CFG	First	Follow	Selection
<start> → <defs> $mfunction main void () {<MST>} <defs>	F(start) → {fixed, abstract, class, $function, enum, interface}	Follow(start) -> { $ }	Sel(start) -> { fixed, abstract, class, $mfunction, enum , interface}
<defs> → <class_dec><defs> <defs> → <fn_dec> <defs> <defs> → <enum> <defs> <defs> -> <interface_dec><defs> <defs> → E	F(defs) → {fixed, abstract, class} F(defs) → {$function} F(defs) → {enum} F(defs) -> {interface} F(defs) → {E}	Follow(defs) -> { $mfunction, $ }	Sel(defs) -> { fixed, abstract, class } Sel(defs) -> { $function } Sel(defs) -> { enum } Sel(defs) -> { interface } Sel(defs) -> { $mfunction, $ }
<SST> → <while_St> <SST> → <for_st> <SST> → <return> <SST> → <switch> <SST> → <break> <SST> → <continue> <SST> → <if-elif-else> <SST> → <dec> <SST> → ID <SST1> <SST> → <inc_dec_op> ID; <SST1> -> <CB6> <SST1> → (<argu>) <SST2> <SST1> → <inc_dec_op>; <SST1> → .ID <assign1><CB6> <SST1> → [<exp>] <assign1><CB6> <SST2> → ; <SST2> → .ID <assign1><CB6>	F(SST) → {while} F(SST) → {for} F(SST) → {return} F(SST) → {switch} F(SST) → {break} F(SST) → {continue} F(SST) → {if} F(SST) → {take} F(SST) → {ID} F(SST) → {++, --} F(SST1) -> { =, CO } F(SST1) → { (} F(SST1) → {++, --} F(SST1) → {.} F(SST1) → { [} F(SST2) → {;} F(SST2) → {.}	Follow(SST) -> {while, for, return, switch, break, continue, if, take, ID, ++, --,}, break, F(cfbody)} Follow(SST1) → {while, for, return, switch, break, continue, if, take, ID, ++, --,}, break} Follow(SST2) → {while, for, return, switch, break, continue, if, take, ID, ++, --,}, break}	Sel(SST) -> { while } Sel(SST) -> { for } Sel(SST) -> { return} Sel(SST) -> { switch } Sel(SST) -> { break } Sel(SST) -> { continue } Sel(SST) -> { if } Sel(SST) -> { take } Sel(SST) -> { ID } Sel(SST) -> { ++, -- } Sel(SST1) -> { =, CO } Sel(SST1) -> { (} Sel(SST1) -> { ++, -- } Sel(SST1) -> { . } Sel(SST1) -> { [} Sel(SST2) -> { ; } Sel(SST2) -> { . }

<MST> → <SST><MST> | E	F(MST) → {while, for, return, switch, break, continue, if, take, ID, ++, --} F(MST) → {E}	Fol(MST) → { }, break}	Sel(MST) -> { while, for, return, switch, break, continue, if, take, ID, ++, -- } Sel(MST) -> { }, break }
<assign> → ID <assign1><assign_operator> <exp> <assign_list>	F(assign) → {ID}	Fol(assign) -> {present, super, ID, const, (, not } Fol(assign_operator) -> { present, super, ID, const, (, not }	Sel(assign) -> { ID }
<assign_operator> -> = | CO	F(assign_operator) -> { =, CO}		Sel(assign_operator) -> { =, CO }
<assign1> → . ID <assign1> <assign1> → (<argu>) . ID <assign1> <assign1> → [<exp>] <assign1> <assign1> → E	F(assign1) → { . } F(assign1) → { (} F(assign1) → { [} F(assign1) → { E }	Fol(assign1) -> {=, CO}	Sel(assign1) -> { . } Sel(assign1) -> { (} Sel(assign1) -> { [} Sel(assign1) -> { =, CO }
<assign_list> → , <assign> <assign_list> → ;	F(assign_list) → { , } F(assign_list) → { ; }	Fol(assign_list) -> {present, super, ID, const, (, not, while, for, return, switch, break, continue, if, take, ++, --,}, break own}	Sel(assign_list) -> { , } Sel(assign_list) -> { ; }
<dec> → take <dec0>	F(dec) → {take}	Fol(dec) -> {while, for, return, switch, break, continue, if, take, ID, ++, --, }, break, own, ,}	Sel(dec) -> { take }
<dec0> → <f_dt> <arr>: ID <dec1>	F(dec0) → {dt , ID}	Fol(dec0) -> { while, for, return, switch, break, continue, if, take, ID, ++, --, }, break, own, ,}	Sel(dec0) -> { dt, ID }
<dec1> → <dec_list> <dec1> → = <dec2> <dec_list> <dec_list> → , <dec0> | ;	F(dec1) → { , , ;} F(dec1) → { = } F(dec_list) → { ,, ;}	Fol(dec1) -> { while, for, return, switch, break, continue, if, take, ID, ++, --, }, break, own, ,}	Sel(dec1) -> { ,, ; } Sel(dec1) -> { = }

<dec2> → [<argu>] <dec2> → new ID (<argu>) <dec2> → <exp>	F(dec2) → { [} F(dec2) → { new } F(dec2) → { present, super, ID, const, (, not }	Fol(dec_list) -> { while, for, return, switch, break, continue, if, take, ID, ++, --, }, break, own, , } Fol(dec2) -> { ,, ; }	Sel(dec_list) -> { ,, ; } Sel(dec2) -> { [} Sel(dec2) -> { new } Sel(dec2) -> { present, super, ID, const, (, not }
<f_dt> → dt | ID <f_dt1> → <f_dt> <arr> <f_dt1> → void <arr> → [ ] <arr> → E	F(f_dt) → {DT, ID} F(f_dt1) → {DT, ID} F(f_dt1) → {void} F(arr) → {[]} F(arr) → {E}	Fol(f_dt) -> { [] } Fol(f_dt1) -> { (} Fol(arr) -> { :, (}	Sel(f_dt) -> { DT, ID } Sel(f_dt1) -> { DT, ID } Sel(f_dt1) -> { void } Sel(arr) -> { [] } Sel(arr) -> { :, (}
<exp> → <AE><exp'> <exp'> → OR <AE><exp'> | E <AE> → <RE><AE'>	F(exp) → { present, super, ID, const, (, not } F(exp') → {OR} F(exp') → {E} F(AE) → { present, super, ID, const, (, not }	Fol(exp) -> {], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break } Fol(exp') -> {], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break } Fol(AE) -> { OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(exp) -> { present, super, ID, const, (, not } Sel(exp') -> { OR } Sel(exp') -> {], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break } Sel(AE) -> { present, super, ID, const, (, not }

<AE'> → AND <RE><AE'> | E	F(AE') → {AND} F(AE') → {E}	Fol(AE') -> { OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(AE') -> { AND } Sel(AE') -> { OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
<RE> → <E><RE'>	F(RE) → {present, super, ID, const, (, not}	Fol(RE) -> { AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(RE) -> {present, super, ID, const, (, not }
<RE'> → ROP <E> <RE'> |E	F(RE') → {ROP} F(RE') → {E}	Fol(RE') -> { AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(RE') -> { ROP } Sel(RE') -> { AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
<E> → <T><E'>	F(E) → {present, super, ID, const, (, not}	Fol(E) -> { ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(E) -> { present, super, ID, const, (, not }
<E'> → PM <T><E'> <E'> → E	F(E') → {PM} F(E') → {E}	Fol(E') -> { ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(E') -> { PM } Sel(E') -> { ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
	F(T) → {present, super, ID, const, (, not}		Sel(T) -> { present, super, ID,

<T> → <F><T'>	F(T') → {MDM} F(T') → {E}	Fol(T) -> { PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break } Fol(T') -> { PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	const, (, not } Sel(T') -> { MDM } Sel(T') -> { PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
<T'> → MDM <F> <T'> | E	F(F) → { present, super, ID } F(F) → { const } F(F) → { (} F(F) → { not }	Fol(F) -> { MDM, PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(F) -> { present, super, ID } Sel(F) -> { const } Sel(F) -> { (} Sel(F) -> { not }
<F> → <TS> ID <O> <F> → const <F> → (<exp>) <F> → not <F>	F(O) → { [} F(O) -> { (} F(O) -> { . , E }	Fol(O) -> { MDM, PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(O) -> { [} Sel(O) -> { (} Sel(O) -> { ., MDM, PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
<O> → [<exp>]<O'> <O> → (<argu>)<O'> <O> → <O'>	F(O') -> { . } F(O') -> { E }	Fol(O') -> { MDM, PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }	Sel(O') -> { . } Sel(O') -> { MDM, PM, ROP, AND, OR,], ,,), present, super, ID, const, (, not, ;, while, for, return, switch, break, continue, if, take, ID, ++, --, }, break }
<O'> → . ID <O> <O'> → E	F(TS) -> {present} F(TS) -> {super} F(TS) -> {E}		Sel(TS) -> { present }

<TS> → present. <TS> → super. <TS> → E		Fol(TS) -> { ID }	Sel(TS) -> { super } Sel(TS) -> { ID }
<argu> → <exp> <argu1> <argu> → E <argu1> → , <exp> <argu1> <argu1> → E	F(argu) → { present, super, ID, const, (, not } F(argu) → { E } F(argu1) → { , } F(argu) → { E }	Fol(argu) -> {),] } Fol(argu1) -> {),] }	Sel(argu) -> { present, super, ID, const, (, not } Sel(argu) -> {),] } Sel(argu1) -> { , } Sel(argu1) -> {),] }
<while_st> → while (<exp>) {<MST>}	F(while_st) → {while}	Fol(while_st) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }	Sel(while_st) -> { while }
<for_st> → for(<assign> <exp>;<inc_dec_st>){<MST>}	F(for_st) → {for}	Fol(for_st) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }	Sel(for_st) -> { for }
<inc_dec_st> → ID<inc_dec_op> <inc_dec_st> → <inc_dec_op> ID <inc_dec_op> → ++ | --	F(inc_dec_st) → {ID} F(inc_dec_st) → {++, --} F(inc_dec_op) → {++, --}	Fol(inc_dec_st) -> {) } Fol(inc_dec_op) -> { ID, while, for, return, switch, break, continue, if, take, ID, ++, --,}, break,) }	Sel(inc_dec_st) -> { ID } Sel(inc_dec_st) -> { ++, -- } Sel(inc_dec_op) -> { ++, -- }
~~<acc_mod> → shared~~	~~F(acc_mod) → { shared }~~		

~~<acc_mod> → own~~ ~~<acc_mod> → E~~	~~F(acc_mod) → { own }~~ ~~F(acc_mod) → { E }~~		
<fixed> → fixed <fixed> → E <abstract> → abstract <abstract> → E <static> → static <static> → E	F(fixed) → { fixed } F(fixed) → { E } F(abstract) → { abstract } F(abstract) → { E } F(static) → { static } F(static) → { E }	Fol(fixed) -> { abstract, class, static, take, $function } Fol(abstract) -> { class, static, take, abstract, $function } Fol(static) -> { take, abstract, $function }	Sel(fixed) -> { fixed } Sel(fixed) -> { abstract, class, static, take, $function } Sel(abstract) -> { abstract } Sel(abstract) -> { class, static, take, abstract, $function } Sel(static) -> { static } Sel(static) -> { take, abstract, $function }
<class_dec> → <fixed> <abstract> class ID <interface><inheritance> {<CB>}	F(class_dec) → {fixed, abstract, class}	Fol(class_dec) -> { fixed, abstract, class, $function, enum, Fol(defs) }	Sel(class_dec) -> { fixed, abstract, class }
<inheritance> → child_Of ID <multi_ID> <inheritance> → E <multi_ID> → , ID <multi_ID> | E	F(inheritance) → {child_Of} F(inheritance) → {E} F(multi_ID) → {,} F(multi_ID) → {E}	Fol(inheritance) -> { { } Fol(multi_ID) -> { { }	Sel(inheritance) -> { child_Of } Sel(inheritance) -> { { } Sel(multi_ID) -> { , } Sel(multi_ID) -> { { }
<interface_dec> -> interface ID {<interface_body>} <interface_dec> -> E <interface_body> -> <dec> <interface_body>	F(interface_dec)-> {interface} F(interface_dec) -> {E} F(interface_body)-> { take }	Fol(interface_dec) -> { fixed, abstract, class, $function, enum, interface, $mfunction, $ } Fol(interface_body) -> { } }	Sel(interface_dec) -> { interface } Sel(interface_dec) -> { fixed, abstract, class, $function, enum, interface, $mfunction, $ } Sel(interface_body) -> { take }

<interface_body> -> $function ID <f_dt1> (<parameter>) ;<interface_body1> <interface_body1> -><interface_body> <interface> → implements ID <interface1> <interface> → E <interface1> → , ID <interface1> <interface1> → E	F(interface_body)-> { $function } F(interface_body1) -> {take, $function} F(interface) → { implements } F(interface) → E F(interface1) → { , } F(interface1) → { E }	Fol(interface_body1) -> { } } Fol(interface) -> { child_Of, { } Fol(interface1) -> { child_Of, { }	Sel(interface_body) -> {$function} Sel(interface_body1) -> {take, $function} Sel(interface) -> { implements } Sel(interface) -> { child_Of, { } Sel(interface1) -> { , } Sel(interface1) -> { child_Of, { }
<CB> → <CB0> <CB1> <CB0> → shared {<CB2>} | E <CB1> → own { <CB2>} | E <CB2> → <fixed><static><CB3> <CB2> → ID <CB4> <CB2> → E <CB3> → <dec><CB2> <CB3> → <abstract>$function ID <f_dt1>(<parameter>){<c_f_body>}<CB2>	F(CB) → {shared, own, E} F(CB0) → {shared} F(CB0) → {E} F(CB1) → {own} F(CB1) → {E} F(CB2) → {fixed, static, abstract, $function, take} F(CB2) → {ID} F(CB2) → {E} F(CB3) → {take} F(CB3) → {abstract, $function} F(CB4) → { (} F(CB4) → { . } F(CB4) → { [}	Fol(CB) -> { } } Fol(CB0) -> { own , } } Fol(CB1) -> { } } Fol(CB2) -> { } } Fol(CB3) -> { } } Fol(CB4) -> { } }	Sel(CB) -> { shared, own, } } Sel(CB0) -> { shared } Sel(CB0) -> { own , }} Sel(CB1) -> { own } Sel(CB1) -> { } } Sel(CB2) -> { fixed, static, abstract, $function, take } Sel(CB2) -> { ID } Sel(CB2) -> { } } Sel(CB3) -> { take } Sel(CB3) -> { abstract, $function } Sel(CB4) -> { (} Sel(CB4) -> { . }

<CB4> → (<argu>) <CB5> <CB4> → .ID<assign1> <CB6> <CB4> -> [<exp>]<assign1> <CB6> <CB5> -> <CB2> <CB5> -> . ID <assign1><CB6> <CB6> -> <assign_operator> <exp> <assign_list> <c_f_body> -> this. <c_f_body1> <c_f_body> <c_f_body> -> super. <c_f_body1><c_f_body> <c_f_body> -> <SST><c_f_body> <c_f_body> -> E <c_f_body1> -> ID <c_f_body2> <c_f_body2> -> (<argu>) <c_f_body3> <c_f_body2> -> = <exp>; <c_f_body2> -> .ID<assign1> <CB6> <c_f_body2> -> [<exp>] <assign1><CB6> <c_f_body2> -> ; <c_f_body3> -> .ID<assign1><CB6> <c_f_body3> -> ;	F(CB5) -> {fixed, static, abstract, $function, take, ID, E} F(CB5) ->{ . } F(CB6) → { =, CO } F(c_f_body) -> {this} F(c_f_body) -> {super} F(c_f_body) -> {while, for, return, switch, break, continue, if, take, ID, ++, --} F(c_f_body) -> {E} F(c_f_body1) -> {ID} F(c_f_body2) -> { (} F(c_f_body2) -> {.} F(c_f_body2) -> { [} F(c_f_body2) -> { = } F(c_f_body2) -> {;} F(c_f_body3) -> {.} F(c_f_body3) -> {;}	Fol(CB5) -> { } } Fol(CB6) -> {while, for, return, switch, break, continue, if, take, ID, ++, --,}, break} Fol(c_f_body) -> { } } Fol(c_f_body1) -> { this, super, while, for, return, switch, break, continue, if, take, ID, ++, -- } Fol(c_f_body2) -> {this, super, while, for, return, switch, break, continue, if, take, ID, ++, -- } Fol(c_f_body3) -> { this, super, while, for, return, switch, break, continue, if, take, ID, ++, -- }	Sel(CB4) -> { [} Sel(CB5) -> { fixed, static, abstract, $function, take, ID, own } Sel(CB5) -> { . } Sel(CB6) -> { =, CO } Sel(c_f_body) -> { this } Sel(c_f_body) -> { super} Sel(c_f_body) -> {while, for, return, switch, break, continue, if, take, ID, ++, --} Sel(c_f_body) -> { } } Sel(c_f_body1) -> { ID} Sel(c_f_body2) -> { (} Sel(c_f_body2) -> { . } Sel(c_f_body2) -> { [} SEl(c_f_body2) -> { = } Sel(c_f_body2) -> { ; } Sel(c_f_body3) -> { . } Sel(c_f_body3) -> { ; }

<continue> → continue ;	F(continue) → { continue }	Fol(continue) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }	Sel(continue) -> { continue }
<break> → break ;	F(break) → { break }	Fol(break) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }	Sel(break) -> { break }
<return> → return <return1>; <return1> → True <return2> <return1> → False <return2> <return1> → <exp> <return2> <return1> → <dec> <return2> <return2> → , <return1> <return2> → E	F(return) → { return } F(return1) → { True } F(return1) → { False } F(return1) → { present, super, ID, const, (, not } F(return1) → { take } F(return2) → { , } F(return2) → { E }	Fol(return) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break } Fol(return1) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break} Fol(return2) -> {while, for, return, switch, break, continue, if, take, ID, ++, --,}, break}	Sel(return) -> { return } Sel(return1) -> { True } Sel(return1) -> { False } Sel(return1) -> { present, super, ID, const, (, not } Sel(return1) -> { take } Sel(return2) -> { , } Sel(return2) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }
<parameter> → DT ID <parameter1> <parameter> -> E <parameter1> → ,DT ID <parameter1> <parameter1> → E	F(parameter) → { DT } F(parameter) -> {E} F(parameter1) → { , } F(parameter1) →{ E }	Fol(parameter) -> {) } Fol(parameter1) -> {) }	Sel(parameter) -> { DT } Sel(parameter) -> {) } Sel(parameter1) -> { , } Sel(parameter1) -> {) }
<fn_dec> → $function ID <f_dt1> (<parameter>) {<MST>}	F(fn_dec) → {$function}	Fol(fn_dec) -> { $mfunction, $, own, } }	Sel(fn_dec) -> { $function }

<enum> → enum ID = {<list>};	F(enum) → enum	Fol(enum) -> { Fol(defs) }	Sel(enum) -> { enum }
<list> → ID <list1> <list> → E <list1> → , ID<list1> <list1> → E	F(list) → { ID } F(list) → { E } F(list1) → { , } F(list1) → { E }	Fol(list) -> { } } Fol(list1) -> { } }	Sel(list) -> { ID } Sel(list) -> { } } Sel(list1) -> { , } Sel(list1) -> { } }
<fn_call> → ID (<argu>)	F(fn_call) → { ID }		Sel(fn_call) -> { ID }
<if-elif-else> → if (<exp>) {<MST>} <elif> <else> <elif> → elif (<exp>) {<MST>} <elif> <elif> → E <else> → else {<MST>} <else> → E	F(if-elif-else) → { If } F(elif) → { elif } F(elif) → { E } F(else) → { else } F(el) → { E }	Fol(if-elif-else) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break } Fol(elif) -> { else, while, for, return, switch, break, continue, if, take, ID, ++, --,}, break } Fol(else) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }	Sel(if-elif-else) -> { if } Sel(elif) -> { elif } Sel(elif) -> { else, while, for, return, switch, break, continue, if, take, ID, ++, --,}, break } Sel(else) -> { else } Sel(else) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break }
<switch> → switch (<exp>) {<case> <default>} <case> → case <const> {<MST>}<break><case> | E <default> → default {<MST>}	F(switch) → { switch } F(case) → { case } F(case) -> { E }	Fol(switch) -> { while, for, return, switch, break, continue, if, take, ID, ++, --,}, break } Fol(case) -> { default } Fol(default) -> { } }	Sel(switch) -> { switch } Sel(case) -> { case } Sel(case) -> { default } Sel(default) -> { default }

	F(default) → { default }		