Step 9 Report

刘轩奇 2018011025

2020年10月7日

1 工作内容

本人选择不使用辅助工具 ANTLR 因此自己实现了 lexer 和 parser。

1.1 文件说明

`montLexer.h/cpp` 词法分析器；

`montParser.h/cpp` 语法分析器；

`montConceiver.h/cpp` 产生中间代码；

`montAssembler.h/cpp` 从中间代码产生汇编代码；

`montLog.h` 记录编译错误信息；

`montCompiler.cpp` MiniDecaf 编译器，包含主函数，编译成功返回 0 否则返回 -1，并将错误信息输出到 `std::cerr`。

1.2 本步骤完成的工作

1 词法分析 添加了新的 Token：

* `BOOL` 关键字 bool；
* `VOID` 关键字 void；
* `COMMA` 逗号。
* `TRUE` 关键字 true；
* `FALSE` 关键字 false。

2 句法分析 添加或修改了以下的 AST 节点：

* `parameters` 等同于实验指导书上的 `parameter_list`。
* `exprlist` 等同于实验指导书上的 `expression_list`。
* `unary`, `postfix` 同实验指导书。
* `function` 函数节点，支持 `void`, `int`, `bool`, `char` 返回值，支持 `int`, `bool`, `char` 参数类型。

3 产生中间代码 在此阶段，遍历语法树过程中也判断每一个树节点的数据类型，例如整数数值节点的数据类型为整数，整数加法节点的类型为整数，逻辑运算节点的类型为布尔类型，等等。据此得到的结果来判断函数调用等是否符合语法。另外也添加了类型转换（主要是整数转换为布尔类型）的支持。

添加了中间代码 `CALL` 并修改了 `RET` 使之对应于函数调用。其中 `CALL` 包含一个字符串参数表示调用函数名，一个数字参数表示函数参数个数。另外添加了中间代码 `CALLV`，`RETV` 与 `void` 类型函数相对应。

4 产生汇编代码 同指导书上说明，不再赘述。

2 思考题

1 MiniDecaf 的函数调用时参数求值的顺序是未定义行为。试写出一段 MiniDecaf 代码，使得不同的参数求值顺序会导致不同的返回结果。

答

```
int add(int a, int b){
    return a+b;
}

int main(){
    int a = 1;
    return add(a=a+1, a=a*2);
}
```