Experiment in Compiler Construction Semantic Analysis Type Checking

ONE LOVE. ONE FUTURE.

Overview

- Problem of type checking
- Type consistency in statements
- Checking the consistency between the declaration and usage of arrays.
- Checking the consistency between the declaration and usage of functions.
- Checking the consistency between the declaration and calling of procedures.
- Checking the consistency in reference usage

Type checking

- Check the *Type Rules* of the language.
- Information about *Data Types* is maintained and computed by the **compiler**.
- The module of a **compiler** devoted to type checking tasks is *Type Checker*.
- The design of a *Type Checker* depends on the *syntactic structure* of language constructs, the *Type Expressions* of the language, and the rules for *assigning types to constructs*.

Structure for types

```
struct Type_ {
  enum TypeClass typeClass;
  int arraySize;
  struct Type_ *elementType;
};

enum TypeClass {
  TP_INT,
  TP_CHAR,
  TP_ARRAY
};
```

Type checking in constant declaration

- Constant:
  - [+/-] <constant>
  - The type of <constant> is integer

Type checking in assign statement

- Assign statement
  - <LValue> := <Expr>;
  - Basic types of <Lvalue> and <Expr> must be the same
  - Depend on object kind of Lvalue, function compileLvalue return its data type
  - Type of the expression is evaluated on parse tree whose leaves are factors

Type of a factor

```
case TK_NUMBER:
    eat(TK_NUMBER);
    type = intType;
    break;
  case TK_CHAR:
    eat(TK_CHAR);
    type = charType;
    break;
```

Type attribute of a factor (represented by an identifier)

```
case OBJ_CONSTANT:// Factor is a constant
    switch (obj->constAttrs->value->type) {
    case TP_INT:
      type = intType;
      break;
    case TP_CHAR:
      type = charType;
      break;
```

Type attribute of a factor (represented by an identifier)

```
case OBJ_VARIABLE://single or subscripted
  if (obj->varAttrs->type->typeClass == TP_ARRAY)
     type = compileIndexes(obj->varAttrs->type);
  else
     type = obj->varAttrs->type;
  break;
```

Type attribute of a factor (represented by an identifier)

```
case OBJ_PARAMETER:// Factor is a parameter
  type = obj->paramAttrs->type;
  break;
case OBJ_FUNCTION:// Factor is a function
  compileArguments(obj->funcAttrs->paramList);
  //check type consistency between list of parameters
  //and list of arguments
  type = obj->funcAttrs->returnType;
  break;
```

Type checking for For statement

- For <var> := <exp1> To <exp2> do <stmt>
- Basic types of <var>, <exp1>, and <exp2> must be the same

Type checking

- Function and procedure:
  - Types of declared parameter and actual parameter (argument)must be the same
  - The corresponding actual parameter (argument) of call by reference parameter must be a LValue.

Type checking for conditions

- <exp1> <op> <exp2>
- The basic types of <exp1> and <exp2> must be the same

Type checking for indexes

- (. <exp> .) → <exp> : integer
- The number of dimensions of the array must be considered

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

Functions for type checking

- Type comparison
  - checkIntType
  - checkCharType
  - checkArrayType
  - checkTypeEquality

Assignment 4

- Update *parser.c* with the implementation of described type checking rules
- Test on provided examples