|  |
| --- |
| sjtu |
| 编译课程设计报告 |
| 基于Tiger语言 |
|  |
| **姓名：陈斌**  **学号：5080309505**  **日期：2011年1月18日** |
|  |

|  |
| --- |
|  |

目录

**前言2**

**概述3**

**词法分析4**

**语法分析**6

**语义分析（类型检查）**9

# 前言

本课程设计主要参考《Modern Compiler Implementation in Java》一书以及网站<http://www.cs.princeton.edu的关于Tiger>语言编译描述，JFlex以及cup文档。基于Tiger语言的编译器较其他语言简单，并且使用我们熟悉的java语言编写，有利于学习编译器。

由于时间以及能力有限，报告中有错漏请见谅。

感谢助教的入门指导以及在次过程中帮忙解决问题的各位同学。

陈斌

2011年1月8日

# 概述

目标：

实现Tiger语言编译器设计

步骤：

1.词法分析(Lexical Analysis) :把Tiger 源程序分割成符号(单词),要求制作自动机文件，词法分析程序由工具JFlex 生成

2.语法分析(Syntax Parsing) :识别程序的语法结构,检查语法错误,要求制作文法文件,语法分析程序由工具Java Cup 生成

3.抽象语法 (Abstract Syntax Tree) :根据语法结构生成抽象语法

4.语义分析(Semantic Analysis) :进行变量和类型检查等

5.活动记录 (Activation Record) :与函数调用相关的活动记录

6.中间代码生成(Intermediate Code):生成中间表示 (IR Trees)

7.规范化(Canonicalize) :优化表达式、条件分支等,只需复制代码

8.指令选择 (Instruction Selection): 生成基本的MIPS 汇编指令

9.活性分析与寄存器分配 (Liveness Analysis and Register Allocation)

10.使之成为整体:生成完整的编译器程序

# 词法分析

编写JFlex或者Jlex文件，由工具自动生成java文件加入工程，能够分析源程序分割出一个个Token，提供给后续的语法分析。

工具使用：使用JFlex工具运行bat导入jflex文件后自动生成Yylex.java

生成Token规则

|  |  |  |  |
| --- | --- | --- | --- |
| 形式 | 词法规则 | 分析 | 相关代码 |
| Identifier | 以字母开头，数字、字母、下划线组成。区分大小写 | 在YYINITIAL状态中返回对应id | alpha= [A-Za-z]  digit= [0-9]  id={alpha}({alpha}|{digit}|\_)\*  <YYINITIAL> {id}{return tok(sym.ID,yytext());} |
| Comment | 包含在/\*和\*/中 | "/\*": 进入COMMENT 状态， "\*/":返回YYINITIAL 状态 | <YYINITIAL> "/\*"{count=1;yybegin(COMMENT);}  <COMMENT> {"/\*"{count++;}  "\*/" {count--; if (count==0) {yybegin(YYINITIAL);}}  [^] {} } |
| Whitespace | 空格 制表符 换行符 回车符 分页符 | 不做操作 | delim= [ \t\n\f\r]  ws = {delim}+  <YYINITIAL> {ws} {} |
| String | 包括在”之间 | “进入STRING状态，”回到YYINITIAL，回车分页异常 | <YYINITIAL> \" {string.setLength(0);yybegin(STRING);} |
| Escape sequence | \n | 在STRING状态内出现，遇到\进入INSLASH状态 | control = [\\^][@A-Z[\\]^\_]  [\t\\]+{string.append(yytext());}  \\n{string.append('\n');}  \\t{string.append('\t');}  \\\" {string.append('\"');}  \\\\{string.append('\\');}  [\n\f] {err("String error");}  [ ] {string.append(' ');}  control{string.append(yytext());}  \\{yybegin(INSLASH);} |
| \t |
| \” |
| \\ |
| \^c |
| \ddd |
| \...\ |
| Punctuation Symbol | , : ; ( ) [ ] { } . + - \* / = <> < <= > >= & | := | 匹配则返回sym中相应符号 | <YYINITIAL> "-"  {return tok(sym.MINUS,null);} |

要点

1. 在编写JFlex文件中要注意编写的顺序，这个会涉及到优先级问题，比如id应该放在最后。
2. 在转义字符\ddd中用Integer.parseInt强制把ddd转换成整型数，检查是否在0~255之间，(char)返回String，否则抛出错误。
3. 在Yylex中调用ErrorMsg.ErrorMsg

遇到问题

本部分在一开始就遇到拦路虎，由于本机java环境安装在D:\Program Files (x86)\Java内，在更改JFlex bat中JAVA\_HOME路径无论如何都无法使用上JFlex工具，最后在终于找到原因路径中不能有空格，负责需要以双引号括起来。

在一开始没有仔细看tiger manual时候编写jflex根据平时碰到java或者C++语言的规则，比如复制使用= 判断相等使用了==，后来细看才发现manual中给了详细的说明是:= 和=，这是个教训。

在Comment中使用的是一个变量记住遇到的/\*和\*/，这是词法分析中唯一有点技术含量的地方，不过似乎大家大同小异。

ID 82

ASSIGN 83

INT 85

TO 88

STRING 93

DO 95

ID 101

ASSIGN 103

ID 106

MINUS 108

INT 110

EOF 113

# 语法分析

编写cup文件生成java文件加入工程中，在工程中调用，利用tiger语言的语法规则来消除左递归，给出调用的顺序，并且输入抽象语法树。

使用cup工具导入cup文件生成Parser.java

基本框架

Symbol.Symbol提供符号,Symbol.Table提供符号表

Dec: 类型、变量、函数声明

DecList:声明块结构

Exp:表达式

FieldExpList:用逗号分割的表达式列表，用于函数调用

Var:简单变量，域变量，下标变量

ExpList:用分号分割的表达式列表

Ty ：数组、记录、自定义类型

FiledList: 域列表，用于记录声明或函数声明

语法数结点定义

|  |  |  |
| --- | --- | --- |
|  | 语法类型 | 名字 |
| 终  结  符  号 | String | ID，STRING |
| Integer | INT |
|  | US,MINUS,UMINUS,TIMES,DIVIDE,ASSIGN,Dec,EQ, NEQ, LT, LE, GT, GE, AND, OR, ARRAY, OF, VAR, TYPE, DOT, NIL, FUNCTION, IF, THEN, ELSE, WHILE, FOR, TO, DO, LET, IN, END, BREAK, COMMA, COLON, SEMICOLON, LPAREN, RPAREN, LBRACK, RBRACK, LBRACE, RBRACE |
| 非  终  结  符  号 | Exp | expr |
| ExpList | expr\_list |
| FieldList | type\_field,type\_fields |
| FieldExpLlist | field\_list |
| Dec | declaration |
| DecList | declaration\_list |
| VarDec | variable\_declaration |
| Var | lvalue |
| TypeDec | type\_declaration |
| Ty | type |
| FunctionDec | Function\_declaration,function\_declarations |
| SeqExp | expr\_seq |

语法树节点翻译

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **Tiger语言** | **形式** | **Absyn结点** |
| expr | string | STRING: s | StringExp( s ) |
| integer | INT: i | IntExp( i.intValue() ) |
| lvalue | lvalue: lv | VarExp( lv) |
| - expr | MINUS: m exp:e | OpExp( IntExp(0), MINUS, e ) |
| expr binary-operator expr | exp: e1 BINOP2 exp: e2 | OpExp( e1, BINOP, e2 ) |
| lvalue := expr | lvalue: l ASSIGN exp: v | AssignExp( l, v ) |
| id ( expr-list) | ID:I LPAREN explist:elist RPAREN | CallExp( symbol(i), elist ) |
| ( expr-seq ) | LPAREN:l explist:elist RPAREN | SeqExp( elist ) |
| ( ) | LPAREN:l RPAREN | SeqExp( null ) |
| type-id { field-list } | ID:i LBRACE TypeList:flist RBRACE | RecordExp( symbol(i), flist ) |
| type-id { } | ID:i LBRACE RBRACE | RecordExp( symbol(i), null ) |
| type-id [ expr ] of expr | ID:i LBRACK exp:type LBRACK OF exp:value | ArrayExp( symbol(i), type, value ) |
| if expr then expr | IF:i exp:cond THEN exp:true | IfExp( cond, true, null ) |
| if expr then else expr | IF:I exp:cond THEN exp:true ELSE exp:false | IfExp( cond, true, false ) |
| while expr do expr | WHILE:i exp:cond DO exp:body | WhileExp( cond, body ) |
| for id := expr to expr do expr | FOR:f ID:i ASSIGN exp:low TO exp:high DO exp:body | ForExp( symbol(i), low, high, body) |
| break | BREAK: i | BreakExp( ) |
| let declaration-list in expr-seqopt end | LET:i declarationlist:declist IN expseq:eseq END | LetExp( declist, eseq ) |
| expr-list | expr | exp: e | ExpList( e, null ) |
| expr, expr-list | exp: e COMMA explist: elist | ExpList( e, elist ) |
| expr-seq | expr | exp: e |  |
| expr; expr-seq | exp: e SEMICOLON epseq: eseq | SeqExp( e, eseq ) |
| field-list | id = expr | ID:i EQ exp: type | FiledExpList( symbol(i), type, null ) |
| id = expr, field-list | ID:i EQ exp: type COMMA TypeList: flist | FiledExpList( symbol(i), type, flist ) |
| lvalue | id | ID:i | SimpleVar( symbol(i) ); |
| id . lvalue | ID:i DOT lvalue: l | FiledVar( symbol(i), l ) |
| lvalue [ expr ] | ID: i LBRACK exp:e RBRACK | SubscriptVar( symbol(i), e ) |
| declaration-  list | declaration | declaration: dec | DecList( dec, null ) |
| declaration, declaration-list | declaration: dec declarationlist: declist | DecList( dec, declist ) |
| declaration | type-declaration | typedeclist: declist | declist |
| variable-declaration | vardec: dec | vardec |
| function-declaration | funcdeclist: declist | declist |
| type-declar  ation | type type-id = type | Type:i ID:id EQ type: t | TypeDec( symbol(id), t ) |
| type | type-id | ID: i | NameTy( symbol(i) ) |
| { type-fields } | LBRACE:i typelist:tlist RBRACE | RecordTy( tlist ) |
| { } | LBRACE:i RBRACE | RecordTy( null ) |
| array of type-id | ARRAY:i OF ID:i | ArrayTy( i ) |
| type-fields | id: type | ID:i COLON ID:t | FiledList( symbol(i), symbol(t), null ) |
| id: type, type-fields | ID:i COLON ID:t COMMA typelist:tlist | FieldList( symbol(i), symbol(t), tlist ) |
| variable-dec  laration | var id := expr | VAR:i ID:id ASSIGN exp: value | VarDec( symbol(id), null, value ) |
| var id : type-id := expr | VAR:i ID:id COMMA ID:tyASSIGN exp:value | VarDec( symbol(id), NameTy(symbol(ty)), vallue ) |
| function-d  claration | function id(type-fields) = expr | FUNCTION:f ID:i LPAREN typelist:list RPAREN EQ exp:v | FunctionDec( symbol(i), list, null, v ) |
| function id( ) = expr | FUNCTION:f ID:i LPAREN RPAREN EQ exp:v | FunctionDec( symbol(i), null, null, v ) |
| function id(type-fields):type-id = expr | FUNCTION:f ID:I LPAREN typelist:list RPAREN COLON  ID:t EQ exp:v | FunctionDec( symbol(i),list,NameTy(symbol(t)), v ) |
| function id( ) :type-id = expr | FUNCTION:f ID:i LPAREN RPAREN COLON ID:t EQ exp:v | FunctionDec( symbol(i),null,NameTy(symbol(t)),  v ) |

要点：

在parse中定义parser parser = new parser(new Yylex(inp,errorMsg), errorMsg);在Yylex词法分析后调用语法分析。调用Absyn.Print打印出抽象语法树。Cup文件按的函数声明中注意有关于是否有参数，是否有返回的四种情况。定义负数使用的是0减去正整数。

遇到问题:

使用java-cup-11a工具生成的java不能和教材中给的框架匹配，所以在询问同学发现使用10版本没有问题，故改用java-cup-10。

ForExp(

VarDec(i,

IntExp(10),

true),

StringExp( ),

AssignExp(

SimpleVar(i),

OpExp(

MINUS,

varExp(

SimpleVar(i)),

IntExp(1))))FOR 78

# 语义分析（类型检查）

基本框架：

Env.Env 储存符号、变量和函数。

Env.tEnv<Symbol.Symbol, Type.Type>: 符号-类型表。

Env.vEnv<Symbol.Symbol, Semant.VarEntry>, <Symbol.Symbol, Semant.FuncEntry>: 符号-变量函数表。

语法抽象树的节点传入 Semant.Semant, 由 Semant.Semant 对其进行类型检查，并进行相关的类型翻译。

类型检查规则

|  |  |  |  |
| --- | --- | --- | --- |
| 调用方法 | 传入参数 | 类型检查 | 返回值 |
| transExp(e) | IntExp(value) | 词法分析已经检查 | INT |
| StringExp(value) | 同上 | STRING |
| NilExp( ) | 同上 | NIL |
| VarExp(var) | 同上 | VAR类型 |
| OpExp(left, right, oper) | 对于= !=不能为void  <<= >=必须为String或者INT | INT |
| CallExp(func, args) | 获得func参数列表fmls  检查args与fmls是否匹配   1. args.head不为void 2. args.head能够强制转换为fmls.head 3. args和fmls结点个数一样 | 对应函数的返回类型 |
| RecordExp(fields, typ) | 获取 RECORD入口 typ的参数列表 fmls  逐个检查传入参数列表  1、fields与 fmls是否匹配  fields.head.name 等于 fmls.name  2、fields.head 类型能强制转换为 fmls.name 类型  3、fields和 fmls的节点个数相同 | typ 对应的类型 |
| SeqExp(list) | list 为空，无需检查  list 非空，对 list 的翻译已保证正确性，无需检查 | VOID  list 最后一个节点的类型 |
| AssignExp(var, exp) | exp 类型不能为 VOID  exp 类型能强制转换为 var类型 | VOID |
| IfExp(test, thenclause, elseclause) | test 为INT  thenclause 与elseclause类型一样 | VOID  对应的 RECORD类型  thenclause 的类型 |
| WhileExp(test, body) | 循环条件 test类型为 INT  循环体 body类型为 VOID |  |
| BreakExp() | 设置全局变量 loops录循环嵌套数  翻译 ForExp, WhileExp时 loops++  翻译完成时 loops--  loops不能为0 | VOID |
| ForExp(var, hi, body) | 循环变量初始值 var.init类型为 INT  循环变量上限 hi类型为 INT  循环变量作用域为 ForExp，使用 beginScope()标记新域  循环体 body类型应为 VOID  循环结束退出使用 endScope()退出当前域 | VOID |
| LetEp(decs, body) | 在 decs里申明的变量作用域为 LetExp  使用符号表的 beginScode()标记新域  对 decs和 body(可为空)的翻译已保证正确性，无需检查  body翻译结束调用 endScope()退出当前域 | body为空: VOID  body非空: body的类型 |
| ArrayExp(size, init, typ) | 符号 typ属于 ARRAY类型入口  获取 ARRAY入口 typ的参数类型 element  数组大小 size 类型为 INT  数组初始化 init类型能强制转换为 element | typ 对应的类型 |
| transVar(v) | SimpleVar(name) | 符号 name 属于类型入口 | name对应的类型 |
| FieldVar(var, field) | var 类型为 RECORD  符号 field应为对应RECORD类型的参数符号之一 |  |
| SubscriptVar(var, index) | var 类型为 ARRAY  index 类型为 INT |  |
| transDec(d) | FunctionDec |  |  |
|  | VarDec(name, typ, init) | init的类型能强制转化成 typ对应的类型 | null |
| transTy(t) | NameTy(e) | 符号 e属于类型入口 | e 对应的类型 |
| ArrayTy(typ) | 符号 typ属于类型入口 | ARRAY(typ 对应的类型) |
| Record(fields) | fields为空，无需检查  fields非空，对 fields的翻译已保证正确性，无需检查 | fields的类型 |
| transExp(e) | FieldList(name, typ, tail) | 设立集合 set存储 RECORD的参数名  符号 name不在 Set中  符号 typ属于类型入口  转化成相应的 RECORD节点 | 转化后的 RECORD类型 |

要点：

上表是结合tiger manual考虑到的类型检查，可能有错漏或者不符合的地方，尚有待完善。Tiger manual中定义的11类标准func在Env中加入

遇到问题：

到了这部分需要自己大量书写代码了，一开始是一片茫然的，后面仔细阅读了虎书以及向同学请教，才慢慢开始写起来，断断续续写得非常慢，由于时间以及个人能力关系，所以类型检查这部分不能算是全部完成，调试还没有成功，比较遗憾。对此自己深感愧疚。

最后一些话：

由于刚刚做完必修的部分，其他后续部分还没有完成，没有一个属于自己的完整的编译器，同时看到一些同学的进度让我自己都感到不好意思。希望自己在后面能够继续完成，与分数无关，起码能够收获到一份喜悦。相信每一位编程者都会对一个完成的工程感到很高兴。

感谢张老师在前面半个学期中给我们教的理论知识，没有这些知识一切都是空谈，感谢助教的评测指导，你们都辛苦了，感谢身边的同学，无论是深夜写代码影响到的宿舍的哥们，还是一起去通宵写代码的兄弟。虽然写代码是一个人对着电脑，但是没有大家我相信自己都没法坚持下来。

因为报告没有规定形式，所以希望这份报告能够基本符合要求。

陈斌

2010年1月11日