ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thực hành
Xây dựng chương trình dịch

Bài 3: Phân tích ngữ nghĩa

Project 2: Xây dựng bảng ký hiệu

Xây dựng bảng ký hiệu cho ngôn ngữ KPL

- Khởi tạo và giải phóng
- Khai báo hằng
- Khai báo kiểu
- Khai báo biến
- Khai báo thủ tục và hàm
- Khai báo tham số

Khởi tạo và giải phóng

```
int compile(char *fileName) {
  …
  // Khởi tạo bảng ký hiệu
  initSymTab();
  // Dịch chương trình
  compileProgram();
  // In chương trình để kiểm tra kết quả
  printObject(symtab->program,0);
  // Giải phóng bảng ký hiệu
  cleanSymTab();
  …
}
```

Khởi tạo bảng ký hiệu cho chương trình

- Đối tượng chương trình được khởi tạo ở hàm:

```
void compileProgram(void);
```

- Sau khi khởi tạo cho đối tượng chương trình, vào khối chính bằng hàm `enterBlock()`
- Sau khi đọc xong chương trình, ra khỏi khối chính bằng hàm `exitBlock()`

Khai báo hằng

- Các đối tượng hằng số được tạo ra và khai báo ở hàm `compileBlock()`
- Giá trị của hằng số được lấy từ quá trình duyệt giá trị hằng qua hàm

  `ConstantValue* compileConstant(void)`

  - Nếu giá trị hằng là một định danh hằng, phải tra bảng ký hiệu để lấy giá trị tương ứng
- Sau khi duyệt xong một hằng số, phải đăng ký vào block hiện tại bằng hàm `declareObject`

Khai báo kiểu người dùng định nghĩa

- Các đối tượng kiểu được tạo ra và khai báo ở hàm `compileBlock2()`
- Kiểu thực tế được lấy từ quá trình duyệt kiểu bằng hàm

  `Type* compileType(void)`

  - Nếu gặp định danh kiểu thì phải tra bảng ký hiệu để lấy kiểu tương ứng
- Sau khi duyệt xong một kiểu người dùng định nghĩa, phải đăng ký vào block hiện tại bằng hàm `declareObject`

Khai báo biến

- Các đối tượng biến được tạo ra và khai báo ở hàm `compileBlock3()`
- Kiểu của biến được lấy từ quá trình duyệt kiểu bằng hàm

  `Type* compileType(void)`
- Lưu trữ phạm vi hiện tại vào danh sách thuộc tính của đối tượng biến để phục vụ mục đích sinh mã sau này
- Sau khi duyệt xong một biến, phải đăng ký vào block hiện tại bằng hàm `declareObject`

Khai báo hàm

- Các đối tượng hàm được tạo ra và khai báo ở hàm `compileFuncDecl()`
- Các thuộc tính của đối tượng hàm sẽ được cập nhật bao gồm:
  - danh sách tham số: `compileParams`
  - kiểu dữ liệu trả về: `compileType`
  - phạm vi của hàm
- Lưu ý đăng ký đối tượng hàm vào block hiện tại và chuyển block hiện tại sang block của hàm trước khi duyệt tiếp các đối tượng cục bộ

Khai báo thủ tục

- Các đối tượng thủ tục được tạo ra và khai báo ở hàm `compileProcDecl()`
- Các thuộc tính của đối tượng thủ tục sẽ được cập nhật bao gồm:
  - danh sách tham số: `compileParams`
  - phạm vi của thủ tục
- Lưu ý đăng ký đối tượng thủ tục vào block hiện tại và chuyển block hiện tại sang block của hàm trước khi duyệt tiếp các đối tượng cục bộ

Khai báo tham số hình thức

- Các đối tượng tham số hình thức được tạo ra và khai báo ở hàm `compileParam()`
- Các thuộc tính của đối tượng tham số hình thức bao gồm:
  - Kiểu dữ liệu cơ bản
  - Tham biến (PARAM_REFERENCE) hoặc tham trị (PARAM_VALUE)
- Lưu ý: đối tượng tham số hình thức nên được đăng ký vào đồng thời vào cả thuộc tính paramList của hàm/thủ tục hiện tại, cả vào danh sách đối tượng trong phạm vi hiện tại

Nhiệm vụ thực hành

- Tìm hiểu lại cấu trúc của bộ parser (có thay đổi)
- Bổ xung các đoạn code vào những hàm có đánh dấu TODO để thực hiện các công việc đăng ký đối tượng
- Biên dịch và thử nghiệm với các ví dụ mẫu

So sánh với bài 1: Tạo chương trình

```
 obj = createProgramObject("PRG");
  enterBlock(obj->progAttrs->scope);
```

```
// TODO: create, enter, and exit program block
  eat(KW_PROGRAM);
  eat(TK_IDENT);
  eat(SB_SEMICOLON);
  compileBlock();
  eat(SB_PERIOD);
}
```

So sánh với bài 1: Tạo hằng

```
obj = createConstantObject("c1");
  obj->constAttrs->value =
makeIntConstant(10);
  declareObject(obj);
  obj = createConstantObject("c2");
  obj->constAttrs->value =
makeCharConstant('a');
  declareObject(obj);
```

```
void compileBlock(void) {
  // TODO: create and declare constant objects
  if (lookAhead->tokenType == KW_CONST) {
    eat(KW_CONST);
    do {
      eat(TK_IDENT);
      eat(SB_EQ);
      compileConstant();
      eat(SB_SEMICOLON);
    } while (lookAhead->tokenType == TK_IDENT);

    compileBlock2();
  }
  else compileBlock2();
}
```

Hàm compileConstant2

```
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 = lookupObject(currentToken->string);
    if ((obj != NULL) && (obj->kind == OBJ_CONSTANT) && (obj->constAttrs->value->type ==
TP_INT))
      constValue = duplicateConstantValue(obj->constAttrs->value);
    else
      error(ERR_UNDECLARED_INT_CONSTANT,currentToken->lineNo, currentToken->colNo);
    break;
  default:
    error(ERR_INVALID_CONSTANT, lookAhead->lineNo, lookAhead->colNo);
    break;
  }
  return constValue;
}
```

So sánh với bài 1: Tạo kiểu

```
obj = createTypeObject("t1");
  obj->typeAttrs->actualType =
makeArrayType(10,makeIntType());
  declareObject(obj);
```

```
compileBlock2(void) {
 // TODO: create and declare type objects
 if (lookAhead->tokenType == KW_TYPE) {
   eat(KW_TYPE);

   do {
     eat(TK_IDENT);
     eat(SB_EQ);
     compileType();
     eat(SB_SEMICOLON);
   } while (lookAhead->tokenType == TK_IDENT);

   compileBlock3();
 }
 else compileBlock3();
}
```

So sánh với bài 1: Tạo biến

```
obj =
createVariableObject("v1");
  obj->varAttrs->type =
makeIntType();
  declareObject(obj);
obj =
createVariableObject("v2");
  obj->varAttrs->type =
makeArrayType(10,makeAr
rayType(10,makeIntType()))
;
  declareObject(obj);
```

```
void compileBlock3(void) {
 // TODO: create and declare variable objects
 if (lookAhead->tokenType == KW_VAR) {
   eat(KW_VAR);
   do {
     eat(TK_IDENT);
     eat(SB_COLON);
     compileType();
     eat(SB_SEMICOLON);
   } while (lookAhead->tokenType == TK_IDENT);
   compileBlock4();
 }
 else compileBlock4();
}
```

Sử dụng các hàm duplicate

```
Type* compileType(void) {
 // TODO: create and return a type
 Type* type;
 switch (lookAhead->tokenType) {
 case KW_INTEGER:
  eat(KW_INTEGER);
  break;
 case KW_CHAR:
  eat(KW_CHAR);
  break;
 case KW_ARRAY:
  eat(KW_ARRAY);
  eat(SB_LSEL);
  eat(TK_NUMBER);
  eat(SB_RSEL);
  eat(KW_OF);
  elementType)= compileType();
  type = makeArrayType(arraySize, elementType);
  break;
 case TK_IDENT:
  eat(TK_IDENT);
  break;
 default:
  error(ERR_INVALID_TYPE, lookAhead->lineNo, lookAhead->colNo);
  break;
 }
 return type;
}
```

```
ConstantValue* compileUnsignedConstant(void) {
 // TODO: create and return an unsigned constant value
 ConstantValue* constValue;
 switch (lookAhead->tokenType) {
 case TK_NUMBER:
  eat(TK_NUMBER);
constValue = makeIntConstant(currentToken->value);
  break;
 case TK_IDENT:
  eat(TK_IDENT);
.....
constValue = duplicateConstantValue(obj->constAttrs->value);
  break;
 case TK_CHAR:
  eat(TK_CHAR);
  break;
 default:
  error(ERR_INVALID_CONSTANT, lookAhead->lineNo, lookAhead->colNo);
  break;
 }
 return constValue;
}
```

So sánh với bài 1: Tạo hàm

```
obj = createFunctionObject("f");
  obj->funcAttrs->returnType = makeIntType();
  declareObject(obj);
      enterBlock(obj->funcAttrs->scope);
     obj = createParameterObject("p1", PARAM_VALUE, symtab->currentScope->owner);
    obj->paramAttrs->type = makeIntType();
    declareObject(obj);
    obj = createParameterObject("p2", PARAM_REFERENCE, symtab->currentScope->owner);
    obj->paramAttrs->type = makeCharType();
    declareObject(obj);


    exitBlock();
```

```
void compileFuncDecl(void) {
  // TODO: create and declare a function object
  eat(KW_FUNCTION);
  eat(TK_IDENT);
  compileParams();
  eat(SB_COLON);
  returnType = compileBasicType();
  funcObj->funcAttrs->returnType = returnType;

  eat(SB_SEMICOLON);
  compileBlock();
  eat(SB_SEMICOLON);
}
```

So sánh với bài 1: Tạo tham số

```
obj = createParameterObject("p1", PARAM_VALUE, symtab->currentScope-> owner);
    obj->paramAttrs->type = makeIntType();
    declareObject(obj);
    obj = createParameterObject("p2", PARAM_REFERENCE, symtab- >currentScope-> owner);
    obj->paramAttrs->type = makeCharType();
    declareObject(obj);
```

```
void compileParam(void) {
  // TODO: create and declare a parameter
  switch (lookAhead->tokenType) {
  case TK_IDENT:
    eat(TK_IDENT);
    eat(SB_COLON);
    compileBasicType();
    break;
  case KW_VAR:
    eat(KW_VAR);
    eat(TK_IDENT);
    eat(SB_COLON);
    compileBasicType();
    break;
  default:
    error(ERR_INVALID_PARAMETER, lookAhead->lineNo, lookAhead->colNo);
    break;
  }
}
```

So sánh với bài 1: Tạo thủ tục

```
obj = createProcedureObject("p");
  declareObject(obj);
      enterBlock(obj->procAttrs->scope);
 exitBlock();
```

```
 void compileProcDecl(void)
{
  // TODO: create and declare a procedure object
  eat(KW_PROCEDURE);
  eat(TK_IDENT);
  compileParams();
  eat(SB_SEMICOLON);
  compileBlock();
  eat(SB_SEMICOLON);
}
```