Step 1 Report

刘轩奇 2018011025

2020年9月25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 在 `MontLexer` 中，定义了 Token 的分类，所有不同的关键词 /保留字被看作不同的 Token，这样可以减少后续语法分析的工作量。其余 Token 类型还包括标识符 Identifier, 各种符号，整数值，字符值等。`MontLexer` 运行过程类似于有限状态自动机，每读入一个字符后，判断当前读入的这个 Token 是否有可能是符号、标识符、关键字、常数值中的任意一种，并判断是否已经读完一整个 Token ，若是，则返回此 Token 作为 nextToken()。在这一步中也判断了常数值过大的词法错误。

2 句法分析 在 `MontParser` 中，只定义了两种树节点： `MontNode` 和 `MontTokenNode`，分别表示内部节点和终结符节点。所有的内部节点由其类型标识，例如函数节点、语句节点、表达式节点等，有至少一个子节点；而终结符节点包含且仅包含一个 Token，不含子节点。抽象语法树生成过程中，调用各种 `tryParse...()` 方法来产生对应的树节点。例如：对于产生式

function : type Identifier LParen RParen codeblock

将会分别调用 `tryParseType()`, `tryParse(TK_IDENTIFIER)`, `tryParse(TK_LPAREN)`, `tryParse(TK_RPAREN)`, `tryParseCodeblock()` 来生成子节点。

3 产生中间代码 在 `MontConceiver` 中，按照指导书上的说明，将 `value` 节点生成 `PUSH` 语句（根据产生式 `expression : value`），将 `return` 节点生成 `RET` 语句。在这一步中也检查了函数名是否为 `main`。

4 产生汇编代码 在 MontAssembler 中，按照指导书的说明，将对应中间代码转换为对应汇编语句，不再赘述。

5 编译错误行号追踪 在词法分析、语法分析和生成中间代码的过程中，如若遇到编译错误，则在 std::cerr 中输出编译错误信息。同时，由于在 lexer 读入和 Token 结构中记录了行列号，则输出编译错误信息时也可以同时输出，便于用户查错。

6 尚未完整实现或当前尚未使用的功能 支持以 0x... 十六进制形式读入整数常量；支持读入 char 类型作为常量，虽然存储上依然按照整数（即ASCII编码）存储，支持普通显示字符（例如 'A'）、转义字符（例如 '
A'）、八进制表示转义字符（例如 '
123'）、十六进制转义字符（例如 '
x41'）；在 lexer 和 parser 阶段支持单变量定义语句（例如 int a;）。

2 思考题

1 修改 minilexer 的输入（lexer.setInput 的参数），使得 lex 报错，给出一个简短的例子。

答 只需要出现非法字符即可。例如 int main(){return 一百二十三;}

2 修改 minilexer 的输入，使得 lex 不报错但 parse 报错，给出一个简短的例子。

答 只需要出现语法错误即可。例如 int main(){turn 123;}

3 在 riscv 中，哪个寄存器是用来存储函数返回值的？

答 a0 寄存器。