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

Experiment in Compiler Construction

Semantic Analysis (2)
Scope Management

Overview

- Implementation of symbol tables
- Checking duplicate object declaration
- Checking reference to object

Implement symbol table for KPL

- Initialize and Clean symbol table
- Constant declaration
- Type declaration
- Variable declaration
- Function/Procedure declaration
- Parameter declaration

Initialize & Clean a symbol table

```
int compile(char *fileName) {

  // Initialize a symbol table
  initSymTab();
  // Compile the program
  compileProgram();
  // Display result for checking
  printObject(symtab->program,0);
  // Clean symbol table
  cleanSymTab();
  …
}
```

Initialize program

- The program object is initialized by

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

- After program initialization, we enter the outermost block by `enterBlock()`
- When program is completely analysed, we exit by `exitBlock()`

Constant declaration

- Constant objects are created and declared inside the function `compileBlock()`
- During analysing process, constants' values are filled by

```
ConstantValue* compileConstant(void)
```

*In case a constant's value is identifier constant, like **const b=a**; refer to symbol table to find actual value.*

- When a constant has been analysed, he has to be declared in current block by function `declareObject`

Example

- Insert information of a constant
- obj = createConstantObject("c1");

```
obj->constAttrs->value = makeIntConstant(10);
declareObject(obj);
```

void compileBlock(void)

```
{  Object* constObj;
 ConstantValue* constValue;
 if (lookAhead->tokenType == KW_CONST) {
   eat(KW_CONST);
   do {
     eat(TK_IDENT);
         constObj = createConstantObject(currentToken->string);
     eat(SB_EQ);
         constValue = compileConstant();
          constObj->constAttrs->value = constValue;
     declareObject(constObj);
     eat(SB_SEMICOLON);
   } while (lookAhead->tokenType == TK_IDENT);
   compileBlock2();
 }
 else compileBlock2();
}
```

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); // check if the integer constant identifier is declared
    if (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;
}
```

User-defined type declaration

- Type objects are created and declared inside the function `compileBlock2()`
- Actual type is learned during the analysing by function `Type* compileType(void)`
  - If we meet identifier type, refer to symbol table to find actual type
- When a user-defined type has been analysed, he has to be declared in current block by function `declareObject`

Variable declaration

- Variable objects are created and declared inside function

  `compileBlock3()`

- Type of a variable is filled when analysing type by using function

  `Type* compileType(void)`

- For later code generation, one of variable object's attributes should be the current scope.
- When a variable object is analysed, he has to be declared in current block by function `declareObject`

Function declaration

- Function objects are created and declared in function `compileFuncDecl()`
- Attributes of a function object need to be filled include:
  - List of parameters, in function `compileParams`
  - Return type, in function `compileType`
  - Function's scope
- Note:The function object has to be declared in current block

  Update function scope as currentScope before deal with function local object.

Procedure declaration

- Function objects are created and declared in function `compileProcDecl()`
- Attributes of a function object need to be filled include:
  - List of parameters, in function `compileParams`
- Note:The function object has to be declared in current block

  Update function scope as currentScope before deal with function local object.

Parameter declaration

- Parameter objects are created and declared in function `compileParam()`
- Parameter objects' attributes:
  - Data type of parameter: a basic type
  - Kind of parameter: Call by value (PARAM_VALUE) or call by reference (PARAM_REFERENCE)
- Note: parameter objects should be declared in both
  - Current function's list of parameter (`paramList`)
  - Current function's list of local objects (`objectList`).

Checking fresh identifier

- A fresh identifier is an identifier that is new (has not been used) in current scope
- Checking fresh identifier is task of function

```
void checkFreshIdent(char *name);
```

Checking fresh identifier

- Checking fresh identifier is performed in
  - Constant declaration
  - User-defined type declaration
  - Variable declaration
  - Parameter declaration
  - Function declaration
  - Procedure declaration

Checking declared constant

- Performed when there is a reference to a constant, e.g:
  - When analysing an unsigned constant
  - When analysing an constant
- If a constant is not declared in current block, search in outer blocks.
- The value of declared constant will be the value of the constant that we are dealing with
  - Share the value
  - Do not share the value → `duplicateConstantValue`

Checking declared type

- Performed when there is a reference to a type, e.g: when analysing a type in function `compileType`
- If a type is not declared in current block, search in outer blocks
- The actual type of refered type name will be used to create the type we are dealing with
  - Share type
  - Do not share type → `duplicateType`

Checking declared variable

- Performed when there is a reference to a variable, e.g:
  - In for statement
  - When analysing factor
- If a variable is not declared in current block, search in outer blocks.

Checking declared LHS

- An identifier that appears in the left-hand side of an assign statement or in a factor possibly is:
  - Current function
  - A declared variable
    - If the variable's type is array type, the array index must follow the variable's name.
- Variable is different from parameters and current function.

Checking declared function

- Performed when a function is referred, e.g
  - As left-hand side of assign statement (current function)
  - In a factor (a list of parameters will follows function's name)
- If a function is not declared in current block, search in outer blocks.
- Global functions: READC, READI

Checking a declared procedure

- Performed when a procedure is referred, e.g:
  - In CALL statement
- If a procedure is not declared in current block, search in outer blocks.
- Global procedures: WRITEI, WRITEC, WRITELN

List of error codes

- ERR_UNDECLARED_IDENT
- ERR_UNDECLARED_CONSTANT
- ERR_UNDECLARED_TYPE
- ERR_UNDECLARED_VARIABLE
- ERR_UNDECLARED_FUNCTION
- ERR_UNDECLARED_PROCEDURE
- ERR_DUPLICATE_IDENT

Project organization

#	Filename	Task
1	Makefile	Project
2	scanner.c, scanner.h	Token reader
3	reader.h, reader.c	Read character from source file
4	charcode.h, charcode.c	Classify character
5	token.h, token.c	Recognize and classify token, keywords
6	error.h, error.c	Manage error types and messages
7	parser.c, parser.h	Parse programming structure
8	debug.c, debug.h	Debugging
9	symtab.c symtab.h	Symbol table construction
10	semantics.c. semantics.h	Analyse the program's semantic
11	main.c	Main program

Assignment 2

- Implement the following function in *semantics.c*
  - `checkFreshIdent`
  - `checkDeclaredIdent`
  - `checkDeclaredConstant`
  - `checkDeclaredType`
  - `checkDeclaredVariable`
  - `checkDeclaredProcedure`
  - `checkDeclaredLValueIdent`
- Test on provided examples