**class ASTGeneration(MCVisitor):**

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

*'''program: declare+ EOF;'''*

return Program(list(reduce(lambda x,y: x+y,[self.visit(t) for t in ctx.declare()],[])))

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

*'''declare: var\_declaration | function\_declaration;'''*

return self.visit(ctx.getChild(0))

**def visitVar\_declaration(self, ctx:MCParser.Var\_declarationContext):**

*'''var\_declaration: primitive\_type var\_list SEMI;*

*'''*

var\_typ = self.visit(ctx.primitive\_type())

var\_lst = self.visit(ctx.var\_list())

return [VarDecl(x[0],ArrayType(x[1],var\_typ) if x[1]>0 else var\_typ) for x in var\_lst]

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

*'''var\_list: variable COMMA var\_list | variable ;'''*

return [self.visit(ctx.variable())] + self.visit(ctx.var\_list()) if ctx.var\_list() else [self.visit(ctx.variable())]

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

if ctx.INTEGER(): return IntType()

if ctx.FLOAT(): return FloatType()

if ctx.BOOLEAN(): return BoolType()

if ctx.STRING(): return StringType()

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

return VoidType()

**def visitVariable(self, ctx:MCParser.VariableContext):**

*'''variable: ID | ID LSB INTLIT RSB ;'''*

if ctx.getChildCount()==1:

return (ctx.ID().getText(),-1)

else:

return (ctx.ID().getText(),int(ctx.INTLIT().getText()))

**def visitFunction\_declaration(self, ctx:MCParser.Function\_declarationContext):**

*'''function\_declaration: return\_type function\_name LP param\_list? RP body ;'''*

func\_name = Id(self.visit(ctx.function\_name()))

param = self.visit(ctx.param\_list()) if ctx.param\_list() else []

return\_typ = self.visit(ctx.return\_type())

func\_body = self.visit(ctx.body())

return [FuncDecl(func\_name, param, return\_typ, func\_body)]

**def visitFunction\_name(self, ctx:MCParser.Function\_nameContext):**

*'''function\_name: ID ;'''*

return ctx.ID().getText()

**def visitParam\_list(self, ctx:MCParser.Param\_listContext):**

*'''param\_list: param\_declaration COMMA param\_list | param\_declaration ;'''*

return [self.visit(ctx.param\_declaration())] + self.visit(ctx.param\_list()) if ctx.param\_list() else [self.visit(ctx.param\_declaration())]

**def visitParam\_declaration(self, ctx:MCParser.Param\_declarationContext):**

*'''param\_declaration: ( primitive\_type ID ) | ( primitive\_type ID LSB RSB );'''*

return VarDecl(ctx.ID().getText(), self.visit(ctx.primitive\_type())) if (ctx.getChildCount() == 2) else VarDecl(ctx.ID().getText(), ArrayPointerType(self.visit(ctx.primitive\_type())))

**def visitReturn\_typeContext(self, ctx:MCParser.Return\_typeContext):**

*'''return\_type: primitive\_type | void\_type | array\_pointer\_type;'''*

if ctx.primitive\_type():

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

elif ctx.void\_type:

return self.visit(ctx.void\_type())

else:

return self.visit(ctx.array\_pointer\_type())

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

*'''array\_pointer\_type: primitive\_type LSB RSB;'''*

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

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

*'''body: block\_statement ;'''*

return self.visit(ctx.block\_statement())

**def visitBlock\_statement(self, ctx:MCParser.Block\_statementContext):**

*'''block\_statement: LCB statement\_list? RCB ;'''*

return Block(self.visit(ctx.statement\_list())) if ctx.statement\_list() else Block([])

**def visitStatement\_list(self, ctx:MCParser.Statement\_listContext):**

*'''statement\_list: stmt\_decl+ ;'''*

declList = []

for x in ctx.stmt\_decl():

decl = self.visit(x)

if isinstance(decl, list):

declList.extend(decl if decl else [])

else:

declList.append(decl)

return declList

**def visitStmt\_decl(self, ctx:MCParser.Stmt\_declContext):**

*'''stmt\_decl: statement | var\_declaration ;'''*

return self.visit(ctx.statement()) if ctx.statement() else self.visit(ctx.var\_declaration())

**def visitStatement(self, ctx:MCParser.StatementContext):**

*'''statement: block\_statement*

| assign\_statement

| if\_statement

| do\_while\_statement

| for\_statement

| break\_statement

| continue\_statement

| return\_statement

| expression\_statement

;'''

if ctx.block\_statement(): return self.visit(ctx.block\_statement())

elif ctx.assign\_statement(): return self.visit(ctx.assign\_statement())

elif ctx.if\_statement(): return self.visit(ctx.if\_statement())

elif ctx.do\_while\_statement(): return self.visit(ctx.do\_while\_statement())

elif ctx.for\_statement(): return self.visit(ctx.for\_statement())

elif ctx.break\_statement(): return self.visit(ctx.break\_statement())

elif ctx.continue\_statement(): return self.visit(ctx.continue\_statement())

elif ctx.return\_statement(): return self.visit(ctx.return\_statement())

else: return self.visit(ctx.expression\_statement())

**def visitAssign\_statement(self, ctx:MCParser.Assign\_statementContext):**

*'''assign\_statement: assignment SEMI;'''*

return self.visit(ctx.assignment())

**def visitAssignment(self, ctx:MCParser.AssignmentContext):**

*'''assignment: assign\_LHS ASSIGN assignment | exp ;'''*

if ctx.ASSIGN():

left = self.visit(ctx.assign\_LHS())

right = self.visit(ctx.assignment())

return BinaryOp(ctx.ASSIGN().getText(), left, right)

else:

return self.visit(ctx.exp())

**def visitAssign\_LHS(self, ctx:MCParser.Assign\_LHSContext):**

*'''assign\_LHS: ID | index\_expression ;'''*

return Id(ctx.ID().getText()) if ctx.ID() else self.visit(ctx.index\_expression())

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

*'''exp: exp OR exp1 | exp1 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp1())

else:

left = self.visit(ctx.exp())

right = self.visit(ctx.exp1())

return BinaryOp(ctx.OR().getText(), left, right)

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

*'''exp1: exp1 AND exp2 | exp2 ; '''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp2())

else:

left = self.visit(ctx.exp1())

right = self.visit(ctx.exp2())

return BinaryOp(ctx.AND().getText(), left, right)

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

*'''exp2: exp3 ( EQ | NEQ ) exp3 | exp3 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp3(0))

else:

left = self.visit(ctx.exp3(0))

right = self.visit(ctx.exp3(1))

op = ctx.EQ().getText() if ctx.EQ() else ctx.NEQ().getText()

return BinaryOp(op, left, right)

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

*'''exp3: exp4 ( LT | LTE | GT | GTE ) exp4 | exp4 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp4(0))

else:

left = self.visit(ctx.exp4(0))

right = self.visit(ctx.exp4(1))

if ctx.LT(): op = ctx.LT().getText()

elif ctx.LTE(): op = ctx.LTE().getText()

elif ctx.GT(): op = ctx.GT().getText()

else: op = ctx.GTE().getText()

return BinaryOp(op, left, right)

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

*'''exp4: exp4 ( ADD | SUB ) exp5 | exp5 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp5())

else:

left = self.visit(ctx.exp4())

right = self.visit(ctx.exp5())

op = ctx.ADD().getText() if ctx.ADD() else ctx.SUB().getText()

return BinaryOp(op, left, right)

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

*'''exp5: exp5 ( DIV | MUL | DIV\_INT ) exp6 | exp6 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp6())

else:

left = self.visit(ctx.exp5())

right = self.visit(ctx.exp6())

if ctx.DIV(): op = ctx.DIV().getText()

elif ctx.MUL(): op = ctx.MUL().getText()

else: op = ctx.DIV\_INT().getText()

return BinaryOp(op, left, right)

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

*'''exp6: ( SUB | NOT ) exp6 | exp7 ;'''*

if ctx.getChildCount() == 1:

return self.visit(ctx.exp7())

else:

right = self.visit(ctx.exp6())

op = ctx.SUB().getText() if ctx.SUB() else ctx.NOT().getText()

return UnaryOp(op, right)

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

*'''exp7: operand ;'''*

return self.visit(ctx.operand())

**def visitExp\_bool(self, ctx:MCParser.Exp\_boolContext):**

*'''exp\_bool: assignment ;'''*

return self.visit(ctx.assignment())

**def visitExp\_for(self, ctx:MCParser.Exp\_forContext):**

*'''exp\_for: assignment ;'''*

return self.visit(ctx.assignment())

**def visitExp\_re(self, ctx:MCParser.Exp\_reContext):**

*'''exp\_re: assignment ;'''*

return self.visit(ctx.assignment())

**def visitOperand(self, ctx:MCParser.OperandContext):**

*'''operand: literal | operand1 ;'''*

return self.visit(ctx.literal()) if ctx.literal() else self.visit(ctx.operand1())

**def toBool(self, x):**

return x == "true"

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

*''' literal: INTLIT | FLOATLIT | BOOLLIT | STRINGLIT;'''*

if ctx.INTLIT():

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

elif ctx.FLOATLIT():

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

elif ctx.BOOLLIT():

return BooleanLiteral(self.toBool(ctx.BOOLLIT().getText()))

else:

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

**def visitOperand1(self, ctx:MCParser.Operand1Context):**

*'''operand1: ID | func\_call | LP assignment RP | operand1 operand2 ;'''*

if ctx.ID():

return Id(ctx.ID().getText())

elif ctx.func\_call():

return self.visit(ctx.func\_call())

elif ctx.assignment():

return self.visit(ctx.assignment())

else:

op1 = self.visit(ctx.operand1())

op2 = self.visit(ctx.operand2())

return ArrayCell(op1, op2)

**def visitOperand2(self, ctx:MCParser.Operand2Context):**

*'''operand2: LSB exp RSB ;'''*

return self.visit(ctx.exp())

**def visitIndex\_expression(self,** **ctx:MCParser.Index\_expressionContext):**

*'''index\_expression: operand1 operand2 ;'''*

op1 = self.visit(ctx.operand1())

op2 = self.visit(ctx.operand2())

return ArrayCell(op1, op2)

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

'''func\_call: ID LP exp\_list? RP ;'''

exp\_lst = self.visit(ctx.exp\_list()) if ctx.exp\_list() else []

return CallExpr(Id(ctx.ID().getText()), exp\_lst)

**def visitExp\_list(self, ctx:MCParser.Exp\_listContext):**

*'''exp\_list: exp COMMA exp\_list | exp;'''*

return [self.visit(ctx.exp( ))] + self.visit(ctx.exp\_list()) if ctx.exp\_list() else [self.visit(ctx.exp())]

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

*'''assign\_right: assignment | exp ;'''*

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

**def visitIf\_statement(self, ctx:MCParser.If\_statementContext):**

*'''if\_statement: IF LP exp\_bool RP statement (ELSE statement)? ;'''*

return If(self.visit(ctx.exp\_bool()),self.visit(ctx.statement(0)),self.visit(ctx.statement(1))) if ctx.ELSE() else If(self.visit(ctx.exp\_bool()),self.visit(ctx.statement(0)))

**def visitDo\_while\_statement(self,** **ctx:MCParser.Do\_while\_statementContext):**

*'''do\_while\_statement: DO statement+ WHILE exp\_bool SEMI ; '''*

stmt\_lst = [self.visit(x) for x in ctx.statement()]

exp\_bool = self.visit(ctx.exp\_bool())

return Dowhile(stmt\_lst, exp\_bool)

**def visitFor\_statement(self, ctx:MCParser.For\_statementContext):**

*'''for\_statement: FOR LP exp\_for SEMI exp\_for SEMI exp\_for RP statement ;'''*

exp0 = self.visit(ctx.exp\_for(0))

exp1 = self.visit(ctx.exp\_for(1))

exp2 = self.visit(ctx.exp\_for(2))

stmt = self.visit(ctx.statement())

return For(exp0, exp1, exp2, stmt)

**def visitBreak\_statement(self,** **ctx:MCParser.Break\_statementContext):**

*'''break\_statement: BREAK SEMI ;'''*

return Break()

**def visitContinue\_statement(self,** **ctx:MCParser.Continue\_statementContext):**

*'''continue\_statement: CONTINUE SEMI ;'''*

return Continue()

**def visitReturn\_statement(self,** **ctx:MCParser.Return\_statementContext):**

*'''return\_statement: RETURN exp\_re? SEMI ;'''*

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

**def visitExpression\_statement(self,** **ctx:MCParser.Expression\_statementContext):**

*'''expression\_statement: exp SEMI ;'''*

return self.visit(ctx.exp())