S'	S	S'.IR = S.IR print(S'.IR)
S	<Body>	S.IR = <Body>.IR
S	<Body> S	S.IR = <Body>.IR + S.IR
<Body>	<Decl>	<B>.IR = <D>.IR
<Decl>	<VarDecl>	<D>.IR = <VD>.IR
<Decl>	<FuncDecl>	<D>.IR = <FD>.IR
<VarDecl>	int ID ;	ID.valType = 'int' IR = 'ID = 0' insert into var symbol table
<VarDecl>	int ID = <Exprsn> ;	ID.valType = 'int' if <E>.valType is not 'int': ERROR IR = 'ID = <E>.val' insert into var symbol table
<VarDecl>	float ID ;	ID.valType = 'float' IR = 'ID = 0.0' insert into var symbol table
<VarDecl>	float ID = <Exprsn> ;	ID.valType = 'float' if <E>.valType is not 'float': ERROR IR = 'ID = <E>.val' insert into var symbol table
<FuncDecl>	int ID (<FormalParams>) <StmtBlock>	<FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(...):' + <SB>.IR + 'j' <FD>.returnType = 'int' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table push <SB>.innerVarAmount vars from var symbol table
<FuncDecl>	float ID (<FormalParams>) <StmtBlock>	<FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(...):' + <SB>.IR + 'j' <FD>.returnType = 'float' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table push <SB>.innerVarAmount vars from var symbol table
<FuncDecl>	void ID (<FormalParams>) <StmtBlock>	<FD>.returnType != <SB>.rT: ERROR <FD>.IR = 'def func(...):' + <SB>.IR + 'j' <FD>.returnType = 'void' <FD>.paramName = <FP>.paramName <FD>.paramType = <FP>.paramType insert into func symbol table push <SB>.innerVarAmount vars from var symbol table
<FormalParams>	<ParamList>	<FP>.paramType = <PL>.paramType <FP>.paramName = <PL>.paramName
<FormalParams>	void	<FP>.paramType = [] <FP>.paramName = []
<FormalParams>	ε	<FP>.paramType = [] <FP>.paramName = []
<ParamList>	<Param>	<PL>.paramType.putHead(<P>.type) <PL>.paramName.putHead(<P>.name)
<ParamList>	<Param> , <ParamList>	<PL1>.paramType = <PL2>.paramType <PL1>.paramName = <PL2>.paramName <PL1>.paramType.putHead(<P>.type) <PL1>.paramName.putHead(<P>.name)
<Param>	int ID	<P>.type = 'int' <P>.name = ID.name
<Param>	float ID	<P>.type = 'float' <P>.name = ID.name
<StmtBlock>	{ <Stmts> }	<SB>.IR = <S>.IR <SB>.returnType = <S>.returnType <SB>.innerVarAmount = <S>.vA
<Stmts>	<Stmt> <Stmts>	<Stmts1>.IR = <Stmt>.IR + <Stmts2>.IR if <S> <Stmts2>.rT equal: <Stmts1>.rT = <Stmt>.rT else: ERROR <Stmts1>.innerVarAmount = <Stmt>.vA + <Stmts2>.vA
<Stmts>	<Stmt>	<Stmts>.IR = <S>.IR <Stmts>.returnType = <S>.returnType <Stmts>.innerVarAmount = <Stmt>.innerVarAmount
<Stmt>	<VarDecl>	<S>.IR = <VD>.IR <S>.returnType = 'void' <Stmt>.innerVarAmount += 1 (default is 0)
<Stmt>	<IfStmt>	<S>.IR = <IS>.IR <S>.returnType = <IS>.returnType
<Stmt>	<WhileStmt>	<S>.IR = <WS>.IR <S>.returnType = <WS>.returnType
<Stmt>	<ReturnStmt>	<S>.IR = <RS>.IR <S>.returnType = <RS>.returnType
<Stmt>	<AssignStmt>	<S>.IR = <AS>.IR <S>.returnType = 'void'
<AssignStmt>	ID = <Exprsn> ;	if ID not in var table: ERROR if ID and <E>.valType not match: ERROR IR = 'ID = <E>.val'
<ReturnStmt>	return <Exprsn> ;	IR = 'ret <E>.val;' <RS>.returnType = <E>.varType
<ReturnStmt>	return ;	IR = 'ret ;' <RS>.returnType = 'void'
<WhileStmt>	while (<Exprsn>) <StmtBlock>	<WS>.returnType = <SB>.returnType <WS>.IR = 'L1:' + <E>.IR + 'if(<E>.val != 1) goto L2:' + <SB>.IR + 'goto L1' + 'L2:' push <SB>.innerVarAmount vars from var symbol table
<IfStmt>	if (<Exprsn>) <StmtBlock> else <StmtBlock>	<SB1>.rT equals to <SB2>.rT: <S>.rT = <SB1>.rT else: ERROR <IF>.IR = <E>.IR + 'if (<E>.val != 1) goto L1:' + <SB1>.IR + 'goto L2' + 'L1:' + <SB2>.IR + 'L2:'
<IfStmt>	if (<Exprsn>) <StmtBlock>	<S>.returnType = <SB>.returnType <IF>.IR = <E>.IR + 'if (<E>.val != 1) goto L1:' + <SB>.IR + 'L1:' push <SB>.innerVarAmount vars from var symbol table
<Exprsn>	<AddExprsn>	<E>.val = <A>.val <E>.valType = <A>.valType
<Exprsn>	<AddExprsn> < <Exprsn>	IR = 'newTemp1 = (<A>.val < <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<Exprsn>	<AddExprsn> <= <Exprsn>	IR = 'newTemp1 = (<A>.val <= <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<Exprsn>	<AddExprsn> > <Exprsn>	IR = 'newTemp1 = (<A>.val > <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<Exprsn>	<AddExprsn> >= <Exprsn>	IR = 'newTemp1 = (<A>.val >= <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<Exprsn>	<AddExprsn> == <Exprsn>	IR = 'newTemp1 = (<A>.val == <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<Exprsn>	<AddExprsn> != <Exprsn>	IR = 'newTemp1 = (<A>.val != <E2>.val); if newTemp1 goto L1 newTemp2 = 0 goto L2 L1: newTemp2 = 1 L2:' newTemp.valType = 'int' <E1>.val = newTemp <E1>.valType = newTemp.valType
<AddExprsn>	<Item> + <AddExprsn>	if <I> <A2> type not match: ERROR IR = 'newTemp = <I>.val + <A2>.val' <A1>.val = newTemp <A1>.valType = newTemp.valType
<AddExprsn>	<Item> - <AddExprsn>	if <I> <A2> type not match: ERROR IR = 'newTemp = <I>.val - <A2>.val' <A1>.val = newTemp <A1>.valType = newTemp.valType
<AddExprsn>	<Item>	<A>.val = <Item>.val <A>.valType = <Item>.valType
<Item>	<Factor> * <Item>	if <F> <I2> type don't match: ERROR IR = 'newTemp = <F>.val * <I2>.val' <I1>.val = newTemp <I1>.valType = newTemp.valType
<Item>	<Factor> / <Item>	if <F> <I2> type don't match: ERROR IR = 'newTemp = <F>.val / <I2>.val' <I1>.val = newTemp <I1>.valType = Temp.valType
<Item>	<Factor>	<Item>.val = <Factor>.val <Item>.valType = <Factor>.valType
<Factor>	inum	<Factor>.val = inum.lexVal <Factor>.valType = 'int'
<Factor>	fnum	<Factor>.val = fnum.lexVal <Factor>.valType = 'float'
<Factor>	(<Exprsn>)	<Factor>.val = <Exprsn>.val <Factor>.valType = <Exprsn>.valType
<Factor>	ID	ID not in var table : ERROR else: <Factor>.val = ID.name <Factor>.valType = ID.varType
<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 IR = 'newTemp = call ID(...)' newTemp.valType = ID.funcReturnType <F>.val = newTemp <F>.valType = newTemp.valType
<FuncCall>	(<ActualArgs>)	<F>.args = <Ac>.args <F>.argType = <Ac>.argType
<ActualArgs>	<ArgList>	<Ac>.args = <Arg>.args <Ac>.argType = <Arg>.argType
<ActualArgs>	void	<A>.args = [] <A>.argType = []
<ActualArgs>	ε	<A>.args = [] <A>.argType = []
<ArgList>	<Exprsn> , <ArgList>	<Arg1>.args = <Arg2>.params <Arg1>.argType = <Arg2>.paramType <Arg1>.args.putHead(<E>.val) <Arg1>.argType.putHead(<E>.valType)
<ArgList>	<Exprsn>	<Arg>.args.putHead(<E>.val) <Arg>.argType.putHead(<E>.valType)