Function call, nested variable assignment, and inc\_dec

<var\_fn\_assign> -> ID<var\_fn\_assign1>|self<this\_super1>|grand . ID<this\_super1>

<var\_fn\_assign1> -> <array\_index><var\_fn\_assign2>|<var\_fn\_assign2>|<args><var\_fn\_assign3>

<var\_fn\_assign2> -> .<var\_fn\_assign4>|inc\_dec|<equals><assign1>

<var\_fn\_assign3> -> .<var\_fn\_assign4>|<array\_index><var\_fn\_assign2>|ϵ

<var\_fn\_assign4> -> ID<var\_fn\_assign1>

<this\_super> -> self .|grand . ID .

<this\_super1> -> <args>|. ID<var\_fn\_assign1>

Assignment right side

<assign1> -> ID<assign2>|<this\_super>ID<assign2>|<exp1>|inc\_dec<var><unpacked>

<exp1> -> <B><unpacked>|!<F><unpacked>|<const><unpacked>|PM<F><unpacked>|<obj\_dec>

<assign2> -> <array\_index><assign3>|<assign3>|<args><assign4>

<assign3> -> .<assign5>|<unpacked>|inc\_dec<unpacked>|as<type><unpacked>|<equals><assign1>

<assign4> -> .<assign5>|<unpacked>|as<type><unpacked>|<array\_index><assign3>

<assign5> -> ID<assign2>

<equals> -> =|compound\_assignment

Declaration

<dec> -> ID<dec1>

<dec1> -> :<type><dec2>

<dec2> -> =<dec3>|ϵ

<dec3> -> <assign1>|<array\_dec>

For loop

<for\_st> -> for(<decs>in<iterator>)<body>

<decs> -> ID :<type>|<des\_dec\_ref>

<des\_dec\_ref> -> [ ID ,<des\_dec\_ref1>] :<type>

<des\_dec\_ref1> -> , ID<des\_dec\_ref1>|ϵ

<iterator> -> <exp><range1>

<range> -> range<exp>:<exp>

<range1> -> <range>|ϵ

De-structured assignment and declaration

<des\_dec\_assign> -> [<des\_this\_super>

<des\_this\_super> -> ID<var\_is>|<this\_super><var><des\_dec\_assign2>

<des\_this\_super1> -> <var>|<this\_super><var>

<var\_id> -> <var4><des\_dec\_assign2>|<des\_dec\_assign1>

<des\_dec\_assign2> -> ,<des\_this\_super1><des\_dec\_assign2>|]=<exp>;

<des\_dec\_assign1> -> , ID<var\_id>|]<des\_dec\_assign3>

<des\_dec\_assign3> -> =<exp>;|:<type><des\_dec\_assign4>

<des\_dec\_assign4> -> ;|=<exp\_array>;

Variable

<var> -> ID<var1>

<var1> -> <array\_index><var2>|<var2>|<args><var4>

<var2> -> .<var>|ϵ

<var4> -> <array\_index><var2>|<args>.<var>|.<var>

<var3> -> .<var>|<array\_index><var2>

Operand of expression

<operand> -> ID<operand1>|<this\_super>ID<operand1>|<const>|inc\_dec<var>

<operand1> -> <array\_index><operand2>|<operand2>|<args><operand3>

<operand2> -> .<operand4>|inc\_dec|as<type>|ϵ

<operand3> -> .<operand4>|<array\_index><operand2>|ϵ

<operand4> -> ID<operand1>

Expression

<OE> -> <AE><OE1>

<OE1> -> or<AE><OE1>|ϵ

<AE> -> <RE><AE1>

<AE1> -> &&<RE><AE1>|ϵ

<RE> -> <E><RE1>

<RE1> -> relational<E><RE1>|ϵ

<E> -> <T><E1>

<E1> -> PM<T><E1>|ϵ

<T> -> <P><T1>

<T1> -> MDM<P><T1>|ϵ

<P> -> <F><P1>

<P1> -> ^<F><P1>|ϵ

<F> -> <operand>|<B>|!<F>|PM<F>

<B> -> (<OE>)

<unpacked> -> <P1><T1><E1><RE1><AE1><OE1>

<exp> -> <OE>|<obj\_dec>

Object declaration

<obj\_dec> -> make<obj\_dec1>

<obj\_dec1> -> str<obj\_dec2>|ID<obj\_dec2>|primitive\_type<array\_ref>

<obj\_dec2> -> <args>|<array\_ref>

<array\_ref> -> [<array\_ref1>

<array\_ref1> -> <exp>]<array\_ref2>|]<array\_ref3>

<array\_ref2> -> [<array\_ref\_exp>|ϵ

<array\_ref\_exp> -> <exp>]<array\_ref2>|]<array\_ref4>

<array\_ref4> -> [ ]<array\_ref3>|ϵ

<array\_ref3> -> [ ]<array\_ref3>|:<array\_dec>

Only indexing syntax [1][a + b]…

<array\_index> -> [<exp>]<array\_index1>

<array\_index1> -> [<exp>]<array\_index1>|ε

Only function arguments (a, b, 1 + 2,…)

<args> -> (<args1>

<args1> -> <exp><args2>|)

<args2> -> ,<exp><args2>|)

Data type

<type> -> str<dim>|primitive\_type<dim>|ID<dim>

<dim> -> []<dim>|ϵ

Body

<body> -> ;|<SST>|{<MST>}

Array declaration multi dim support (only semantic verification)

<array\_dec> -> [<array\_dec1>

<array\_dec1> -> <exp\_array><array\_dec2>|]

<array\_dec2> -> ,<exp\_array><array\_dec2>|]

<exp\_array> -> <exp>|<array\_dec>

Match case default

<match\_st> -> match(<exp>)<match\_body>

<match\_body> -> <case>|<default>|{<case\_default>}

<case> -> case<exp>-><body>

<default> -> default-><body>

<case\_default> -> <case><case\_default>|<default>|ϵ

Try except finally

<try\_st> -> try<body><except\_multi><finally>

<except> -> except ( ID :<type>) {<MST>}

<except\_multi> -> <except><except\_multi1>

<except\_multi1> -> <except><except\_multi1>|ϵ

<finally> -> finally{<MST>}|ϵ

Constants

<const> -> int\_const|float\_const|char\_const|str\_const|bool\_const

If else

<if\_st> -> if (<exp>)<body><else>

<else> -> else|ϵ

While loop

<while\_st> -> while (<exp>)<body>

Function declaration

<func\_dec> -> func <func\_dec1>

<func\_dec1> -> ID(<params> -><type\_void>{<MST>}

<params> -> ID:<type><params1>|)

<params1> -> , ID:<type><params1>|)

<type\_void> -> <type>|void

Increment decrement

<inc\_dec\_st> -> inc\_dec<var>;

Single-line and multi-line statement(s)

<SST> -> ID<SST1>|const<SST2>|self<this\_super1>;|grand . ID<this\_super1>;|inc\_dec<var>;|<if\_st>|

<match\_st>|<while\_st>|<for\_st>|<try\_st>|<return\_st>|continue\_break;|<des\_dec\_assign>

<SST1> -> <dec1>;|<var\_fn\_assign1>;

<SST2> -> <dec>;|<des\_dec>

<MST> -> <SST><MST>|ϵ

Return statement

<return\_st> -> return<return\_exp>

<return\_exp> -> <exp>;|;

Class definition

<class\_def> -> type ID<enhance>{<class\_body>}

<enhance> -> enhances ID<enhance1>|ϵ

<enhance1> -> , ID<enhance1>|ϵ

<class\_body> -> <cb\_am\_ps\_cn><class\_body>|ϵ

<cb\_am\_ps\_cn> -> access\_modifier<cb\_am\_ps\_cn1>|passive<am\_cn><adecs>|const<am\_ps><adecs>|<cdecs>

<cb\_am\_ps\_cn1> -> <cdecs>|passive<cn><adecs>|const<ps><adecs>

<am\_ps\_cn> -> access\_modifier<ps\_cn>|passive<am\_cn>|const<am\_ps>|ϵ

<ps\_cn> -> passive<cn>|const<ps>|ϵ

<am\_cn> -> access\_modifier<cn>|const<am>|ϵ

<am\_ps> -> access\_modifier<ps>|passive<am>|ϵ

<cdecs> -> <dec>;|func<fn\_init\_dec>|<des\_dec\_assign>

<fn\_init\_dec> -> <func\_dec1>|constructor (<params>) {<MST>}

<cn> -> const|ϵ

<am> -> access\_modifier|ϵ

<ps> -> passive|ϵ

<des\_dec> -> [ID<des\_dec1>

<des\_dec1> -> , ID<des\_dec1>|] :<type><des\_dec\_assign4>

<des\_dec\_const> ->

Abstract class definition

<abstract\_class\_def> -> abstract type ID<enhance>{<abstract\_body>}

<abstract\_body> -> abstract<abstract\_func><abstract\_body>|<am\_ps\_cn><adecs><abstract\_body>|ϵ

<adecs> -> <dec>;|<func\_dec>|<des\_dec\_assign>

<abstract\_func> -> func (<params>) -> <type\_void>;

Language structure

<lang> -> <imports><defs1>

<defs> -> <class\_def>|<abstract\_class\_def>|const<defs3>|<dec>;|<des\_dec>

<defs3> -> <class\_def>|<dec>;|<des\_dec>

<defs1> -> <defs><defs1>|func<lang1>|ϵ

<defs2> -> <defs><defs2>|<func\_dec><defs2>|ϵ

<lang1> -> <func\_dec1><defs1>|<main><defs2>

<main> -> main ( ) : void {<MST>}

<imports> -> <import><imports1>|ϵ

<imports1> -> <import><imports1>|ϵ

<import> -> import ID from str\_const ;