编译系统第二次实验报告

120L021310 王耀宁

1 程序功能及其实现

本程序完成编译器的语义分析部分。更具体地，给定源程序，在词法分析和语法分析的基础上，本次实验检查程序中的语义错误。下面对功能的实现进行说明。

1.1 语义分析

语义分析需要对源程序的变量定义、调用等方面进行检查，确保其符合源语言的标准。本次实验共定义了 17 类错误，对于这些错误的说明在此省略。错误可大致分为 (1) 未定义错误;(2) 重复声明错误;(3) 分量类型不匹配;(4) 操作符和操作数不匹配错误。为了检查未定义错误和重复声明，我们需要实现符号表，记录已经声明的变量；为了检查匹配错误，我们需要在产生式的符号上附加属性，在语法分析过程中计算这些属性并检查。

1.1.1 符号表

我们首先实现符号表。符号表的每个节点声明如下：

```
1  enum item_type { VAR, FUN };
2
3  struct table_item {
4    enum item_type type;
5    char *symbol_name;
6    union {
7      struct {
8        struct struct_info *vtype;
9        struct listnode *shape;
10     } var_info;
11     struct fun_info *fun_info;
12   } data;
13 };
```

其中，type 为该符号的类型，可以为 VAR(变量定义) 或 FUN(函数定义)。symbol_name 为符号名；联合 data 域中，var_info 和 fun_info 分别为变量对应的信息和函数对应的信息。为了使结构体和基本类型 (int, float) 统一，变量的类型统一用 struct_info 指针进行保存；其形状 (即维度) 用一个链表 shape 进行保存。函数的信息由 fun_info 进行保存。各个符号利用链表进行保存（为了节省篇幅，其余结构体定义在此不给出，代码中有对这些

结构体的注释说明)。下面，我们以错误类型 1 的检查为例，说明符号表的使用。在变量定义时，我们在其语义动作中将该变量名称、形状、类型等信息存入符号表：

```
insert_var(table, /* 符号表 */
  ptr->value, /* 用全局链表保存一系列定义中变量的名称 */
  find_struct_by_name(struct_table, type), /* 给定类型名称type，返回其定义*/
  ptr2->value); /* 用全局的链表保存一系列定义中变量的形状 */
```

之后，在变量调用处我们就可以检查符号表中是否有该变量的定义:

```
Exp: ID {
  if (has_item(table, $1) == 0) {
    report_error("Variable \"%s\" undeclared", @1.first_line, "1", $1);
    // ...
  }
}
```

1.1.2 节点属性的计算

下面我们以错误类型 7 为例，说明如何利用综合属性和继承属性检查语义错误。对于 Exp(表达式) 节点，我们需要维护它的维度信息和类型信息，以检查运算和赋值的合法性。在 Bison 中，我的表达式属性定义如下：

```
%union {
  //...
  struct {
    struct struct_info *type; // 类型
    int dim; // 维度
    int assignable; // 能否被赋值
  } exp;
}
%type <exp> Exp
```

由于在本实验中，只要变量的维度相同，就认为它们的类型相同 (如，a[3][5] 和 b[5][3] 认为是相同类型的)，所以只需要维护维度即可。同时，我们用 assignable 属性维护左值/右值。以 **Exp → LB Exp RB** 一条产生式为例，属性的计算如下：

```
$$.type = $1.type; // 基本类型相同
$$.dim = $1.dim - 1; // 维度-1
$$.assignable = $1.assignable; // 若未索引之前能赋值，索引之后也能赋值
```

之后，在两个表达式相加时，即可判断是否出现不匹配问题：

```
Exp: Exp PLUS Exp {
  if ($1.type != $3.type) { // 类型不同
    report_error("Operation type mismatch: +", @1.first_line, "7", $1.type,
        $3.type);
  }
  else if (!($1.type <= FLOAT && $1.dim == $3.dim && $1.dim == 0)) {
    // 维度不同
    report_error("Operation type mismatch", @1.first_line, "7");
  }
  // ...
}
```

1.2 编译方式

本实验采用 Makefile 进行编译。Makefile 依次调用 bison,flex 和 gcc 的组件 cc 进行编译。因此，源码可以通过

```
make
```

等价于

```
cc: cc.l cc.y cc.h ccutils.c symbol_table.c
bison -d cc.y
flex -o cc.lex.c cc.l
cc -o $@ -w cc.tab.c cc.lex.c symbol_table.c ccutils.c
```

进行编译；编译后，会生成 cc 可执行文件，它的调用方法如下所示：

```
./cc [test_dir] [to_file]
```

其中 test_dir 为测试文件目录，to_file 是一个可选参数，可以是 1 或不填写，若 to_file=1，结果会被输出到 result.txt 文件中。以下调用都是合法的：

```
./cc ./test/
./cc ./test/ 1
```