

    // Vào scope của chương trình
    enterBlock(program->progAttrs->scope);

    eat(SB_SEMICOLON);
    compileBlock();
    eat(SB_PERIOD);

    // Ra khỏi scope khi kết thúc
    exitBlock();

}
```

**Giải thích:**

- Tạo đối tượng PROGRAM với tên lấy từ identifier sau từ khóa PROGRAM
- Sử dụng `enterBlock()` để đặt `currentScope = scope` của chương trình
- Sau khi duyệt xong, dùng `exitBlock()` để trở về scope bên ngoài

## Hàm `compileBlock()` - Xử lý khai báo hằng

**Chức năng:** Tạo và đăng ký các đối tượng hằng số

```
void compileBlock(void) {
    Object* constObj;
    ConstantValue* constValue;

    if (lookAhead->tokenType == KW_CONST) {
        eat(KW_CONST);
        do {
            eat(TK_IDENT);

            // Tạo đối tượng hằng với tên
            constObj = createConstantObject(currentToken->string);

            eat(SB_EQ);

            // Lấy giá trị hằng từ biểu thức
    }
```

```

constValue = compileConstant();
constObj->constAttrs->value = constValue;

// Đăng ký vào scope hiện tại
declareObject(constObj);

eat(SB_SEMICOLON);
} while (lookAhead->tokenType == TK_IDENT);

compileBlock2();
}
else compileBlock2();

}

```

**Ví dụ:**

CONST MAX = 10;

Tạo đối tượng: MAX: CST = 10

## Hàm **compileBlock2()** - Xử lý khai báo kiểu

**Chức năng:** Tạo và đăng ký các đối tượng kiểu người dùng định nghĩa

```

void compileBlock2(void) {
    Object* typeObj;
    Type* actualType;

    if (lookAhead->tokenType == KW_TYPE) {
        eat(KW_TYPE);
        do {
            eat(TK_IDENTIFIER);

            // Tạo đối tượng kiểu
            typeObj = createTypeObject(currentToken->string);

            eat(SB_EQ);

            // Lấy kiểu thực tế (có thể là INT, CHAR, ARRAY)
            actualType = compileType();
            typeObj->typeAttrs->actualType = actualType;

            // Đăng ký kiểu
            declareObject(typeObj);

            eat(SB_SEMICOLON);
        } while (lookAhead->tokenType == TK_IDENTIFIER);
    }
}

```

```

    compileBlock3();
}
else compileBlock3();
}

```

**Ví dụ:**

TYPE T1 = ARRAY[10] OF INTEGER;

Tạo đối tượng: T1 : TY = Arr(10, Int)

### Hàm **compileBlock3()** - Xử lý khai báo biến

**Chức năng:** Tạo và đăng ký các đối tượng biến

```

void compileBlock3(void) {
    Object* varObj;
    Type* varType;

    if (lookAhead->tokenType == KW_VAR) {
        eat(KW_VAR);
        do {
            eat(TK_IDENT);

            // Tạo đối tượng biến
            varObj = createVariableObject(currentToken->string);

            eat(SB_COLON);

            // Lấy kiểu của biến
            varType = compileType();
            varObj->varAttrs->type = varType;

            // Đăng ký biến
            declareObject(varObj);

            eat(SB_SEMICOLON);
        } while (lookAhead->tokenType == TK_IDENT);

        compileBlock4();
    }
    else compileBlock4();
}

```

**Ví dụ:**

VAR N : INTEGER;

Tạo đối tượng: N: VAR : Int

### Hàm **compileFuncDecl()** - Xử lý khai báo hàm

**Chức năng:** Tạo và đăng ký đối tượng hàm, quản lý scope của hàm

```
void compileFuncDecl(void) {
    Object* funcObj;
    Type* returnType;

    eat(KW_FUNCTION);
    eat(TK_IDENT);

    // Tạo đối tượng hàm
    funcObj = createFunctionObject(currentToken->string);

    // Đăng ký hàm vào scope HIỆN TẠI (scope bên ngoài)
    declareObject(funcObj);

    // Chuyển vào scope của hàm để khai báo tham số và biến cục bộ
    enterBlock(funcObj->funcAttrs->scope);

    // Biên dịch tham số (các tham số sẽ được thêm vào scope của hàm)
    compileParams();

    eat(SB_COLON);

    // Lấy kiểu trả về
    returnType = compileBasicType();
    funcObj->funcAttrs->returnType = returnType;

    eat(SB_SEMICOLON);

    // Biên dịch block bên trong hàm
    compileBlock();
    eat(SB_SEMICOLON);

    // Ra khỏi scope của hàm
    exitBlock();
}
```

#### Ví dụ:

FUNCTION F(x : INTEGER) : INTEGER;

Tạo đối tượng: F: FN: INT → INT với tham số x: PAR : INT

### Hàm **compileProcDecl()** - Xử lý khai báo thủ tục

**Chức năng:** Tương tự hàm nhưng không có kiểu trả về

```
void compileProcDecl(void) {
    Object* procObj;

    eat(KW_PROCEDURE);
    eat(TK_IDENT);

    // Tạo đối tượng thủ tục
    procObj = createProcedureObject(currentToken->string);

    // Đăng ký thủ tục
    declareObject(procObj);

    // Vào scope của thủ tục
    enterBlock(procObj->procAttrs->scope);

    // Biên dịch tham số
    compileParams();

    eat(SB_SEMICOLON);
    compileBlock();
    eat(SB_SEMICOLON);

    // Ra khỏi scope
    exitBlock();
}
```

### Hàm **compileParam()** - Xử lý khai báo tham số

**Chức năng:** Tạo và đăng ký tham số hình thức

```
void compileParam(void) {
    Object* param;
    Type* paramType;
    enum ParamKind paramKind;

    switch (lookAhead->tokenType) {
        case TK_IDENT:
            // Tham trị (pass by value)
            paramKind = PARAM_VALUE;
            eat(TK_IDENT);
```

```

// Tạo tham số với owner là hàm/thủ tục hiện tại
param = createParameterObject(currentToken->string,
                             paramKind,
                             symtab->currentScope->owner);

eat(SB_COLON);
paramType = compileBasicType();
param->paramAttrs->type = paramType;

// Khai báo tham số (vào cả paramList VÀ objList)
declareObject(param);
break;

case KW_VAR:
    // Tham biến (pass by reference)
    paramKind = PARAM_REFERENCE;
    eat(KW_VAR);
    eat(TK_IDENT);

    param = createParameterObject(currentToken->string,
                                 paramKind,
                                 symtab->currentScope->owner);

    eat(SB_COLON);
    paramType = compileBasicType();
    param->paramAttrs->type = paramType;

    declareObject(param);
    break;
}
}

```

**Lưu ý quan trọng:** Hàm `declareObject()` trong `symtab.c` đã xử lý việc thêm tham số vào CẢ HAI nơi:

1. `paramList` của hàm/thủ tục
2. `objList` của scope hiện tại

## Các hàm xử lý kiểu và hằng số

### `compileType()` - Xử lý kiểu dữ liệu

```
Type* compileType(void) {
    Type* type;
    Type* elementType;
    int arraySize;
    Object* obj;
```

```

switch (lookAhead->tokenType) {
    case KW_INTEGER:
        eat(KW_INTEGER);
        type = makeIntType();
        break;

    case KW_CHAR:
        eat(KW_CHAR);
        type = makeCharType();
        break;

    case KW_ARRAY:
        eat(KW_ARRAY);
        eat(SB_LSEL);
        eat(TK_NUMBER);
        arraySize = currentToken->value;
        eat(SB_RSEL);
        eat(KW_OF);
        elementType = compileType(); // Đệ quy cho mảng nhiều chiều
        type = makeArrayType(arraySize, elementType);
        break;

    case TK_IDENT:
        eat(TK_IDENT);
        // Tra bảng ký hiệu để lấy kiểu đã định nghĩa
        obj = lookupObject(currentToken->string);

        if (obj == NULL) {
            error(ERR_UNDECLARED_TYPE, currentToken->lineNo, currentToken->colNo);
        }

        if (obj->kind != OBJ_TYPE) {
            error(ERR_INVALID_TYPE, currentToken->lineNo, currentToken->colNo);
        }

        // Sao chép kiểu
        type = duplicateType(obj->typeAttrs->actualType);
        break;
    }
    return type;
}

```

### Giải thích:

- Xử lý 4 loại kiểu: INTEGER, CHAR, ARRAY, và kiểu người dùng định nghĩa
- Với ARRAY: đệ quy để xử lý mảng nhiều chiều
- Với IDENT: tra bảng ký hiệu và sao chép kiểu

### **compileConstant() và compileConstant2()**

```
ConstantValue* compileConstant(void) {
    ConstantValue* constValue;

    switch (lookAhead->tokenType) {
        case SB_PLUS:
            eat(SB_PLUS);
            constValue = compileConstant2();
            break;
        case SB_MINUS:
            eat(SB_MINUS);
            constValue = compileConstant2();
            // Đáo dấu giá trị
            if (constValue->type == TP_INT) {
                constValue->intValue = -(constValue->intValue);
            }
            break;
        case TK_CHAR:
            eat(TK_CHAR);
            constValue = makeCharConstant(currentToken->string[0]);
            break;
        default:
            constValue = compileConstant2();
            break;
    }
    return constValue;
}

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);
            // Tra bảng ký hiệu để lấy giá trị hằng
            obj = lookupObject(currentToken->string);

            if (obj == NULL) {
                error(ERR_UNDECLARED_INT_CONSTANT, currentToken->lineNo,
                    currentToken->colNo);
            }
    }
}
```

```
if (obj->kind != OBJ_CONSTANT) {
    error(ERR_INVALID_CONSTANT, currentToken->lineNo, currentToken->colNo);
}

if (obj->constAttrs->value->type != TP_INT) {
    error(ERR_UNDECLARED_INT_CONSTANT, currentToken->lineNo,
currentToken->colNo);
}

// Sao chép giá trị hằng
constValue = duplicateConstantValue(obj->constAttrs->value);
break;
}
return constValue;
}
```