<break_st> → break | ε

<continue_st> → continue | ε

<return_st> → return <x'>

<x'> → <oe> | ε

<n> → \n <n'>

<n'> → <n> | ε

<MST> → ε | <sst> <n> { <MST>

<if_else> if <oe> : <n> <body> .. <op_else>

<body> → ε | <sst> <n> | { <n> <MST> } <n>

<op_else> → else <n> <body> | ε

<while_st> → while <oe> : <n> <body>

<do-while> do <n> { <n> <MST> } <n>
while <oe> : <n>

for e1 : e2 : e3 <n> <body>

<e1> → <dec> | <Assign-st> | ε

<e2> → <oe> | ε

<e3> → id <e3'> | inc-dec <x> | ε

<e3'> → <assign-op> | inc_dec

<arr_def> → id <arr> | dt <arr>

<arr> → [<arr'>

<arr'> →] id <arr1> | ,] id <arr1>

<arr1> → = <arr1'> | ε

<arr1'> → new <arr2> | { <values>

<arr2> → dt [<oe> <arr3> | id [<oe> <arr3>

<arr3> →] { <values> } | , <oe>] { <values> }

<values> → <oe> <v>

<values'> → { <values> } <v1>

<v> → , <oe> <v> | ε

<v1> → , <values'> | ε

<P> → dt <P1> | id <P1> | ε

<P1> → id <P1'> <P3>

<P2> →] <P3> | ,] <P3>

<P3> → , <P'> | ε

<P'> → id <P1> | dt <p1>

<P1'> → [<P2> | ε

<type'> → [<t2> | ε

<t2> →] | ,]

<dec> → dt id <init> <list> | id id <init> <list>

<list> → , id <init> <list> | ε

<init> → = <oe> | ε

<fun_def> → <static> <type> fun id (<p>) <n> { <n> <MST> }

<static> → static | ε

<type> → dt <type'> | void | id <type'>

<type'> → [<t2> | ε

<t2> →] | ,]

<F'> → <Z> | ε

<oe> → <ae> <oe'>

<oe'> → OR <ae> <oe'> | ε

<ae> → <re> <ae'>

<ae'> → AND <re> <ae'> | ε

<re> → <e> <re'> | ε

<re'> → ROP <e> <re'> | ε

<e> → <T> <e'>

<e'> → PM <T> <e'> | ε

<T> → <F> <T'>

<T'> → MDM <F> <T'> | ε

<F> → <const> | (<e>) | !<F> | inc-dec<X>
| id <F'>

<Z> → . id <Z2> | [<oe> <W1> <Z1> | inc dec
| (<PL>) <Z2>

<Z1> → . id <Z> | inc_dec

<Z2> → id <Z> | [<oe> <W1> <Z1> | ε

<Fun-call> → id <Y>

<Assign_st> → <X> <Assign_op>
<oe>

<inc-dec_st> → <X> inc_dec | inc_dec<X>

<Y> → .id <Y> | (<PL>) <Y1> | <a7>.id<Y>

<Y1> → .id <Y> | <a7>.id<Y1> ε

<a1> → . [<oe> <W1>

$\langle X \rangle \rightarrow id \langle X_1 \rangle$

$\langle X_1 \rangle \rightarrow . id \langle X_2 \rangle \mid \varepsilon \mid \langle X_3 \rangle$

$\langle X_2 \rangle \rightarrow \langle X_4 \rangle \mid \langle X_1 \rangle \mid \varepsilon$

$\langle X_3 \rangle \rightarrow [\langle Oe \rangle \langle W1 \rangle \langle X_1 \rangle$

$\langle X_4 \rangle \rightarrow (\langle PL \rangle) \langle X_4' \rangle$

$\langle X_4' \rangle \rightarrow \langle X_3 \rangle \mid \langle X \rangle$

<SST> → <while-st> | <do-while> | <if-else> |
<for-st> | <return-st> | <break-st> |
<Continue-st> | &inc-dec id <X> |
id <SST1> | dt <SST2>
void func id (<P>) <h> { <h> <MST>
}

<SST1> → <W> | [<SST1'> | id <SST3> | <SST5>
<SST1'> →] <SST4> | ,] <SST1'>
<SST2> → id <init> <list> | [<SST2'> | <SST5>
<SST3> → = <SST3'>
<SST3'> → new id <Y> | <Oe>
<SST4> → id <arr1> | & <SST5>
<SST5> → func id (<P>) <h> { <h> <MST>
}

$\langle w \rangle \rightarrow .id \langle w \rangle \mid [\langle oe \rangle \langle w_1 \rangle \langle w_2 \rangle \mid$
$inc\_dec \mid \langle assign\_op \rangle \langle oe \rangle \mid$
$(\langle PL \rangle) \langle w_3 \rangle$
$\langle w_1 \rangle \rightarrow] \mid , \langle oe \rangle]$
$\langle w_2 \rangle \rightarrow .id \langle w \rangle \mid inc\_dec \mid \langle Assign\_op \rangle \langle oe \rangle$
$\langle w_3 \rangle \rightarrow .id \langle w \rangle \mid [\langle oe \rangle \langle w_1 \rangle \langle w_2 \rangle$
$\mid \varepsilon$

<defs> → public <defs'> | <defs> | ε

<defs'> → <class_def> <n> <defs> |
<struct_def> <n> <defs>

<struct_def> → struct id <n> { <n> <sbody>
}

<sbody> → <ac> <sbody1>

<sbody1> → <struct_def> | <static> <sbody'>
<n> <sbody>

<sbody'> → void Func id (<P>) <n> { <n> <MST>
} | dt <sb> | id <sb1>

<sb> → <type'> Func id (<P>) <n> { <n>
<MST> } | <arr> | id <init> <list>

<sb1> → <arr> | (<P>) { <n> <cons-body> }
| <type'> Func id (<P>) <n> {
<n> <MST> }

<ac> → public | private | ε

<cons-body> → <Assign-st> <n> <cons-body>
| ε

<class_def> → <ab_seal> <static> class
id <inh> <n> { <n>
<cbody> }

<cbody> → <acc_mod> <static> <cbody'>
<n> <cbody> | ε

<cbody'> → <ab-seal> <C> | id <C1> | dt <C2>

<C> → dt <C'> | id <C3> | void func id (<P>)
{ <n> <MST> }

<C'> → id <init> <list> | <type'> func id (<P>) { <n>
<MST> }

<C1> → <arr> | (<P>) { <n> <cons_body> }
| id = new id <Y>

<C2> → <arr>

<C3> → <type'> func id (<P>) { <n> <MST> }
| id <init> <list>

<inh> → : id | ε

<ab_seal> → abstract | sealed | ε

<static> → static | ε

<S> → public class id <inh> { <n>
<S1> | class id <inh> { <n> <S1>

<S1> → public <S2> | private <static> <cbody>
<n> <cbody>
| protected <static> <cbody> <n> <cbody>
| <static> <cbody> <n> <cbody>

<S2> → static <S3> | <cbody> <n> <cbody>

<S3> → void <S4> | dt <C> | id <C3>
| abstract <S5> | ~~sealed <S5>~~

<S5> → dt <C> | id <C3> | void func id
(<P>) { <n> <MST>

<S4> → main() { <n> <MST> } <n> <cbody> }
<n> <defs> | func id (<P>) {
<n> <MST> }

<ab_seal> → Abstract cat = "Abstract"
cat
| sealed cat = "Sealed"
| ε cat = "General"

<inh> → : ID N = Token[index].vp

T = lookup_MT(N, out cat)

if(T == null)
{" undeclared Identifier!"}

if (T == "struct"

"Class can't inherit
from struct".

if (T == "class" &&
cat = "sealed")
{" sealed class can't
be inherited"}
PrN ≠ N

<Dec> → DT T.type = token[i].vp ID N.name = token[i].vp

if (insert-fT(N,T) == false) redeclaration

<INIT>T <LIST>T

<INIT>T → = <QE>T1 if (compatibility (T, T1, "=")) { "Type match error" }

<LIST>T → , id N.name = token[i].vp

if (insert-fT(N,T) == false) { "redeclaration" }

<INIT>T <LIST>T

<return-st> → return <r> Type

<r'> → <oe> Type | ε

<Accessmod> → public AM = "public"
| private AM = "private"
| protected AM = "protected"
| ε AM = "private"

<Static> → static TM = "static"
| ε

$<P>_{P_s} \rightarrow$ dt T = token[index].vp $<P1>_{P,T}$

| id N = token[index].vp, T = lookup-MT (N, outcat, outref)
if (T == null) {"undeclared"}

$<P1>_{P,T}$
| $\varepsilon_{P,T}$

$<P1>_{P,T} \rightarrow$ id N = Token[index].vp

if (! insert-FT(N, T)) {"redeclaration"}
P = T

$<P1'>$ $<P3>_{P1}$ P += ", " P1

$<P2> \rightarrow$] $<P3>$ | ,] $<P3>$

$<P3> \rightarrow$, $<P'>_{P}$ | ε

$<P'>_{P,T} \rightarrow$ id N = token[index].vp, T = lookup-MT (N, outcat, null)
if (T == null) {"undeclared"}

$<P1>_{P,T}$

| dt T = token[index].vp $<P1>_{P,T}$

AM = "public"

<S> → public class T = "class"

id N = token[index].vp <inh> print

CT_ref = create_DT()

if((insert_MT(N, T, Cat, Prnt, Ref))

{

"Redeclaration" }

{ create_scope() <h> <S1> CT_ref

<S1> → public AM = token[index].vp <S2> CT_ref, AM, T

| private AM = token[index].vp <static> TM

<C_body'> <h> <Cbody> CT_ref
AM, TM, CT_ref

| protected AM = token[index].vp <static> TM

<Cbody'> <h> <Cbody> CT_ref
AM, TM, CT_ref

| <static> <Cbody'> <h> <Cbody> CT_ref
TM AM, TM, CT_ref

<S2> → static TM = "static" <S3>
AM, TM, CT-ref

| <cbody'> <h> <cbody>
AM, TM, CT-ref CT-ref

<S3> → void T = "void" <S4>
T, AM, TM, CT-ref

| dt T = Token[index].vp <C'>
T, AM, TM, CT-ref

| abstract cat = "abstract" <S5>
CAT, AM, TM, CT-ref

| id N = token[index].vp <E3>
TM, N, T, AM, CT-ref

T = lookup MT (N, out cat)

```
if (T == null)
{ "undeclared " }
if ( T == "struct")
{ "struct can't use" }
}
```

if(!insert_ST(N,T,TM,AM,CT_ref))
{"redeclaration"};

<S4> → main N=Token[index].vp (Create_Scope()

) { <n> <MST> } Scope_destroy()
CT_ref

<n> <cbody> } Scope_destroyd()
CT_ref

| func id N=Token[index].vp

(Create_Scope() <P> if(!insert_ST(N,
P
P+"->"T,AM,t
CT_ref))
{"redeclr
}

) { <n> <MST> } destroyScope()
CT_ref

<S5> → dt T=Token[index].vp <C'>
T,AM,TM,CT_re

|void T="void" func id N=Token[index]vp

(CreateScope() <P> if(!insert_ST(N,P+"->"T,
P
AM,TM,CT_
{"redeclaration"};

) { <n> < MST > } destroyScope()
CT_ref

id N.token[index].vp <(3>
TM, N, T, AM, CT-ref

T = lookup-MT (N ; out cat)

if (T == null)
{ "undeclared Class" }
if (T == "struct")
{ "
}

if (T ==

$\underset{T}{\langle OE \rangle} \rightarrow \underset{T_1}{\langle AE \rangle} \quad \underset{T_1, T}{\langle OE' \rangle}$

$\underset{T_1, T}{\langle OE' \rangle} \rightarrow OR \quad op = token[index].vp \quad \underset{T_2}{\langle AE \rangle} \quad T_3 = Compatibility(T_1, T_2, op)$

if (T_3 == Null) { "Type error" }

$\underset{T_3, T}{\langle OE' \rangle} \mid \varepsilon$

$\underset{T}{\langle AE \rangle} \rightarrow \underset{T_1}{\langle RE \rangle} \quad \underset{T_1, T}{\langle AE' \rangle}$

$\underset{T_1, T}{\langle AE' \rangle} \rightarrow AND \quad op = token[index].vp \quad \underset{T_2}{\langle RE \rangle}$

$T_3 = Compatibility(T_1, T_2, op)$

if (T_3 == null) { "Type error" }

$\underset{T_3, T}{\langle AE' \rangle} \mid \varepsilon$

$\underset{T}{\langle RE \rangle} \rightarrow \underset{T_1}{\langle E \rangle} \quad \underset{T_1, T}{\langle RE' \rangle}$

$\underset{T_1, T}{\langle RE' \rangle} \rightarrow ROP \quad op = token[index].vp \quad \underset{T_2}{\langle E \rangle}$

$T_3 = Compatibility(T_1, T_2, op)$

if (T_3 == Null) { "Type error" }

$\underset{T_3, T}{\langle RE' \rangle} \mid \varepsilon$

$\underset{T}{\langle E \rangle} \rightarrow \underset{T_1}{\langle T \rangle} \quad \underset{T_1, T}{\langle E' \rangle}$

$\underset{T_3, T}{\langle E' \rangle} \rightarrow PM \quad op = token[index] \underset{T_2}{\langle T \rangle}$

$T_3 = Compatibility(T_1, T_2, op)$
{ if ($T_3 == null$)
{ ("type error") }

$\underset{T_3, T}{\langle E' \rangle} \mid \varepsilon$

$\underset{T}{\langle T \rangle} \rightarrow \underset{T_1}{\langle F \rangle} \quad \underset{T_1, T}{\langle T' \rangle}$

$\underset{T_1, T}{\langle T' \rangle} \rightarrow MDM \; op = token[index].up \underset{T_2}{\langle F \rangle}$

$T_3 = Compatibility(T_1, T_2, op)$
{ if ($T_3 == null$)
{ "type error" }

$\underset{T_3, T}{\langle T' \rangle} \mid \varepsilon$

<F> → <const>
T, CT-ref

| (<e>)
T

| ! op=token[i].vp
<F>
T1, CR

T2 = Compatibility(op, T1)

if (T == null)
{ Type error
T = T2

| index op=token[index].vp
<X>
T1, CT-ref

T2 = Compatability(op, T1)
if(T == null)
{ " Type error "
return T = T2

| id N=token[index].vp
<F'>
T, N, CT-ref

<F'> → <Z>
I, N, CT-ref
T, N, CT-ref

| ε

<X> → id N=token[index].vp <X1>
T, cT-ref N, cTref, flag

<X1> → . id N1=token[index].vp <X2> | ε
N, flag, cT-ref N1, flag, cTref

```
if (flag == 0)
{ T = lookup_MT(N)          flag = 1
  if (T == null)
  { T = lookup_FT(N)
    if (T == null)
    { " undeclared " }
    }
  elseif (T == "class")
  { isStaticM = 1 }
    cT-ref = ref
else{
  T = lookup_all_DT(cTref, N)
  if (T == null){ "undeclared" }
  if (AM == private) { "Can't access" }
  if (IM != isStaticM)
  { " Can't access" }
}
```

<X2> → <X4> | <X1> | ε
N, flag, CT-ref

<X4> → (<PL>) <X4'>
N, flag, CT-ref P, N, CT-ref N, flag, CT-ref

T = lookup-fun-DT (N, P, ct-ref)
if (T == null) { "undeclared" }

<X4'> → <X3> | <X>
N, flag, CT-ref T, CT-ref

if (isStatic != static) || (AM == "Private") { "Can't Access" }

<Z> → .

N, T, CT-ref
flag, is-static-M

if (flag == 0)
{ T = lookup_MT(N)
if (T == null)
{ T = lookup_ST(N)
if (T == null)
{ "undeclared" }
else if (T == "class")
{ is_static_M = "static" }
CT_ref = ref
}
else{ T = lookup-att-ST
(N, CT_ref)
{if(T == null)
{ "undeclared" }
if(AM == "private")
{ " can't access" }
if(TM != "static")
{ " can't access" }
}

id N = token.value <Z>

T, N, CT-ref,
flag, is static M

| [<oe> <wi> <Z> </wi> </oe>

inc_dec

op = token[index]

T_1 = lookup_ST(N,)

if (T == null)

{"undeclared"}

T = Compatibility (op, T_1)

if (T == null)

{ "type not matched" }

I (<PL>) <Z>

P, N, CT-ref flg, N, CT-ref

T = lookup_fn_DT(N, p, CTref)

if (T == null)

{ "undeclared" }

if (is_static_M != "static"

|| AM == "private")

{ "Can't Access" }

<Z1> →

N, T_1, CT-ref, flg

is_static_M

if (flg == 0)

{ flg = 1

T = lookup_MT (N)

if (T == null)

{ T = lookup_ST (N)

if (T == null)

{" undeclared"}

else if (T == "class")

```
{ if(Static_M == "static" }
    { T_ref = ref
    }
def { T = lookup_att_ST(N, T_ref)
      if (T == null)
      { "undeclared" }
        if ( AM == "private")
        { "can't access" }
        if ( TM != "static")
        { "can't access" }
    }
```

1. inc-dec

op = token[index].vp

<fun_def> → <static> <type> func
AM, CT_ref TM T

id N = token[index].vp

(create_Scope( ) <P>
                   P

if (! insert_DT(N, P.t "→" + T, AM, TM, CT_ref))
{" function redeclaration "}

) <h> { <h> <MST>
          CT_ref

} destroy_Scope( )

<static> → static TM = "static"
TM
| ε TM = null

<type> → dt T = token[index].vp <type'>
T

| void T = "void"
| id N = token[index].vp
T = lookup_MT (N,)
if (T == null)
{" undeclared "}

```
else if( T == "struct" ||
         cat == "sealed")
{
    "access denied";
}
```

$\langle defs \rangle \rightarrow$ public AM="public" $\langle defs' \rangle_{AM}$

$\mid \langle defs' \rangle_{AM} \mid \varepsilon$

$\langle defs' \rangle_{AM} \rightarrow \langle class\_def \rangle_{AM} \langle n \rangle \langle defs \rangle$

$\mid \langle struct\_def \rangle_{AM} \langle n \rangle \langle defs \rangle$

<Assign_st>.T → <X>.T1, CT_ref <Assignop>.op

<oe>.T2, CT_ref

T = Compatibility (T1, T2, op)

if (T == null)
{ "type not matched" }

<inc_dec_st>.T → <X>.T1, CT_ref ^inc_dec.op

T = compatibility (T1, op)
if (T == null)
{ "type not matched" }

| inc_dec.op <X>.T1, CT_ref

T = compatibility (T1, op)
if (T == null)
{ "type not matched" }

<fun_call> → id N = token[index].vp
T, CT-ref

<Y>
N, T, CT-ref
flag, is_static_M

<for_st> → for create_Scope() <e1> :
T

<e2> : if(T1 != bool)
T1 { "type error" }

<e3> <b> <b-dy> scope_dep
T2 ;{}

if (T1 = T2)
{ "error" }

<e1> → <dec> | <Assign_st>
T T

| ε

<e2> → <oe> T | ε
T

<e3> → id N = token[index].vp
T

<e3> T = lookup_FT(N, T, Scope)
T, T if(T == null)
"Undeclared"

| inc_dec <X>
op = token[index].vp T, CT_ref

T = compatibility (T, op)

if (T == null)
"type not matched"

<if_else_st> → if create_scope()
CT_ref
<oe> : <n>
T

if (T != bool)
{ "error" }

<body> destroyScope()
<op-else>

<op-else> → else createScope() <n>
CT_ref
<body> destroyScope()
CT_ref
| ε

if(T != "bool")

↑{"error"}

<while> → while <oe> : <n>
ct-ref T

create scope() <body> destroy
ct-ref scope()

<do_while> → do <n>{ create scope()
ct-ref

<n> <MST> }< destroy scope()
ct-ref

<n> while <oe> : <n>
T

if(T != "bool")
{ "error" }

Date: ____________

struct def → <ac> struct TM="struct" id N=t[i].vp
T, AM

{create.scope(); <sbody> ct_ref=create.DT(); scope destroy()
CT_ref if (insert.ST(N, T, cat, Prnt, ref))
→ "Redeclaration" }};

sbody → <acs> <static> TM="static" <sbody'>
T, AM, N AM, TM, CT_ref

<n> <sbody>
AM, TM, CT_ref

sbody' → void T="void" func id N=t[i].vp (<P>) P
create.scope(); destroy scope();

{ <MST> → CT_ref }
create.scope(); destroy scope();

dt <sb> | id N_2=t[i].vp <sb1>
T=t[i].vp CT_ref CT_ref
if(!insert_DT (N, T, TM, AM, CT_ref) → redeclaration

T=lookup.MT(N, ... cat)
if(T==Null) → "undeclared"
if(T="struct" → can't access

sb → func | id N_3=t[i].p (<P>) P {
create scope(); → scope create();
destroy scope()

<MST> scope destroy ← }
MST

<ans> | id N_4=t[i].vp <int> <list>
T T

Date:

st → const | (CB) { → create scope
<const_body> destroy scope } destroy scope

func id N=t[i].vp (CB P destroy scope) { check scope
create scope

<MST> MST } destroy scope "

AM = "Public" / "Private"
ac → Public | Private | ε = "General"
AM = t[i].vp

W → id <W> | [<oe> <W1> <W2> | inc_dec | <assi_op> <oe> |
(<PL>) <W3>

W1 →] | , <oe>]

W2 → id <W> inc_dec | <Arith_op> <oe>

W3 → id <W> | <oe> W1 W2 | ε

Date: _____

W → • id N=tokenUnderDot <W>
N, CT_ref, flag)

if (flag == 0)
{ T = lookup_MT(N)

if (T == null)
{ T = lookup_ST(N)
if (T == null) {
"undeclared"
}
else if (T = "class")
{static M=1 }
CT_ref = ref

else {
T = lookup DT (CT_S, N)
if (T = Null) → "is declared"
if (M = private → "can't access"
if (T.M != is static.M)
{"can't access" }

Date:

$W_1 \rightarrow$] | ? <oe>]

away closed ? oe away closed

$W_2 \rightarrow$ id <W>

N = t[i].vp w

| inc_dec

N ?T

inc ++

| assign_op <oes

$W_3 \rightarrow$ id N = t[i].vp <W3> | [<oes> <W1> <W2>

scope vars

| ε cat = "keyword"