|  |  |  |
| --- | --- | --- |
| Program | 🡪 | Decl Prog |
| Prog | 🡪 | Program | ε |
| Decl | 🡪 | VariableDecl | FunctionDecl |
| VariableDecl | 🡪 | Variable **;** |
| Variable | 🡪 | Type **ident** |
| Type | 🡪 | **int** | **double** | **bool** | **string** |
| FunctionDecl | 🡪 | Type **ident** **(**Formals**)** StmtBlock |
| Formals | 🡪 | Variable MoreFormals | ε |
| MoreFormals | 🡪 | **,** Variable MoreFormals | ε |
| StmtBlock | 🡪 | **{** BlockBody **}** |
| BlockBody | 🡪 | VariableDecl BlockBody | Stmt BlockBody | ε |
| Stmt | 🡪 | LValue= Expr **;** | IfStmt | WhileStmt | ReturnStmt | StmtBlock |
| IfStmt | 🡪 | **if** (Expr) Stmt ElseStmt |
| ElseStmt | 🡪 | **else** Stmt | ε |
| WhileStmt | 🡪 | **while** (Expr) Stmt |
| ReturnStmt | 🡪 | **return** RetExpr**;** |
| RetExpr | 🡪 | Expr | ε |
| Expr | 🡪 | Constant | LValue | Call | (Expr) | Expr**+**Expr | Expr**\***Expr | Expr**<**Expr | Expr**<=**Expr |
| LValue | 🡪 | **Ident** |
| Call | 🡪 | **ident** (Actuals) |
| Actuals | 🡪 | Expr MoreActuals | ε |
| MoreActuals | 🡪 | **,** Expr MoreActuals | ε |
| Constant | 🡪 | **intConstant** | **doubleConstant** | **boolConstant** | **stringConstant** | **null** |

1. **Operator Precedence:**

Now forming the grammar to handle precedence. 4 levels can be identified.

**Level 1:** Parentheses (). This level will also include all the base blocks i.e. {Constant, LValue, Call}

**Level 2:** Multiplication \*

**Level 3:** Addition +

**Level 4:** Comparisons {<, <=}

These Precedence’s are needed to change the productions of Expr. So, introducing new non-terminals.

**Expr** for level 4 i.e. Comparisons (Already existing non-terminal resued)

**Add** for level 3i.e.Addition

**Mul** for level 2 i.e. Multiplication

**ExprBase** for level 1 i.e. the Parentheses and base blocks i.e. {Constant, LValue, Call}

Hence, the new grammar can be written as:

Expr 🡪 Expr < Add | Expr <= Add | Add

Add 🡪 Add + Mul | Mul

Mul 🡪 Mul \* ExprBase | ExprBase

ExprBase 🡪 (Expr) | Constant | LValue | Call

1. **Left Refactoring:**

The Decl is indirectly having “Type ident” common prefix in both its productions. So, the grammar can be changed into:

Decl 🡪 Type ident RFDecl

RFDecl 🡪 ; | (Formals) StmtBlock

This grammar allowed me to simply remove the VariableDecl and FunctionDecl non-terminals. And BlockBody production has to be changed a little bit.

Left re-factoring is also needed in Expr production.

Expr 🡪 Expr < RFExpr | Add

RFExpr 🡪 Add | **=** Add

ExprBase is also including the left factoring indirectly because LValue and Call both have same prefix “ident”. So changing its grammar also:

ExprBase 🡪 (Expr) | Constant | ident CallOrLValue

CallOrLValue 🡪 (Actuals) | ε

**Grammar After adding precedence handling and doing left re-factoring is:**

|  |  |  |
| --- | --- | --- |
| Program | 🡪 | Decl Prog |
| Prog | 🡪 | Program | ε |
| Decl | 🡪 | Type **ident** RFDecl |
| RFDecl | 🡪 | **;** | **(**Formals**)** StmtBlock |
| Variable | 🡪 | Type **ident** |
| Type | 🡪 | **int** | **double** | **bool** | **string** |
| Formals | 🡪 | Variable MoreFormals | ε |
| MoreFormals | 🡪 | **,** Variable MoreFormals | ε |
| StmtBlock | 🡪 | **{** BlockBody **}** |
| BlockBody | 🡪 | Variable **;** BlockBody | Stmt BlockBody | ε |
| Stmt | 🡪 | **ident =** Expr **;** | IfStmt | WhileStmt | ReturnStmt | StmtBlock |
| IfStmt | 🡪 | **if** (Expr) Stmt ElseStmt |
| ElseStmt | 🡪 | **else** Stmt | ε |
| WhileStmt | 🡪 | **while** (Expr) Stmt |
| ReturnStmt | 🡪 | **return** RetExpr**;** |
| RetExpr | 🡪 | Expr | ε |
| Expr | 🡪 | Expr**<** RFExpr | Add |
| RFExpr | 🡪 | Add | **=** Add |
| Add | 🡪 | Add **+** Mul | Mul |
| Mul | 🡪 | Mul **\*** ExprBase | ExprBase |
| ExprBase | 🡪 | **(**Expr**)** | Constant | ident CallOrLValue |
| CallOrLValue | 🡪 | **(**Actuals**)** | ε |
| Actuals | 🡪 | Expr MoreActuals | ε |
| MoreActuals | 🡪 | **,** Expr MoreActuals | ε |
| Constant | 🡪 | **intConstant** | **doubleConstant** | **boolConstant** | **stringConstant** | **null** |

**3. Left Recursion:**

Now removing the left recursions from the grammar. The left recursions are only seen in Expr production rule. So, changing its grammar:

Expr 🡪 Add Expr’

Expr’ 🡪 < RFExpr Expr’ | ε

RFExpr 🡪Add | = Add

Add 🡪 Mul Add’

Add’ 🡪 + Mul Add’ | ε

Mul 🡪 ExprBase Mul’

Mul’ 🡪 \* ExprBase Mul’ | ε

ExprBase 🡪 (Expr) | Constant | ident CallOrLValue

**Grammar After removing Left Recursion:**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | Program | 🡪 | Decl Prog |
| 1 | Prog | 🡪 | Program | ε |
| 2 | Decl | 🡪 | Type **ident** RFDecl |
| 3 | RFDecl | 🡪 | **;** | **(**Formals**)** StmtBlock |
| 4 | Variable | 🡪 | Type **ident** |
| 5 | Type | 🡪 | **int** | **double** | **bool** | **string** |
| 6 | Formals | 🡪 | Variable MoreFormals | ε |
| 7 | MoreFormals | 🡪 | **,** Variable MoreFormals | ε |
| 8 | StmtBlock | 🡪 | **{** BlockBody **}** |
| 9 | BlockBody | 🡪 | Variable **;** BlockBody | Stmt BlockBody | ε |
| 10 | Stmt | 🡪 | **ident =** Expr **;** | IfStmt | WhileStmt | ReturnStmt | StmtBlock |
| 11 | IfStmt | 🡪 | **if** **(**Expr**)** Stmt ElseStmt |
| 12 | ElseStmt | 🡪 | **else** Stmt | ε |
| 13 | WhileStmt | 🡪 | **while** **(**Expr**)** Stmt |
| 14 | ReturnStmt | 🡪 | **return** RetExpr**;** |
| 15 | RetExpr | 🡪 | Expr | ε |
| 16 | Expr | 🡪 | Add Expr’ |
| 17 | Expr’ | 🡪 | **<** RFExpr Expr’ | ε |
| 18 | RFExpr | 🡪 | Add | **=** Add |
| 19 | Add | 🡪 | Mul Add’ |
| 20 | Add’ | 🡪 | **+** Mul Add’ | ε |
| 21 | Mul | 🡪 | ExprBase Mul’ |
| 22 | Mul’ | 🡪 | \* ExprBase Mul’ | ε |
| 23 | ExprBase | 🡪 | **(**Expr**)** | Constant | ident CallOrLValue |
| 24 | CallOrLValue | 🡪 | **(**Actuals**)** | ε |
| 25 | Actuals | 🡪 | Expr MoreActuals | ε |
| 26 | MoreActuals | 🡪 | **,** Expr MoreActuals | ε |
| 27 | Constant | 🡪 | **intConstant** | **doubleConstant** | **boolConstant** | **stringConstant** | **null** |

* 1. **FIRST Sets:**

|  |  |  |
| --- | --- | --- |
| Non-Terminals | FIRST Set Inc | FIRST Set |
| Program | FIRST(Decl) | {**int, double, bool, string**} |
| Prog | FIRST(Program) U {**ε**} | {**int, double, bool, string**, **ε**} |
| Decl | FIRST(Type) | {**int, double, bool, string**} |
| RFDecl |  | {**;, (**} |
| Variable | FIRST(Type) | {**int, double, bool, string**} |
| Type |  | {**int, double, bool, string**} |
| Formals | FIRST(Variable) U {**ε**} | {**int, double, bool, string, ε**} |
| MoreFormals |  | {**,** , **ε** } |
| StmtBlock |  | {**{**} |
| BlockBody | FIRST(Variable) U FIRST(Stmt) U {**ε**} | {**int, double, bool, string, Ident, if, while, return, {, ε**} |
| Stmt | Ident U FIRST(IfStmt) U FIRST(WhileStmt) U FIRST(ReturnStmt) U FIRST(StmtBlock) | {**Ident, if, while, return, {**} |
| IfSmt |  | {**if**} |
| ElseStmt |  | {**else**, **ε**} |
| WhileStmt |  | {**while**} |
| ReturnStmt |  | {**return**} |
| RetExpr | FIRST(Expr) U {**ε**} | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null, ε**} |
| Expr | FIRST(Add) | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null**} |
| Expr’ |  | {**<, ε**} |
| RFExpr | FIRST(Add) U {**=**} | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null, =**} |
| Add | FIRST(Mul) | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null**} |
| Add’ |  | {**+**, **ε**} |
| Mul | FIRST(ExprBase) | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null**} |
| Mul’ |  | {**\***, **ε**} |
| ExprBase | {**(, ident**} U FIRST(Constant) | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null**} |
| CallOrLValue |  | {**(** , **ε** } |
| Actuals | FIRST(Expr) U {**ε**} | {**(, ident, intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null, ε**} |
| MoreActuals |  | {**,** , **ε** } |
| Constant |  | {**intConstant,** **doubleConstant,**  **boolConstant,** **stringConstant,** **null**} |

**4.2 FOLLOW Sets:**

|  |  |  |
| --- | --- | --- |
| Non-Terminals | FOLLOW Set Inc | FOLLOW Sets |
| Program | {$} U FOLLOW(Prog) | {$} |
| Prog | FOLLOW(Program) | {$} |
| Decl | FIRST(Prog) – {**ε**} U FOLLOW(Program) | {**int, double, bool, string**, $} |
| RFDecl | FOLLOW(Decl) | {**int, double, bool, string**, $} |
| Variable | FIRST(MoreFormals) – {**ε**} U FOLLOW(Formals) U {**;**} | {**,**} U {**)**} U {**;**}  = {**,**, **;**, **)**} |
| Type |  | {**ident**} |
| Formals |  | {**)**} |
| MoreFormals | FOLLOW(Formals) | {**)**} |
| StmtBlock | FOLLOW(RFDecl) U FOLLOW(Stmt) | {**int, double, bool, string, Ident, if, while, return, {, }, else**} |
| BlockBody |  | {**}**} |
| Stmt | FIRST(BlockBody) – {**ε**} U FOLLOW(BlockBody) U FIRST(ElseStmt) – {**ε**} U FOLLOW(ElseStmt) U FOLLOW(IfStmt) U FOLLOW(WhileStmt) | {**int, double, bool, string, Ident, if, while, return, {**} U {**}**} U {**else**} |
| IfSmt | FOLLOW(Stmt) | {**int, double, bool, string, Ident, if, while, return, {, }, else**} |
| ElseStmt | FOLLOW(Stmt) | {**int, double, bool, string, Ident, if, while, return, {, }, else**} |
| WhileStmt | FOLLOW(Stmt) | {**int, double, bool, string, Ident, if, while, return, {, }, else**} |
| ReturnStmt | FOLLOW(Stmt) | {**int, double, bool, string, Ident, if, while, return, {, }, else**} |
| RetExpr |  | {**;**} |
| Expr | {;, )} U FOLLOW(RetExpr) U FIRST(MoreActuals) – {**ε**} U FOLLOW(Actuals) | {**;**, **)**} U {**,**}  =  {**,**, **;**, **)**} |
| Expr’ | FOLLW(Expr) | {**,**, **;**, **)**} |
| RFExpr | FIRST(Expr’) – {**ε**} U FOLLOW(Expr’) | {**<**, **,**, **;**, **)**} |
| Add | FIRST(Expr’) – {**ε**} U FOLLOW(Expr) U FOLLOW(RFExpr) | {**<**, **,**, **;**, **)**} |
| Add’ | FOLLOW(Add) | {**<**, **,**, **;**, **)**} |
| Mul | FIRST(Add’) – {**ε**} U FOLLOW(Add) U FOLLOW(Add’) | {**+**, **<**, **,**, **;**, **)**} |
| Mul’ | FOLLOW(Mul) | {**+**, **<**, **,**, **;**, **)**} |
| ExprBase | FIRST(Mul’) – {**ε**} U FOLLOW(Mul) U FOLLOW(Mul’) | {**\***, **+**, **<**, **,**, **;**, **)**} |
| CallOrLValue | FOLLOW(ExprBase) | {**\***, **+**, **<**, **,**, **;**, **)**} |
| Actuals |  | {**)**} |
| MoreActuals | FOLLOW(Actuals) | {**)**} |
| Constant | FOLLOW(ExprBase) | {**\***, **+**, **<**, **,**, **;**, **)**} |