START:

<start>🡪 def <function><start> | <Class><start> | <For><start> | <While><start> | <If><start> | <Try><start> | <Del> nl <start> |<initialize>nl <start> | €

INITIALIZATION STATEMENT:

<initialize>🡪ID N = tlist.at(itr).getValuePart(); <ID\_rel\_>|<const><init2>|\* N = "\*"; <init3>| accessModifier accessMod = tlist.at(itr).getClassPart(); init4>|<static\_final>ID N= tlist.at(itr).getValuePart(); <ID\_rel> = <init5>  | this AcOp ID N= tlist.at(itr).getValuePart(); <ID\_rel>  <init1> | (<exp>)<OE> | Not <exp\_F> <OE> | <List2> | <Dictionary2>

<init1>🡪AsOP op = tlist.at(itr).getClassPart(); <exp>  |=<init5>  |<init2>

<init2> 🡪 <OE><In>if (in == "in") T="bool\_const";

<init3>🡪accessModifier am = tlist.at(itr).getClassPart(); <init4>|<init4>

<init4>🡪<static\_final>ID n1= tlist.at(itr).getValuePart(); <ID\_rel> = <init5> |ID n1=tlist.at(itr).getValuePart(); <ID\_rel> = <init5> 

<init5>🡪<const><OE>|<This>ID N= tlist.at(itr).getValuePart(); <ID\_rel  <init6> | <List> | <Dictionary>| <Class\_call> | <lambda>

<init6>🡪  =<init5>| <init2>

<ID\_rel\_>🡪<ID\_rel1\_>| (<arg\_list\_call>) <AcOP>

<ID\_rel1\_>🡪[<exp>] <ID\_rel1\_>| AcOp   ID N=tlist.at(itr).getValuePart(); <ID\_rel\_>  | <init1>

<AcOP>🡪 AcOp  ID <ID\_rel\_>|Є

<AM>🡪 accessModifier T= tlist.at(itr++).getValuePart(); | € T=""

<static\_final>🡪 static <s\_final> if (f == "final") s\_f+="f"; | final s\_f="f";

<s\_final>🡪 final f="final"; | € f="";

<id\_const>🡪<This> ID N = tlist.at(itr).getValuePart(); <ID\_rel>  | <const>

<const>🡪 int\_const T=”int\_const” | float\_const T=”float\_const” | string\_const T=”string\_const” | char\_const T=”char\_const” | bool\_const T=”bool\_const”

<Global>🡪 global <initialize>

<pointer>🡪 \* N=”\*” | € N=””

EXPRESSION:

<OE>🡪<exp\_MDM><exp\_PM><exp\_RELOP><exp\_AND><exp\_OR>

<exp>🡪<expAND><exp\_OR>

<exp\_OR>🡪 Or op = "Or"; <expAND> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_OR> | € T3=T

<expAND>🡪<expRELOP><exp\_AND>

<exp\_AND>🡪 And op = "And"; <expRELOP> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_AND> | € T3=T

<expRELOP>🡪<expPM><exp\_RELOP>

<exp\_RELOP>🡪 RelOp op = "RelOp"; <expPM> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_RELOP> | € T3=T

<expPM>🡪<expMDM><exp\_PM>

<exp\_PM>🡪 PM op = tlist.at(itr).getValuePart(); <expMDM> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_PM> | € T3=T

<expMDM>🡪<exp\_F><exp\_MDM>

<exp\_MDM>🡪 DM op = tlist.at(itr).getValuePart(); <exp\_F> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_MDM> | \* op = "\*"; <exp\_F> if((T2=isCompatible(T, T1, op))=="false") cout<<”Incompatible types” <exp\_MDM> |€ T3=T

<exp\_F>🡪<This>ID N = tlist.at(itr).getValuePart(); <ID\_rel>  <In> | <const><In> if (in == "in") T="bool\_const";| (<exp>) | Not <exp\_F> | <lambda>

<ID\_rel> 🡪 [exp] N += "[]"; <ID\_rel1> | (<arg\_list\_call>)<ID\_rel1> | AcOp    ID N=tlist.at(itr).getValuePart(); <ID\_rel>  | € N=N1

<ID\_rel1> 🡪 AcOp   ID N=tlist.at(itr).getValuePart(); <ID\_rel>  | <ID\_rel2>

<ID\_rel2> 🡪[exp] N += "[]"; <ID\_rel1> | € N=N1

<This>🡪 this AcOp N=”this” | € N=””

<In>🡪 in <In\_> in=”in” | € in=””

<In\_>🡪 ID N = tlist.at(itr).getValuePart(); |<List>

BODY:

<body>🡪<S\_St> |nl IndentInit <M\_St> IndentOut

<M\_St>🡪<S\_St><M\_St\_>

<M\_St\_>🡪<M\_St> | €

<S\_St>🡪 <S\_St1> nl| <S\_St2>

<S\_St1>🡪 break | continue | return <sst1> if (sst1()!=currentFuncType) cout<< "Return type and function type do not match."; | <Del>| <Global> |<initialize> | pass

<S\_St2>🡪<For> | <While> | <If> | <Try>

<sst1> 🡪 <exp> | <List2> | Dictionary2>  
<NL>🡪 nl | €

CLASS:

<Class>🡪 class ID N = tlist.at(itr).getValuePart(); currentClass = N;  (<inherit> v = inherit() ) insertClassTable(N, PUBLIC, false, false, v.at(1), v.at(0)); : nl IndentInit <class\_body><NL> IndentOut currentClass="global";

<Inherit>🡪 ID N1 = tlist.at(itr).getValuePart(); <inherit\_> N={N1, N2} | € N={“”, “”}

<inherit\_>🡪 , ID N = tlist.at(itr).getValuePart(); | € N=””

<class\_body>🡪<class\_body1> | pass

<class\_body1>🡪<initialize> nl <class\_body3> | def <class\_body2><class\_body3>

<class\_body2>🡪<constructor> | <function>

<class\_body3>🡪< class\_body1> | €

CLASS CALL:

<Class\_call>🡪 new ID N = tlist.at(itr).getValuePart(); ( <arg\_list\_call> ) T=PL+">"+N;

CONSTRUCTOR:

<constructor>🡪<AM> ID  ( createScope() <arg\_list> T += ">"; ) : <body> destroyScope()

<arg\_list>🡪<data\_type><pointer> ID N = tlist.at(itr).getValuePart(); <arg\_list1> T += "," + T1; | € T=””

<arg\_list1>🡪 , <data\_type><pointer> ID N = tlist.at(itr).getValuePart();<arg\_list1> T += "," + T1; | € T=””

<arg\_list\_call>🡪<exp><arg\_list\_call1> if (T1 != "") T += "," + T1; | € T=””

<arg\_list\_call1>🡪 , <exp><arg\_list\_call1> if (T1 != "") T += "," + T1; | € T=””

FUNCTION:

<function>🡪<data\_type><AM><function\_>

<function\_> 🡪 <static\_final> ID N = tlist.at(itr).getValuePart(); ( createScope() <arg\_list> T += ">" + T1; )  : <body> destroyScope() | ID N = tlist.at(itr).getValuePart(); ( createScope() <arg\_list> T += ">" + T1; )  : <body> destroyScope()

<data\_type>🡪 DT T= tlist.at(itr).getClassPart(); | string T=tlist.at(itr).getClassPart(); | ID T= tlist.at(itr).getClassPart();

LOOPS:

<For>🡪 for createScope() ID N = tlist.at(itr).getValuePart(); in <For\_> : <body> destroyScope()

<For\_>🡪 ID N = tlist.at(itr).getValuePart(); <ID\_rel>  | <List2> | <Dictionary2>

<While>🡪 while ( createScope() <exp>  ) : <body> destroyScope()

IF\_ELIF\_ELSE:

<If>🡪 if ( createScope() <exp> ) : <body> destroyScope()<Elif>

<Elif>🡪 elif ( createScope() <exp> ) : <body> destroyScope() <Elif> | <Else>

<Else>🡪 else : createScope() <body> destroyScope() | €

LIST/ARRAY:

<List>🡪<List2> | List ( <List1> )

<List1>🡪<List2> | ID N = tlist.at(itr).getValuePart();

<List2>🡪 [ <List3> ]

<List3>🡪<id\_const><List4> | <List2><List4> | <Dictionary2> <List4>|Є

<List4>🡪 , <List5> | €

<List5> 🡪 <id\_const><List4> | <List2><List4> | <Dictionary2> <List4>

DICTIONARY:

<Dictionary>🡪<Dictionary2> | dict (<Dictionary1>)

<Dictionary1>🡪 ID N = tlist.at(itr).getValuePart();| <Dictionary2> | €

<Dictionary2>🡪 { <Dictionary3> }

<Dictionary3>🡪<id\_const> : <Dictionary4>| €

<Dictionary4>🡪 <id\_const> <Dictionary6> | <Dictionary2> <Dictionary6> | <List2> <Dictionary6>

<Dictionary6>🡪 ,<Dictionary7> | €

<Dictionary7> 🡪<id\_const> : <Dictionary4>

DEL:

<Del>🡪 del ID N = tlist.at(itr).getValuePart(); <Del\_> 

<Del\_>🡪 [<exp>] N+="[]"; | € N=””

TRY\_EXCEPT\_FINALLY:

<Try>🡪 try : createScope() <body> destroyScope() <Except>

<Except>🡪<Finally> | except (createScope() <Exception>) : <body> destroyScope() <Except>

<Finally>🡪 finally : createScope() <body> destroyScope()

<Exception>🡪 exception ID 

LAMBDA:

<Lambda>🡪 lambda ID N = tlist.at(itr).getValuePart(); : <exp>