1	S'	S	S' IR = S.IR print(S' IR)	
2	S	<Body>	S.IR = <Body>.IR	
3	S	<Body> S	S.IR = <Body>.IR + S.IR	
4	<Body>	<Dec>	<B>.IR = <D>.IR	
5	<Dec>	<VarDec>	if global declaration is not a constant: ERROR <D>.IR = <VD>.IR	Transform
6	<Dec>	<FunDec>	<D>.IR = <F>.IR	
7	<VarDec>	int ID ;	ID exists: ERROR ID.varType = 'int' <VD>.IR = 'ID = 0' insert into var symbol table	
8	<VarDec>	int ID = <Expr>> ;	ID exists: ERROR ID.varType = 'int' if <E>.varType != 'int': ERROR <VD>.IR = <E>.IR + 'ID = <E>.val' insert into var symbol table	
9	<VarDec>	float ID ;	ID exists: ERROR ID.varType = 'float' <VD>.IR = 'ID = 0e+00' insert into var symbol table	
10	<VarDec>	float ID = <Expr>> ;	ID exists: ERROR ID.varType = 'float' if <E>.varType is not 'float': ERROR <VD>.IR = <E>.IR + 'ID = <E>.val' insert into var symbol table	
11	<FunDec>	int ID (<FormalParams>) <StmtBlock>	ID not exists: ERROR <FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(' + <FP>.IR + ') -> <SB>.IR + ':' <FD>.returnType = 'int' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table pop out the params from var table	Transform
12	<FunDec>	float ID (<FormalParams>) <StmtBlock>	ID not exists: ERROR <FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(' + <FP>.IR + ') -> <SB>.IR + ':' <FD>.returnType = 'float' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table pop out the params from var table	Transform
13	<FunDec>	void ID (<FormalParams>) <StmtBlock>	ID not exists: ERROR <FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(' + <FP>.IR + ') -> None:' <FD>.returnType = 'void' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table pop out the params from var table	Transform
14	<FormalParams>	<ParamList>	<FP>.paramType = <PL>.paramType <FP>.paramName = <PL>.paramName <FP>.IR = <PL>.IR All params push into var table	
15	<FormalParams>	void	<FP>.paramType = [] <FP>.paramName = []	
16	<FormalParams>	ε	<FP>.paramType = [] <FP>.paramName = []	
17	<ParamList>	<Param>	<PL>.IR = <P>.IR	
18	<ParamList>	<Param> , <ParamList>	<PL1>.paramType = <PL2>.paramType <PL1>.paramName = <PL2>.paramName <PL1>.paramType.append(<P>.type) <PL1>.paramName.append(<P>.name) <PL1>.IR = <P>.IR + ', ' + <PL2>.IR	
19	<Param>	int ID	ID val is the same as a global var: ERROR <P>.type = 'int' <P>.name = ID.val <P>.IR = 'int %ID'	
20	<Param>	float ID	ID val is the same as a global var: ERROR <P>.type = 'float' <P>.name = ID.val <P>.IR = 'float %ID'	
21	<StmtBlock>	(<Stmts>)	<SB>.IR = <S>.IR <SB>.returnType = <S>.returnType pop <S>.innerVarAmount vars from var symbol table	
22	<Stmts>	<Stmt> <Stmts>	<Stmts1>.IR = <Stmt>.IR + <Stmts2>.IR if <S>.<Stmt2>.rT equal: <Stmts1>.rT = <Stmts>.rT else: ERROR <Stmts1>.innerVarAmount = <Stmt>.IVA + <Stmts2>.IVA	
23	<Stmts>	<Stmt>	<Stmts>.IR = <S>.IR <Stmts>.returnType = <S>.returnType <Stmts>.innerVarAmount = <Stmt>.innerVarAmount	
24	<Stmt>	<VarDec>	<S>.IR = <VD>.IR <S>.returnType = 'void' <Stmt>.innerVarAmount += 1 (default is 0)	
25	<Stmt>	<IfStmt>	<S>.IR = <IS>.IR <S>.returnType = <IS>.returnType	
26	<Stmt>	<WhileStmt>	<S>.IR = <WS>.IR <S>.returnType = <WS>.returnType	
27	<Stmt>	<ReturnStmt>	<S>.IR = <RS>.IR <S>.returnType = <RS>.returnType	
28	<Stmt>	<AssignStmt>	<S>.IR = <AS>.IR <S>.returnType = 'void'	
29	<Stmt>	ID <FuncCall> ;	ID not in func table: ERROR <FC>.args does not match func table item: ERROR if type of args don't match: ERROR ID.funcReturnType is not void: ERROR <S>.IR = <FC>.IR + 'call ID(...)'	
30	<AssignStmt>	ID = <Expr>> ;	if ID not in var table: ERROR if ID and <E>.varType not match: ERROR <AS>.IR = <E>.IR + 'ID = <E>.val' # ID = global	Transform
31	<ReturnStmt>	return <Expr>> ;	<RS>.IR = <E>.IR + 'ret <E>.val |' <RS>.returnType = <E>.varType	
32	<ReturnStmt>	return ;	<RS>.IR = 'ret void' <RS>.returnType = 'void'	
33	<WhileStmt>	while (<Expr>>) <StmtBlock>	if <E> varType is not 'int': ERROR <WS>.returnType = <SB>.returnType <WS>.IR = <E>.IR + 'goto L1 if + L1.1' + 'if<E>.val == 1) goto L2 else L3.' + 'L2' + <SB>.IR + 'goto L1' + 'L3.'	
34	<IfStmt>	if (<Expr>>) <StmtBlock> else <StmtBlock>	if <E> varType is not 'int': ERROR <SB1>.rT equals to <SB2>.rT: <IS>.rT = <SB1>.rT else: ERROR <IS>.IR = <E>.IR + 'if(<E>.val != 1) goto L1 else L2' + 'L1' + <SB1>.IR + 'goto L3' + 'L2' + <SB2>.IR + 'L3'	
35	<IfStmt>	if (<Expr>>) <StmtBlock>	if <E> varType is not 'int': ERROR <IS>.returnType = <SB>.returnType <IS>.IR = <E>.IR + 'if (<E>.val != 1) goto L1 else L2:' + 'L1' + <SB>.IR + 'L2:'	
36	<Expr>>	<AddExpr>>	<E>.val = <A>.val <E>.varType = <A>.varType <E>.IR = <A>.IR	
37	<Expr>>	<AddExpr>> < <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val < <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3:'newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
38	<Expr>>	<AddExpr>> <= <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val <= <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3:'newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
39	<Expr>>	<AddExpr>> > <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val >> <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3:'newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
40	<Expr>>	<AddExpr>> >= <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val >= <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3:'newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
41	<Expr>>	<AddExpr>> == <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val == <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3:'newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
42	<Expr>>	<AddExpr>> != <Expr>>	<A> <E1>.type not match: ERROR <E1>.IR = <A>.IR + <E1>.IR + 'newTemp1 = (<A>.val != <E2>.val); if newTemp1 goto L1 else L2 L1: newTemp2 = 1 goto L3 L2: newTemp2 = 0 goto L3 L3: newTemp.varType = 'int' <E1>.val = newTemp <E1>.varType = newTemp.varType	
43	<AddExpr>	<Item> + <AddExpr>	if <I> <A2> type not match: ERROR <A1>.IR = <I>.IR + <A2>.IR + 'newTemp = <I>.val + <A2>.val' <A1>.val = newTemp <A1>.varType = newTemp.varType	
44	<AddExpr>	<Item> - <AddExpr>	if <I> <A2> type not match: ERROR <A1>.IR = <I>.IR + <A2>.IR + newTemp = <I>.val - <A2>.val' <A1>.val = newTemp <A1>.varType = newTemp.varType	
45	<AddExpr>	<Item>	<A>.val = <Item>.val <A>.varType = <Item>.varType <A>.IR = <Item>.IR	
46	<Item>	<Factor> * <Item>	if <F> <I2> type don't match: ERROR <I1>.IR = <F>.IR + <I2>.IR + 'newTemp = <F>.val * <I2>.val' <I1>.val = newTemp <I1>.varType = newTemp.varType	
47	<Item>	<Factor> / <Item>	if <F> <I2> type don't match: ERROR <I1>.IR = <F>.IR + <I2>.IR + 'newTemp = <F>.val / <I2>.val' <I1>.val = newTemp <I1>.varType = newTemp.varType	
48	<Item>	<Factor>	<Item>.val = <Factor>.val <Item>.varType = <Factor>.varType <Item>.IR = <Factor>.IR	
49	<Factor>	inum	<Factor>.val = inum.lexVal <Factor>.varType = 'int'	
50	<Factor>	fnum	<Factor>.val = fnum.lexVal <Factor>.varType = 'float'	
51	<Factor>	(<Expr>>)	<Factor>.val = <Expr>>.val <Factor>.varType = <Expr>>.varType <Factor>.IR = <Expr>>.IR	
52	<Factor>	ID	ID not in var table : ERROR else: <Factor>.val = ID.val <Factor>.varType = ID.varType <Factor>.IR = ID + val # ID is global	Transform
53	<Factor>	ID <FuncCall>	ID not in func table: ERROR <FC>.args does not match func table item: ERROR if type of args don't match: ERROR ID.funcReturnType is void: ERROR <F>.IR = <FC>.IR + 'call ID(...)' + 'ret |' newTemp.varType = ID.funcReturnType <F>.val = newTemp <F>.varType = newTemp.varType	
54	<FuncCall>	(<ActualArgs>)	<F>.args = <A>.args <F>.argType = <A>.argType <F>.IR = <A>.IR	
55	<ActualArgs>	<ArgList>	<A>.args = <Arg>.args <A>.argType = <Arg>.argType <A>.IR = <Arg>.IR	
56	<ActualArgs>	void	<A>.args = [] <A>.argType = []	
57	<ActualArgs>	ε	<A>.args = [] <A>.argType = []	
58	<ArgList>	<Expr>> , <ArgList>	<Arg1>.args = <Arg2>.args <Arg1>.argType = <Arg2>.argType <Arg1>.args.putHead(<E>.val) <Arg1>.argType.putHead(<E>.varType) <Arg1>.IR = <E>.IR + <Arg2>.IR	
59	<ArgList>	<Expr>>	<Arg>.args.putHead(<E>.val) <Arg>.argType.putHead(<E>.varType) <Arg>.IR = <E>.IR	