.

Compiler Design \_Moon Compiler

Comp 442

Yifan Yang

40038814

2020/04/17

# Overall structure

Lexical Analysis: DFA / Table Driven / **HandWritten**

Syntatic Analysis**:** Recursive descent predictive parser / **Table-driven predictive parser**

ASTtree**: Supported**

Visitor Pattern: **Supported**

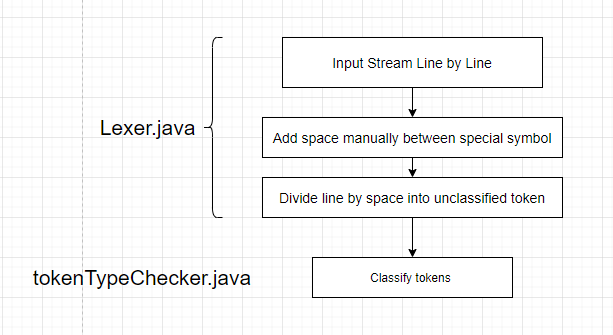
SymbolTable&TypeChecking: syntax-directed translation on parse tree/ **post-order Traverse on ASTtree with visitor mode**

Code Generation: **Tag Based with function implemented**/ stack based

A compiler that compile Moon language, implemented a lexical analyzer, a table-driven predictive parser for syntactic analysis, and then generated an AST tree as the intermediate representation, which generates object code directly from the AST tree at the end.

\\

# Lexical Analysis:



Key point 1: Add Space manually

Basic Logic

AAA\*BB

intputString.split(x);

AAA**|**BB

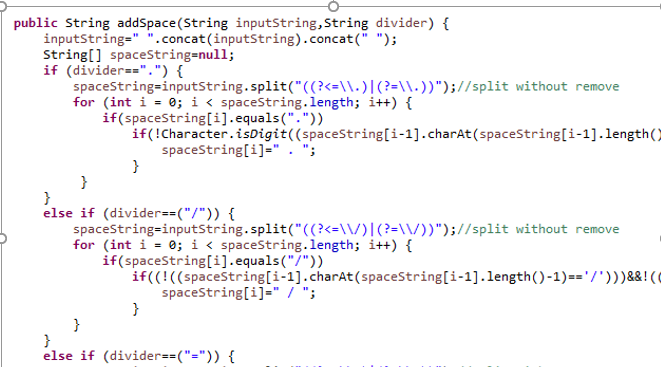
inputString.split("((?<=x)|(?=x)")

AAA**|**\***|**BB

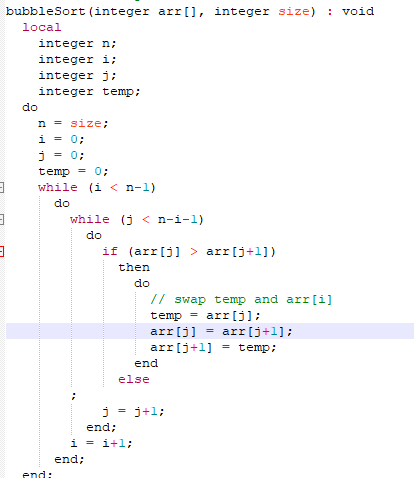
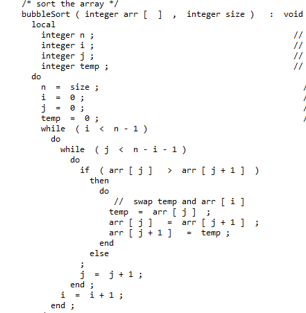
if last a is "/" or first B is "/" ,then it will be AA/\*BB /\* is cmt won't add space

else AAA**|** \* **|**BB

combine string[] to string



Original Code After add space

Key Point 2:Comment

when divide line by space, the structure of arrayList will be

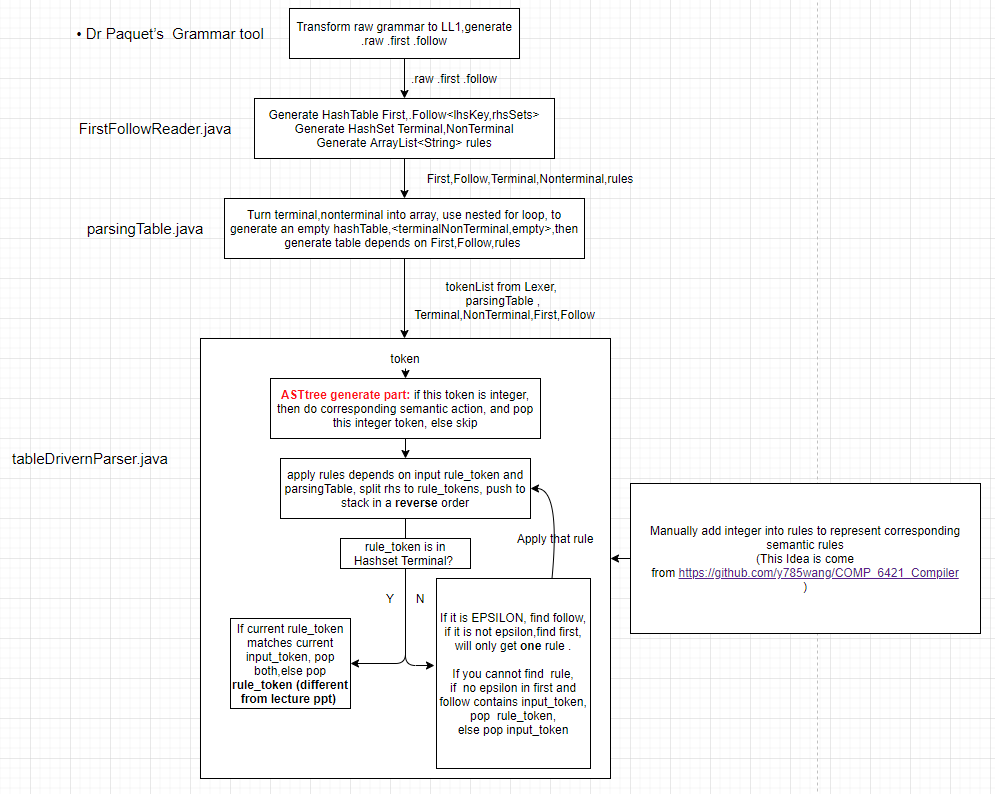
token| line number | token | line number

For inline comment, when meet inline cmt,start ,when line number changes, end

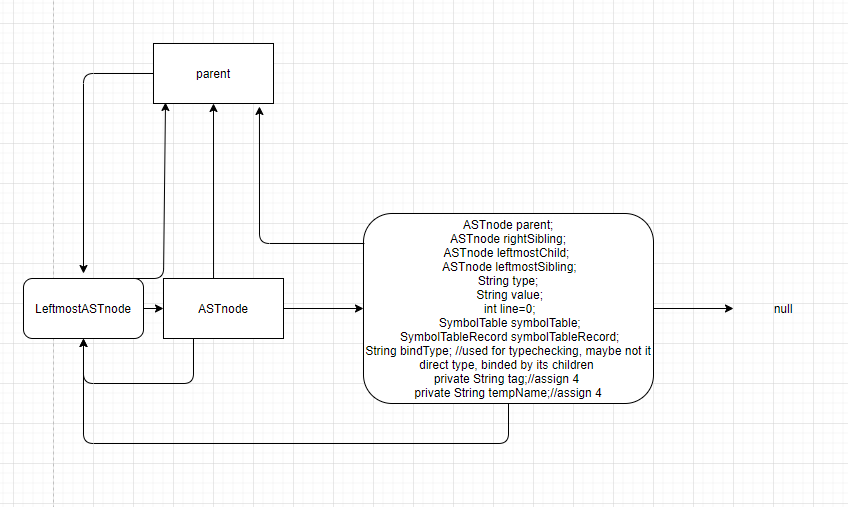
Example : // | 2 | This | 2 | is | 2 | comment | . | 2 | int | 3 | i | 3 .........

For block cmt, when reach /\*, counter ++, when reach \*/ counter--, if counter =0, generate , if to the file end ,counter is still not 0, then it is incomplete comment

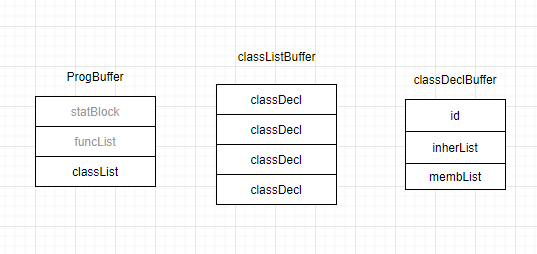
# Syntactic Analysis



ASTNode:



Distributed Buffer post-order-traversal



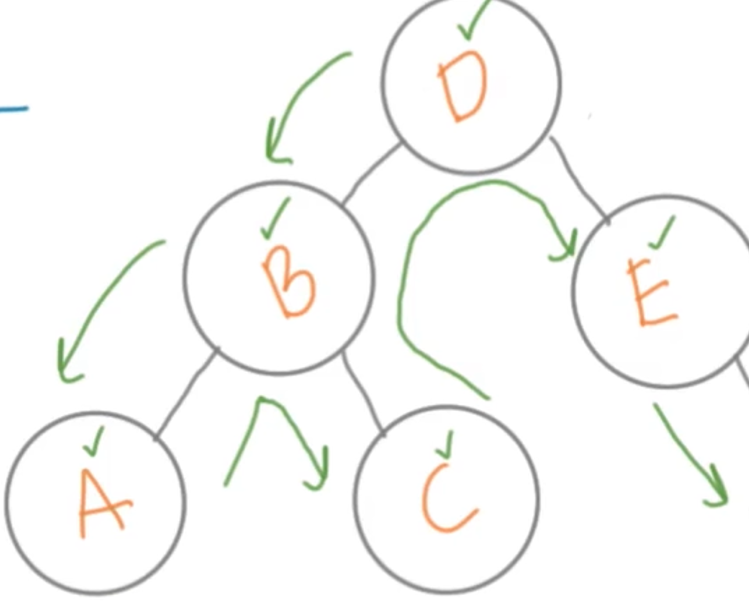
When I meet 5, clear classDeclBuffer to generate classDecl node



when I meet 2 , clear classListBuffer to generate classList (to handle uncertain # child)



So the way to build my tree is post-order traversal



generate A, generate C, clear cache ,generate B

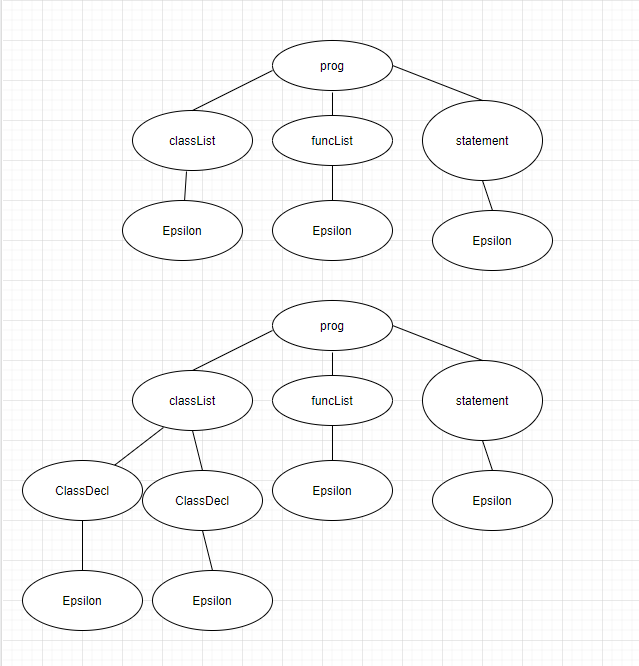
generate E, clear cache(BE), generate D

Advantage :

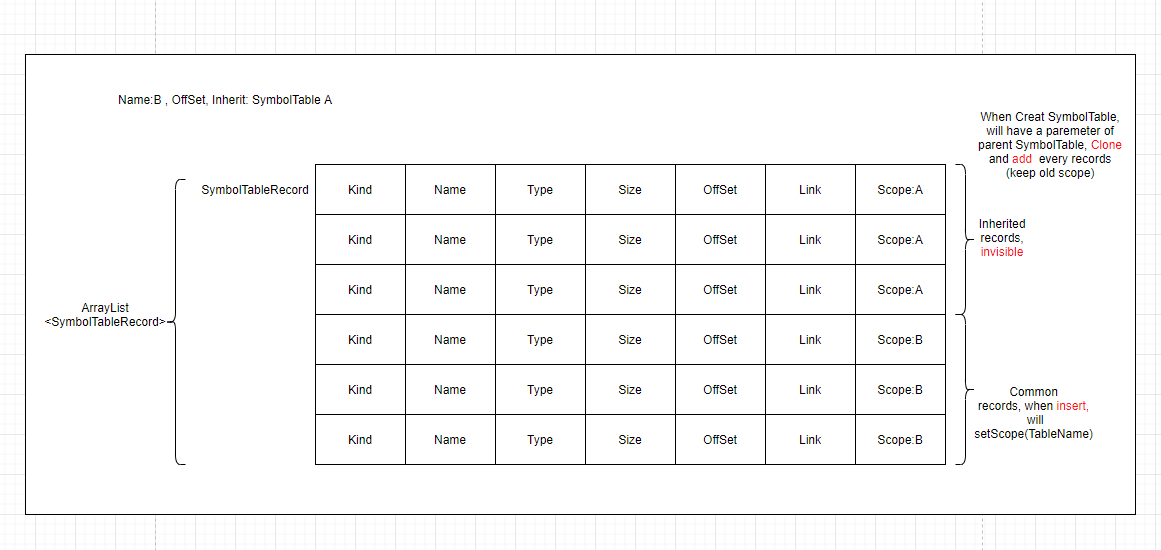
it is distributed so easy to debug even in assignment 3 and 4

It is straightforward so you can build the tree step by step

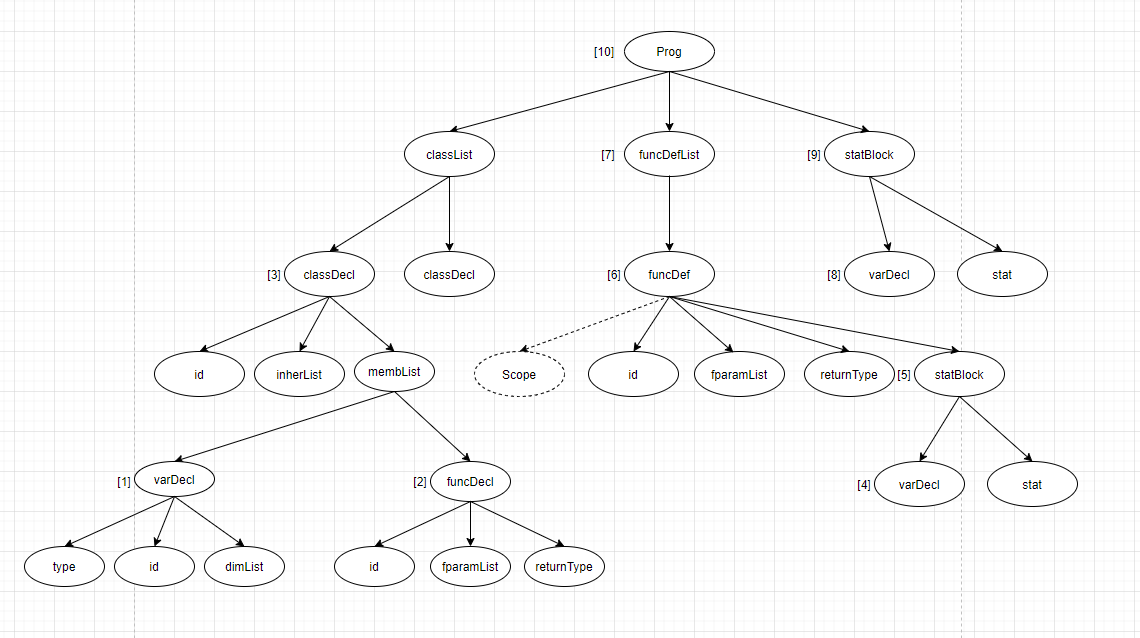
Eg,



By Clear a empty buffer, will generate a node with child EPSILON



SymbolTableCreation Phase1.



SymbolTableRecord (name, kind, type, link)

SymbolTable (name, parent)

[1] Generate var record at vardecl (id,"variable", type ,null)

type=type+dimListAllChildren



for dim, we will remove first char, get second char, so [6] will be recorded as "6", [] will be recorded as "]" , "]" is only allowed in fparamList

[2] node.setTable(name,null), //generate empty table

generate func record at funcDecl(id,"function",type,node.getTable()) //generate record that point to empty table

type=returnType : fparam, fparam,fparam

[3] classDecl

if no inher, generate table (id, null), migrate records from [1] and [2]

if has inher, loop from node.getLeftMostSibling(), if no such parent, Failure.

if can find parent && parentClass.getTable!=null ,create new symboltable and point to parent

if can find parent&&parentClass.getTable==null (Circular dependencies), create new symboltable, point to a new symbolTable with parent name

// then if child class inherit this class, and find its ancester.name= child.name, circular

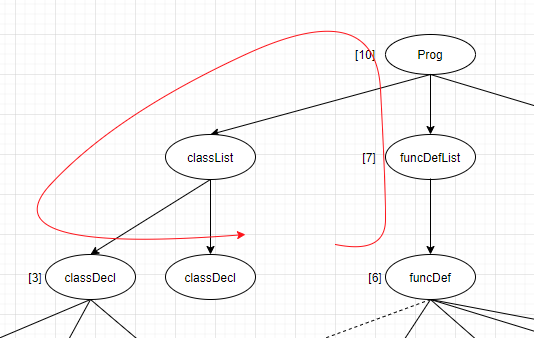
[4] varDecl like [1]

[5] StatBlock

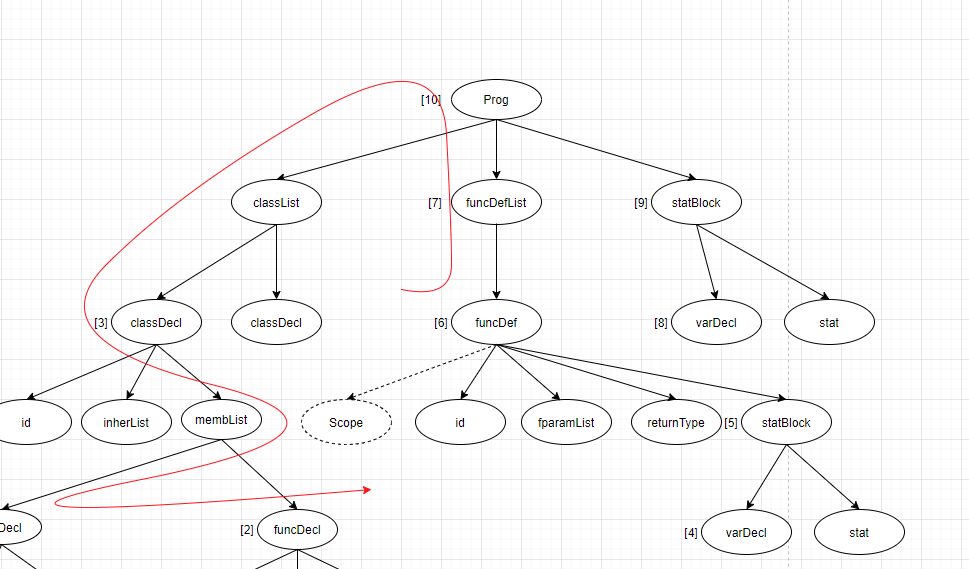
create "main" table, migrate all var records

[6] if fparamList is getChildren(2) : class function

if fparamList is getChildren(1) : free function



for class function, try to find such classDecl, if cannot find such class, failure,



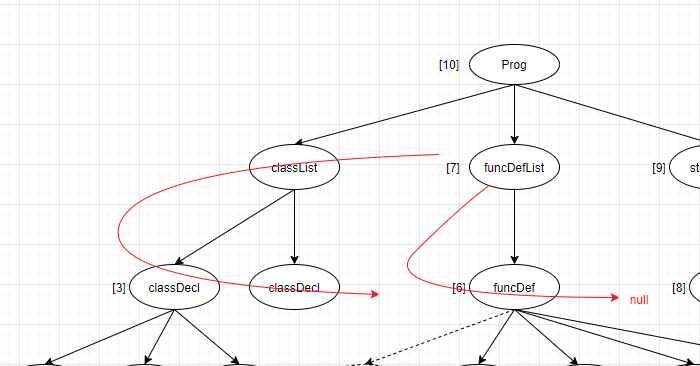
else loop membList, no such funcDecl, failure,

else if you can match the funcDecl, funcDef.setRecord(funcDecl.getRecord()) , cause java is object-oriented, then any operation of funcDef can apply to funcDecl , then we can change funcDecl symbotable, migrate every variable records in statBlock SymbolTable (use insert ,will change scope) to function SymbolTable, migrate every fparamList to function SymbolTable , then delete the statBlock symboltable (only main statBlock won't be deleted)

for free function just migrate every variable record in statBlock SymbolTable to function SymbolTable, delete statBlock symbolTable

[7] funcDefList

cause the visitor is post-order, when we reach funcDefList, all funcDef are done.



funcDefList.getLeftmostSibling.getLeftMostChild.getSymbolTable,

if records.getType=function,

loop every funcDef, if current funcDef=null and still cannot match, then failure

[8] varDecl like [1]

[9] statBlock like [5]

[10] prog

create global symboltable, insert class Records, free func records, main function

SymbolTable Creation Phase 2

when symbol creation is done, add Class Datamember record to its function symboltable, cause function symboltable only have parameters+ local variable.

TypeChecking phase

addOp/multOp/relExpr/assignStat, compare operands, if equal ,

set bindType=leftmostChild.getbindType()

else set bindType= "typeError"

sign/not: migrate up children's bind type, set bindType=leftmostChild.getBindType(). //only one children

There is no real factor/arithExpr/term/expr node

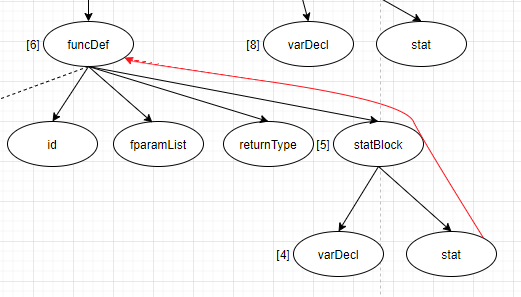
indexList: loop all children, if children.getBindType!=integer, set bindType="typeError"

else set bindType= # of children

returnStat: while (current!=funcDef&&current!=null) current=current.getParent();

till reach funcDef, then compare returnType and node.getLeftmostChild().getBindType()

if current =null, means it is a return stat in mainBlock, failure



intNum,floatNum,var: they are **basics** of typeChecking, any semantic type are from these three elements.

intNum, floatNum, setBindType("integer"), setBindType("float")

var:

case 1: dataMember:

case 2: fCall

case 3: dataMember fCall  
case 4: dataMember dataMember

case1: dataMember

find its classDecl/funcDef/statBlock symbolTable,

if indexList.getBindType=0; search variable/dataMember/parameter/object with same name, setBindType= SymbolTableRecord.getType()

if indexList.getBindType>0,

compare name ,then compare indexLength,

if cannot find such variable/datamember/parameter, failure

if indexList.getBindType==typeError, setBindType(typeError)

case 2: fCall

fCall, firstly search "function" with same name,

if function has no parameter && aParams==null, match, return type

if function has parameter

compare # of parameters , if equal , then compare parameter one by one, if you can reach end, then match, return type . And for indexList, remove integer temprorarily. Eg. a[7] should == a[]

case 3: dataMember.fCall

Firstly, use dataMember node, repeat case[1] (but don't set bindType),should match a not-function record, then getType from this symbolTableRecord, try to match a class, if you can match such Class. then use fCall node. repeat case[2], the final bindType should be function return type

case 4: dataMember.dataMember

use first dataMember , repeat first half of case 3, then use second dataMember, repeat Case[1].

## Code Generation

TagVisitor: post order,

Temp Operator: addOp,multOp,relExpr,not,sign

These operators will generate a tempTag, share a static int counter,

their tag will be, **Scope+"t"+operator+counter,** eg: maintadd1

AfterGenerating the tag, astnode.setTag,

astnode.setRecord(The record won't have )

find scope and insert this record

Common datamember tag:

loop main/func symbolTable ,

Their tag will be ,**Scope+variableName**, eg mainX

funcTag:

when funcDef tag

name+parameter

eg:



multintegerinteger

TableSizeVisitor:post-order

//It can compute size for array(float/int), so I can allocate memory for simple array, but I cannot use them

addTableSize(SymbolTable):compute every record size

Size=basicSize(integer4，float8)\* dimensions

addOffSet(SymbolTable):compute offset of every record

offset=accumulator

accumulator-=record.size

CodeGeneratorVisitor ： This visitor is **not** post-order, neet to let children accept visitor manuially.

statBlock:

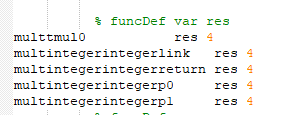
funcDef:

loopTable, allocate memory for every variable's tag

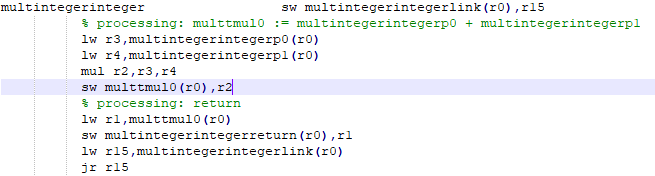
allocate memory for function link

allocate memory for function return (only int, limitation)

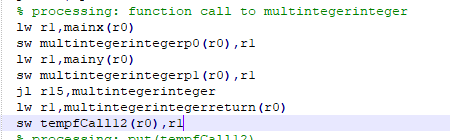
allocate memory for function parameter: parameter tag will be generated here, **functionTag+p+counter .** Because tag parameter is based on function parameter, and tagVisitor is postOrder.



generate statBody



funcCall:when use a fcall, create a tempTag to save return value, (tempfCall+counter)



Tool Used:

RegEx simulator: https://regexr.com/

used to check regular expression, help java find the type of string

RegEx->DFA: https://cyberzhg.github.io/toolbox/

Turn RegEx into DFA. I have offered the RegEx input if the screenshot is too small.

Dr.Paquet's tool:

remove left recursion , remove repeat/optional, turn original gramar into atocc and ucalgary,generate first and follow file

University of Calgary tool:

Check whether the grammar is ll1 , check conflict, generate simple possible src file to test derivation

AtoCC grammar checker

To check the specific first and follow conflict

Library:

java.io.PrintStream;

Error is in different file, so I change outputstream

javax.swing.JOptionPane: It is a build-in library that can generate a basic input output GUI fo our .exe.

Language: Java

Class Recommendation.

VirsionControl: Git

I have create a private repo to control version, and work with my desktop and laptop at the same time