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>	<Decl>	<B>.IR = <D>.IR	
5	<Decl>	<VarDecl>	if global declaration is not a constant: ERROR <D>.IR = <VD>.IR	Transform
6	<Decl>	<FuncDecl>	<D>.IR = <FD>.IR	
7	<VarDecl>	int ID ;	ID exists: ERROR ID.valType = 'int' <VD>.IR = 'ID = 0' insert into var symbol table	
8	<VarDecl>	int ID = <Exprsn> ;	ID exists: ERROR ID.valType = 'int' if <E>.valType is not 'int': ERROR <VD>.IR = <E>.IR + 'ID = <E>.val' insert into var symbol table	
9	<VarDecl>	float ID ;	ID exists: ERROR ID.valType = 'float' <VD>.IR = 'ID = 0e+00' insert into var symbol table	
10	<VarDecl>	float ID = <Exprsn> ;	ID exists: ERROR ID.valType = 'float' if <E>.valType is not 'float': ERROR <VD>.IR = <E>.IR + 'ID = <E>.val' insert into var symbol table	
11	<FuncDecl>	int ID (<FormalParams>) <StmtBlock>	ID.val 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	<FuncDecl>	float ID (<FormalParams>) <StmtBlock>	ID.val 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	<FuncDecl>	void ID (<FormalParams>) <StmtBlock>	ID.val exists: ERROR <FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(' + <FP>.IR + ') {' + <SB>.IR + 'ret void }' <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>.paramType.putHead(<P>.type) <PL>.paramName.putHead(<P>.name) <PL>.IR = <P>.IR	
18	<ParamList>	<Param> , <ParamList>	<PL1>.paramType = <PL2>.paramType <PL1>.paramName = <PL2>.paramName <PL1>.paramType.putHead(<P>.type) <PL1>.paramName.putHead(<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> <Stmts2> rT equal: <Stmts1>.rT = <Stmt>.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>	<VarDecl>	<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 = <Exprsn> ;	if ID not in var table: ERROR if ID and <E> valType not match: ERROR <AS>.IR = <E>.IR + 'ID = <E>.val' if ID is global!	Transform
31	<ReturnStmt>	return <Exprsn> ;	<RS>.IR = <E>.IR + 'ret <E>.val ;' <RS>.returnType = <E>.varType	
32	<ReturnStmt>	return ;	<RS>.IR = 'ret void' <RS>.returnType = 'void'	
33	<WhileStmt>	while (<Exprsn>) <StmtBlock>	if <E>.valType is not 'int': ERROR <WS>.returnType = <SB>.returnType <WS>.IR = <E>.IR + 'goto L1' + 'L1:' + 'if(<E>.val == 1) goto L2 else L3:' + 'L2' + <SB>.IR + 'goto L1' + 'L3:'	
34	<IfStmt>	if (<Exprsn>) <StmtBlock> else <StmtBlock>	if <E>.valType 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 (<Exprsn>) <StmtBlock>	if <E>.valType 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	<Exprsn>	<AddExprsn>	<E>.val = <A>.val <E>.valType = <A>.valType <E>.IR = <A>.IR	
37	<Exprsn>	<AddExprsn> < <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
38	<Exprsn>	<AddExprsn> <= <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
39	<Exprsn>	<AddExprsn> > <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
40	<Exprsn>	<AddExprsn> >= <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
41	<Exprsn>	<AddExprsn> == <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
42	<Exprsn>	<AddExprsn> != <Exprsn>	<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.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType	
43	<AddExprsn>	<Item> + <AddExprsn>	if <I> <A2> type not match: ERROR <A1>.IR = <I>.IR + <A2>.IR + 'newTemp = <I>.val + <A2>.val' <A1>.val = newTemp <A1>.valType = newTemp.valType	
44	<AddExprsn>	<Item> - <AddExprsn>	if <I> <A2> type not match: ERROR <A1>.IR = <I>.IR + <A2>.IR + 'newTemp = <I>.val - <A2>.val' <A1>.val = newTemp <A1>.valType = newTemp.valType	
45	<AddExprsn>	<Item>	<A>.val = <Item>.val <A>.valType = <Item>.valType <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>.valType = newTemp.valType	
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>.valType = newTemp.valType	
48	<Item>	<Factor>	<Item>.val = <Factor>.val <Item>.valType = <Factor>.valType <Item>.IR = <Factor>.IR	
49	<Factor>	inum	<Factor>.val = inum.lexVal <Factor>.valType = 'int'	
50	<Factor>	- inum	<Factor>.val = - inum.lexVal <Factor>.valType = 'int'	
51	<Factor>	fnum	<Factor>.val = fnum.lexVal <Factor>.valType = 'float'	
52	<Factor>	- fnum	<Factor>.val = - fnum.lexVal <Factor>.valType = 'float'	
53	<Factor>	(<Exprsn>)	<Factor>.val = <Exprsn>.val <Factor>.valType = <Exprsn>.valType <Factor>.IR = <Exprsn>.IR	
54	<Factor>	ID	ID not in var table : ERROR else: <Factor>.val = ID.val <Factor>.valType = ID.varType ~~<Factor>.IR = 'ID = xxx'~~ if ID is global	Transform
55	<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 + 'newTemp = call ID(...)' newTemp.valType = ID.funcReturnType <F>.val = newTemp <F>.valType = newTemp.valType	
56	<FuncCall>	(<ActualArgs>)	<F>.args = <Ac>.args <F>.argType = <Ac>.argType <F>.IR = <Ac>.IR	
57	<ActualArgs>	<ArgList>	<Ac>.args = <Arg>.args <Ac>.argType = <Arg>.argType <Ac>.IR = <Arg>.IR	
58	<ActualArgs>	void	<A>.args = [] <A>.argType = []	
59	<ActualArgs>	ε	<A>.args = [] <A>.argType = []	
60	<ArgList>	<Exprsn> , <ArgList>	<Arg1>.args = <Arg2>.args <Arg1>.argType = <Arg2>.argType <Arg1>.args.putHead(<E>.val) <Arg1>.argType.putHead(<E>.valType) <Arg1>.IR = <E>.IR + <Arg2>.IR	
61	<ArgList>	<Exprsn>	<Arg>.args.putHead(<E>.val) <Arg>.argType.putHead(<E>.valType) <Arg>.IR = <E>.IR	