*# Student ID: 1820047*

*from MCVisitor import MCVisitor*

*from MCParser import MCParser*

*from AST import \**

*from functools import \**

**class ASTGeneration(MCVisitor):**

**def visitProgram(self, ctx: MCParser.ProgramContext):**

*# program : declare+ EOF;*

dec\_list = []

for x in ctx.declare():

if isinstance(self.visitDeclare(x), list):

dec\_list.extend(self.visitDeclare(

x) if self.visitDeclare(x) else [])

else:

dec\_list.append(self.visitDeclare(x))

return Program(dec\_list)

**def visitDeclare(self, ctx: MCParser.DeclareContext):**

*# declare : var\_dec SEMI | func\_dec;*

return self.visit(ctx.var\_dec()) if ctx.var\_dec() else self.visit(ctx.func\_dec())

**def visitVar\_dec(self, ctx: MCParser.Var\_decContext):**

*# var\_dec: var\_list\_dec+;*

return reduce(lambda a, b: a+b, [self.visit(x) for x in ctx.var\_list\_dec()], [])

**def visitVar\_list\_dec(self, ctx: MCParser.Var\_list\_decContext):**

*# var\_list\_dec : primitive\_type var (COMMA var)\* ;*

var\_list = [self.visit(x) for x in ctx.var()]

return [VarDecl(x[0], self.visit(ctx.primitive\_type())) if x[1] == 'NoneArrayType' else VarDecl(x[0], ArrayType(x[1], self.visit(ctx.primitive\_type()))) for x in var\_list]

**def visitVar(self, ctx: MCParser.VarContext):**

*# var : ID (LSB intlit RSB)? ;*

return (ctx.ID().getText(), 'NoneArrayType') if ctx.getChildCount() == 1 else (ctx.ID().getText(), int(ctx.INTLIT().getText()))

**def visitFunc\_dec(self, ctx: MCParser.Func\_decContext):**

*# func\_dec : func\_type func\_name LP params\_list? RP* *blk\_stmt ;*

return FuncDecl(Id(self.visit(ctx.func\_name())), self.visit(ctx.params\_list()), self.visit(ctx.func\_type()), self.visit(ctx.blk\_stmt())) if ctx.params\_list() else FuncDecl(Id(self.visit(ctx.func\_name())), [], self.visit(ctx.func\_type()), self.visit(ctx.blk\_stmt()))

**def visitParams\_list(self, ctx: MCParser.Params\_listContext):**

*# params\_list : param (COMMA param)\* ;*

return [self.visit(x) for x in ctx.param()]

**def visitParam(self, ctx: MCParser.ParamContext):**

*# params\_list : param (COMMA param)\* ;*

return VarDecl(self.visit(ctx.identifier()), ArrayPointerType(self.visit(ctx.primitive\_type()))) if ctx.LSB() else VarDecl(self.visit(ctx.identifier()), self.visit(ctx.primitive\_type()))

**def visitFunc\_name(self, ctx: MCParser.Func\_nameContext):**

*# func\_name : identifier ;*

return self.visit(ctx.identifier())

**def visitFunc\_type(self, ctx: MCParser.Func\_typeContext):**

*# func\_type : primitive\_type | void\_type | array\_ptr\_type ;*

return self.visit(ctx.primitive\_type()) if ctx.primitive\_type() else self.visit(ctx.void\_type()) if ctx.void\_type() else self.visit(ctx.array\_ptr\_type())

**def visitBlk\_stmt(self, ctx: MCParser.Blk\_stmtContext):**

*# blk\_stmt : LB body? RB ;*

return Block(self.visit(ctx.body())) if ctx.body() else Block([])

**def visitBody(self, ctx: MCParser.BodyContext):**

*# body : var\_dec\_or\_stmt+ ;*

body\_list = []

for x in ctx.var\_dec\_or\_stmt():

if isinstance(self.visit(x), list):

body\_list.extend(self.visit(x) if self.visit(x) else [])

else:

body\_list.append(self.visit(x))

return body\_list

**def visitVar\_dec\_or\_stmt(self, ctx:** **MCParser.Var\_dec\_or\_stmtContext):**

*# var\_dec\_or\_stmt : var\_dec SEMI | stmt ;*

return self.visit(ctx.var\_dec()) if ctx.var\_dec() else self.visit(ctx.stmt())

**def visitStmt(self, ctx: MCParser.StmtContext):**

return self.visit(ctx.blk\_stmt()) if ctx.blk\_stmt() else self.visit(ctx.if\_stmt()) if ctx.if\_stmt() else self.visit(ctx.do\_while\_stmt()) if ctx.do\_while\_stmt() else self.visit(ctx.for\_stmt()) if ctx.for\_stmt() else self.visit(ctx.brk\_stmt()) if ctx.brk\_stmt() else self.visit(ctx.cont\_stmt()) if ctx.cont\_stmt() else self.visit(ctx.ret\_stmt()) if ctx.ret\_stmt() else self.visit(ctx.exp\_stmt()) if ctx.exp\_stmt() else self.visit(ctx.assign\_stmt()) if ctx.assign\_stmt() else self.visit(ctx.func\_call())

**def visitIf\_stmt(self, ctx: MCParser.If\_stmtContext):**

*# if\_stmt : IF LP exp RP stmt (ELSE stmt)? ;*

return If(self.visit(ctx.exp()), self.visit(ctx.stmt(0)), self.visit(ctx.stmt(1))) if ctx.stmt(1) else If(self.visit(ctx.exp()), self.visit(ctx.stmt(0)))

**def visitDo\_while\_stmt(self, ctx: MCParser.Do\_while\_stmtContext):**

*# do\_while\_stmt : DO stmt+ WHILE exp SEMI ;*

return Dowhile([self.visit(x) for x in ctx.stmt()], self.visit(ctx.exp()))

**def visitFor\_stmt(self, ctx: MCParser.For\_stmtContext):**

*# for\_stmt : FOR LP exp SEMI exp SEMI exp RP stmt ;*

return For(self.visit(ctx.exp(0)), self.visit(ctx.exp(1)), self.visit(ctx.exp(2)), self.visit(ctx.stmt()))

**def visitBrk\_stmt(self, ctx: MCParser.Brk\_stmtContext):**

*# brk\_stmt : BREAK SEMI ;*

return Break()

**def visitCont\_stmt(self, ctx: MCParser.Cont\_stmtContext):**

*# cont\_stmt : CONTINUE SEMI ;*

return Continue()

**def visitRet\_stmt(self, ctx: MCParser.Ret\_stmtContext):**

*# ret\_stmt : RETURN exp? SEMI ;*

return Return(self.visit(ctx.exp())) if ctx.exp() else Return()

**def visitAssign\_stmt(self, ctx: MCParser.Assign\_stmtContext):**

*# assign\_stmt : assign SEMI ;*

return self.visit(ctx.assign())

**def visitFunc\_call(self, ctx: MCParser.Func\_callContext):**

*# func\_call : ID LP exps\_list? RP ;*

return CallExpr(Id(ctx.ID().getText()), self.visit(ctx.exps\_list())) if ctx.exps\_list() else CallExpr(Id(ctx.ID().getText()), [])

**def visitExps\_list(self, ctx: MCParser.Exps\_listContext):**

*# exps\_list : exp (COMMA exp)\* ;*

return [self.visit(x) for x in ctx.exp()]

**def visitExp\_stmt(self, ctx: MCParser.Exp\_stmtContext):**

*# exp\_stmt : exp SEMI ;*

return self.visit(ctx.exp())

**def visitExp(self, ctx: MCParser.ExpContext):**

*# exp : assign | exp1 ;*

return self.visit(ctx.assign()) if ctx.assign() else self.visit(ctx.exp1())

**def visitExp1(self, ctx: MCParser.Exp1Context):**

*# exp1 : exp1 OR exp2 | exp2 ;*

return self.visit(ctx.exp2()) if ctx.getChildCount() == 1 else BinaryOp(ctx.OR().getText(), self.visit(ctx.exp1()), self.visit(ctx.exp2()))

**def visitExp2(self, ctx: MCParser.Exp2Context):**

*# exp2 : exp2 AND exp3 | exp3 ;*

return self.visit(ctx.exp3()) if ctx.getChildCount() == 1 else BinaryOp(ctx.AND().getText(), self.visit(ctx.exp2()), self.visit(ctx.exp3()))

**def visitExp3(self, ctx: MCParser.Exp3Context):**

*# exp3 : exp4 (EQ | NEQ) exp4 | exp4 ;*

return self.visit(ctx.exp4(0)) if ctx.getChildCount() == 1 else BinaryOp(ctx.getChild(1).getText(), self.visit(ctx.exp4(0)), self.visit(ctx.exp4(1)))

**def visitExp4(self, ctx: MCParser.Exp4Context):**

*# exp4 : exp5 (LT | LTE | GT | GTE) exp5 | exp5 ;*

return self.visit(ctx.exp5(0)) if ctx.getChildCount() == 1 else BinaryOp(ctx.getChild(1).getText(), self.visit(ctx.exp5(0)), self.visit(ctx.exp5(1)))

**def visitExp5(self, ctx: MCParser.Exp5Context):**

*# exp5 : exp5 (ADD | SUB) exp6 | exp6 ;*

return self.visit(ctx.exp6()) if ctx.getChildCount() == 1 else BinaryOp(ctx.getChild(1).getText(), self.visit(ctx.exp5()), self.visit(ctx.exp6()))

**def visitExp6(self, ctx: MCParser.Exp6Context):**

*# exp6 : exp6 (DIV | MUL | MOD) exp7 | exp7 ;*

return self.visit(ctx.exp7()) if ctx.getChildCount() == 1 else BinaryOp(ctx.getChild(1).getText(), self.visit(ctx.exp6()), self.visit(ctx.exp7()))

**def visitExp7(self, ctx: MCParser.Exp7Context):**

*# exp7 : (SUB | NOT) exp7 | operands ;*

return self.visit(ctx.operands()) if ctx.getChildCount() == 1 else UnaryOp(ctx.getChild(0).getText(), self.visit(ctx.exp7()))

**def visitOperands(self, ctx: MCParser.OperandsContext):**

return self.visit(ctx.literal()) if ctx.literal() else Id(self.visit(ctx.identifier())) if ctx.identifier() else self.visit(ctx.func\_call()) if ctx.func\_call() else self.visit(ctx.exp()) if ctx.exp() else ArrayCell(self.visit(ctx.operands()), self.visit(ctx.postfix\_exp()))

**def visitAssign(self, ctx: MCParser.AssignContext):**

*# assign : assign\_left ASSIGN assign\_right ;*

return BinaryOp(ctx.ASSIGN().getText(), self.visit(ctx.assign\_left()), self.visit(ctx.assign\_right()))

**def visitAssign\_left(self, ctx: MCParser.AssignContext):**

*# assign\_left : identifier | index\_exp ;*

return Id(self.visit(ctx.identifier())) if ctx.identifier() else self.visit(ctx.index\_exp())

**def visitAssign\_right(self, ctx: MCParser.Assign\_rightContext):**

*# assign\_right : assign\_left ASSIGN assign\_right | exp ;*

return self.visit(ctx.exp()) if ctx.getChildCount() == 1 else BinaryOp(ctx.ASSIGN().getText(), self.visit(ctx.assign\_left()), self.visit(ctx.assign\_right()))

**def visitIndex\_exp(self, ctx: MCParser.Index\_expContext):**

*# index\_exp : operands postfix\_exp ;*

return ArrayCell(self.visit(ctx.operands()), self.visit(ctx.postfix\_exp()))

**def visitPostfix\_exp(self, ctx: MCParser.Postfix\_expContext):**

*# postfix\_exp : LSB exp RSB ;*

return self.visit(ctx.exp())

**def visitLiteral(self, ctx: MCParser.LiteralContext):**

return self.visit(ctx.intlit()) if ctx.intlit() else self.visit(ctx.stringlit()) if ctx.stringlit() else self.visit(ctx.floatlit()) if ctx.floatlit() else self.visit(ctx.boolit())

**def visitIntlit(self, ctx: MCParser.IntlitContext):**

*# intlit : INTLIT;*

return IntLiteral(int(ctx.INTLIT().getText()))

**def visitStringlit(self, ctx: MCParser.StringlitContext):**

*# stringlit : STRINGLIT ;*

return StringLiteral(ctx.STRINGLIT().getText())

**def visitFloatlit(self, ctx: MCParser.FloatlitContext):**

*# floatlit : FLOATLIT ;*

return FloatLiteral(float(ctx.FLOATLIT().getText()))

**def visitBoolit(self, ctx: MCParser.BoolitContext):**

*# boolit : TRUE | FALSE ;*

return BooleanLiteral(True) if ctx.TRUE() else BooleanLiteral(False)

**def visitVoid\_type(self, ctx: MCParser.Void\_typeContext):**

*# void\_type : VOID ;*

return VoidType()

**def visitArray\_ptr\_type(self, ctx:** MCParser.Array\_ptr\_typeContext):

*# array\_ptr\_type : primitive\_type LSB RSB ;*

return ArrayPointerType(self.visit(ctx.primitive\_type()))

**def visitPrimitive\_type(self, ctx:** MCParser.Primitive\_typeContext):

*# primitive\_type : INT | BOOLEAN | FLOAT | STRING ;*

return IntType() if ctx.INT() else BoolType() if ctx.BOOLEAN() else FloatType() if ctx.FLOAT() else StringType()

**def visitIdentifier(self, ctx: MCParser.IdentifierContext):**

*# identifier : ID ;*

return ctx.ID().getText()