UNIVERSITY OF KARACHI

NAME:

ASAD AHMED SHEIKH (EB19103017)

BILAL FAYYAZ (EB19103021)

SYED SIRAJ US SALEEKEN (EB19103129)

CLASS: BSSE 3RD YEAR (5TH Semester)

SUBJECT: COMPILER CONSTRUCTION

SECTION: “B”

LAB TEACHER: MISS ARISHA

DEPARTMENT: COMPUTER SCIENCE

COURSE CODE: CSSE-501

LAB FILE INCUDES:

* LANGUAGE SPECIFICATION (PART I)
* CFG’S (PART II)
* FIRST N FOLLOW (PART III)
* LL1 (PART IV)
* ATTRIBUTED GRAMMAR (PART V)

PART I

LANGUAGE SPECIFICATION:

|  |  |
| --- | --- |
| CLASS #1: DATATYPES | SYNTAX |
| int 🡪 for integers | int a =3; |
| string 🡪 for string | string b= “asad”; |
| float 🡪 for float | float f=3.33; |
| double 🡪 for double | double d=294; |
| char 🡪 for char | char h=’A’; |
| bool 🡪 for bool | bool s=true; |
|  |  |
| CLASS #2: ACCESS MODIFIER | SYNTAX |
| protected 🡪 protected | protected class A: end |
| public 🡪 public | public class B: end |
| private 🡪 private | private class C: end |
|  |  |
| CLASS #3: CLASS | SYNTAX |
| class 🡪 class | class A: end |
|  |  |
| CLASS #4: MAIN | SYNTAX |
| main 🡪 main | pub stat void main(): end |
|  |  |
| CLASS #5: VOID | SYNTAX |
| void 🡪 void | pub void func(): end |
|  |  |
| CLASS #6: STATIC | SYNTAX |
| static 🡪 static | public static void A(): end |
|  |  |
| CLASS #7: GOTO | SYNTAX |
| goto 🡪 goto | void checkEven(int num):  goto even end |
| CLASS #8: THIS | SYNTAX |
| this 🡪 this | pub func():  this.a end |
| CLASS #9: BASE | SYNTAX |
| const 🡪 const | const |
|  |  |
| CLASS #10: FOR | SYNTAX |
| foreach 🡪 foreach | foreach (int i in b):  a=b+c-d\*e/f%h  end |
|  |  |
| CLASS #11: IF | SYNTAX |
| if 🡪 if | if(a>3): end |
|  |  |
| CLASS #12: ELSE | SYNTAX |
| else 🡪 else | else: end |
|  |  |
| CLASS #13: SWITCH | SYNTAX |
| switch 🡪 switch | switch(a):  case 1:  break  end |
| CLASS # 14:CASE | SYNTAX: |
| Case 🡪 case | Same example as shown in upper class |
| CLASS #15:RETURN | SYNTAX |
| return 🡪 return | void f(int num):  return 1 end |
| CLASS #16: BREAK | SYNTAX |
| break 🡪 break | switch(a):  case 1:  break  end |
| CLASS #17: CONTINUE | SYNTAX |
| continue 🡪 continue | case 2:  continue  end |
| CLASS #18: NEW | SYNTAX |
| new 🡪 new | class MyClass:  MyClass MC = new MyClass  End |
| CLASS #19: WHILE | **SYNTAX** |
| while 🡪 while | **whi(true):**  **end** |
| OPERATORS |  |
| CLASS #20:INC\_DEC | **SYNTAX** |
| ++ 🡪 ++ | **a++** |
| \_ \_ 🡪 \_ \_ | **a--** |
| CLASS #21: RELATIONAL OPRTR | **SYNTAX** |
| < 🡪 < | **a<3** |
| >🡪 > | **a>3** |
| <= 🡪 =< | **b<=3** |
| >= 🡪 => | **b>=3** |
| != 🡪 != | **5!=3** |
| == 🡪 == | **4==4** |
| CLASS#22: MULTIPLYDIVIDEMODE MDM | **SYNTAX** |
| \*🡪 \* | **5\*6** |
| / 🡪 / | **3/2** |
| % 🡪 % | **4%2** |
| CLASS #23:PM PLUSMINUS | **SYNTAX** |
| + 🡪 + | **4+2** |
| * 🡪 - | **3-2** |
| CLASS #24: ASSIGNMENT OPRTR | **SYNTAX** |
| = 🡪 = | **a =3;** |
|  |  |
| CLASS #25: AND && | **SYNTAX** |
| && 🡪 && | **5 && 3** |
| CLASS #26: || OR | **SYNATAX** |
| || 🡪 || | **6 || 4** |
| CLASS #27: ! NOT | **SYNTAX** |
| ! 🡪 ! | **!a** |
| PUNCTUATOR |  |
| CLASS #28: . doT | **SYNTAX** |
| . 🡪 . | **a.b** |
| CLASS #29: | **SYNTAX** |
| , 🡪 , | **A,b,c** |
| CLASS #30: | **SYNTAX** |
| : 🡪 : | **a:b** |
| CLASS #31: | **SYNTAX** |
| ; 🡪 ; | **Abc;** |
| CLASS #32: | **SYNTAX** |
| ( 🡪 ( | **print(abc);** |
| CLASS #33: | **SYNTAX** |
| ) 🡪 ) | **print(abc);** |
| CLASS #34: | **SYNTAX** |
| { 🡪 { | **Void func(){}** |
| CLASS #35: | **SYNTAX** |
| } 🡪 } | **Void func(){}** |
| CLASS #36: | **SYNTAX** |
| [ 🡪 [ | **Matrix[2]** |
| CLASS #37: | **SYNTAX** |
| ] 🡪 ] | **M[3]** |
| CLASS #38: |  |
| \n 🡪 \n |  |
| \b 🡪 \b |  |
| \t 🡪 \t |  |
| CLASS #39: ABSTRACT | **SYNTAX** |
| abstract 🡪 abstract | **abstract class A: end** |
| CLASS #40: override | **SYNTAX** |
| override 🡪 override | **public static override fn(): end** |
| CLASS # 41:VIRTUAL | **SYNTAX** |
| virtual 🡪 virtual | **public static virtual fn(): end** |
| CLASS #42: :: | **SYNTAX** |
| :: 🡪 inheritance | **Class A::B: end** |
| CLASS #43: IN FOREACH | **SYNTAX** |
| in 🡪 in | foreach (int i in b):  a=b+c-d\*e/f%h  end |
| CLASS #43:end | SYNTAX |
| end 🡪 end | class A: end |
|  |  |
| CLASS #44:INTERFACE | SYNTAX |
| interface 🡪 interface | interface A: end |
| CLASS #45: COMPOUND ASSIGNMENT | SYNTAX |
| += 🡪 Comp\_assi | a+=2 |
| -= 🡪 Comp\_assi | b-=4 |
| \*= 🡪 Comp\_assi | c\*=21 |
| /= 🡪 Comp\_assi | d/=3 |
| %= 🡪 Comp\_assi | e%=14 |

PART II: CONTEXT FREE GRAMMAR (CFG’S)

# DECLARATION

DEC -> DT id INIT LIST.

DT -> double

| bool

| int

| var

| string

| char.

LIST -> ;

| , id INIT LIST.

INIT -> = IT.

IT -> id INIT | CONST.

CONST -> int

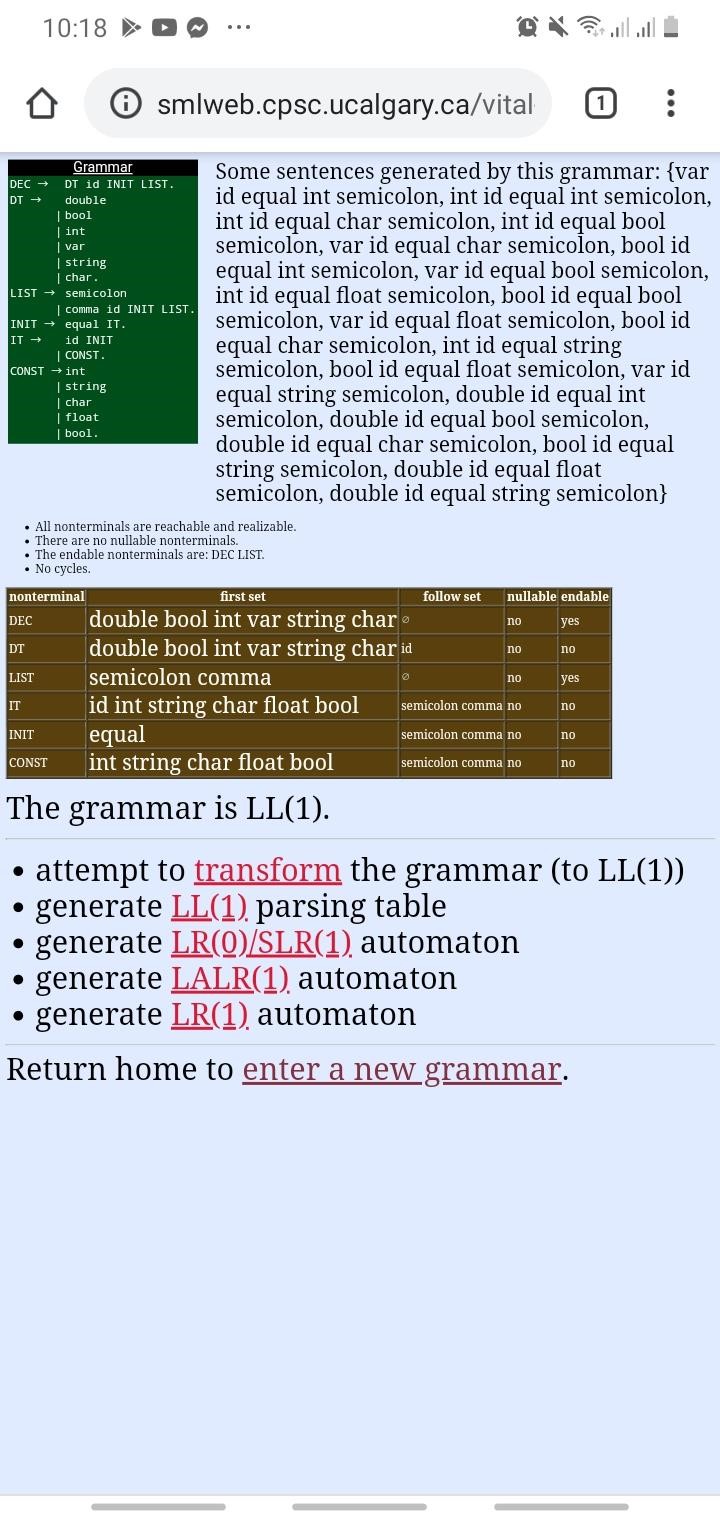
| string

| char

| float | bool.

PART III & IV

FIRST AND FOLLOW & LL1 GRAMMAR



# While\_loop

WHILE\_ST -> while ( COND ) BODY.

COND-> ID LF. LF-> ROP ID\_CONST

| .

ID\_CONST-> ID | CONST.

BODY-> ;

| SST

| { MST }.

SST->WHILE\_ST

| FOR\_ST

| IF\_ELSE | ARR

| OBJECT

| OBJ\_CALL

| FN\_CALL

| FN

| DEC. MST-> SST MST

| .

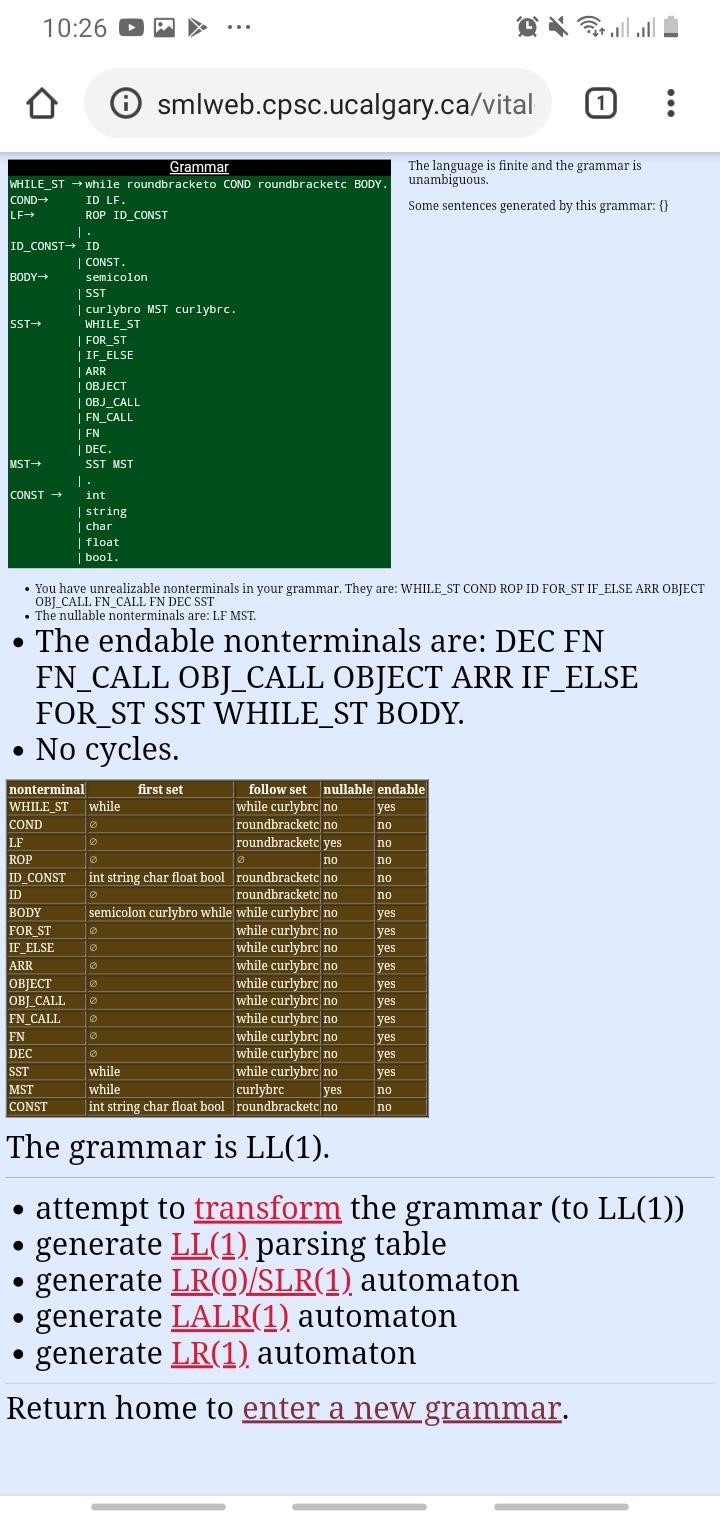
CONST -> int

| string

| char

| float

| bool.



# IFELSE CFG

IF\_ELSE -> if ( COND ) BODY OELSE.

OELSE -> else BODY. COND-> ID LF. LF -> ROP ID\_CONST

| .

ID\_CONST-> ID | CONST.

BODY-> ;

| SST

| { MST }.

SST->WHILE\_ST

| FOR\_ST

| IF\_ELSE | ARR

| OBJECT

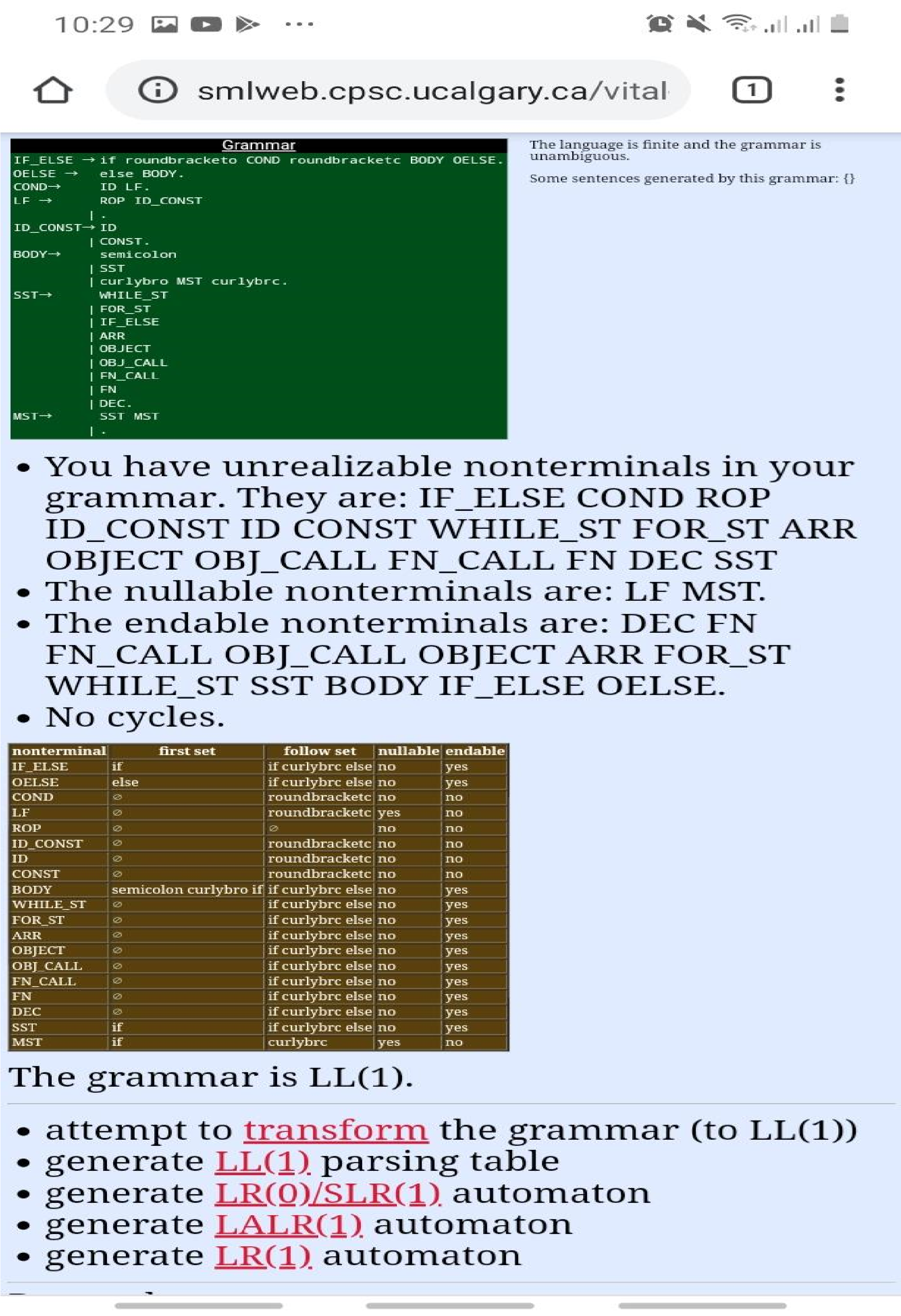
| OBJ\_CALL

| FN\_CALL

| FN

| DEC. MST-> SST MST

| .



INTERFACE

INT -> interface id { BODY }.

BODY-> ;

| SST

| { MST }.

SST->WHILE\_ST

| FOR\_ST | IF\_ELSE | ARR

| OBJECT

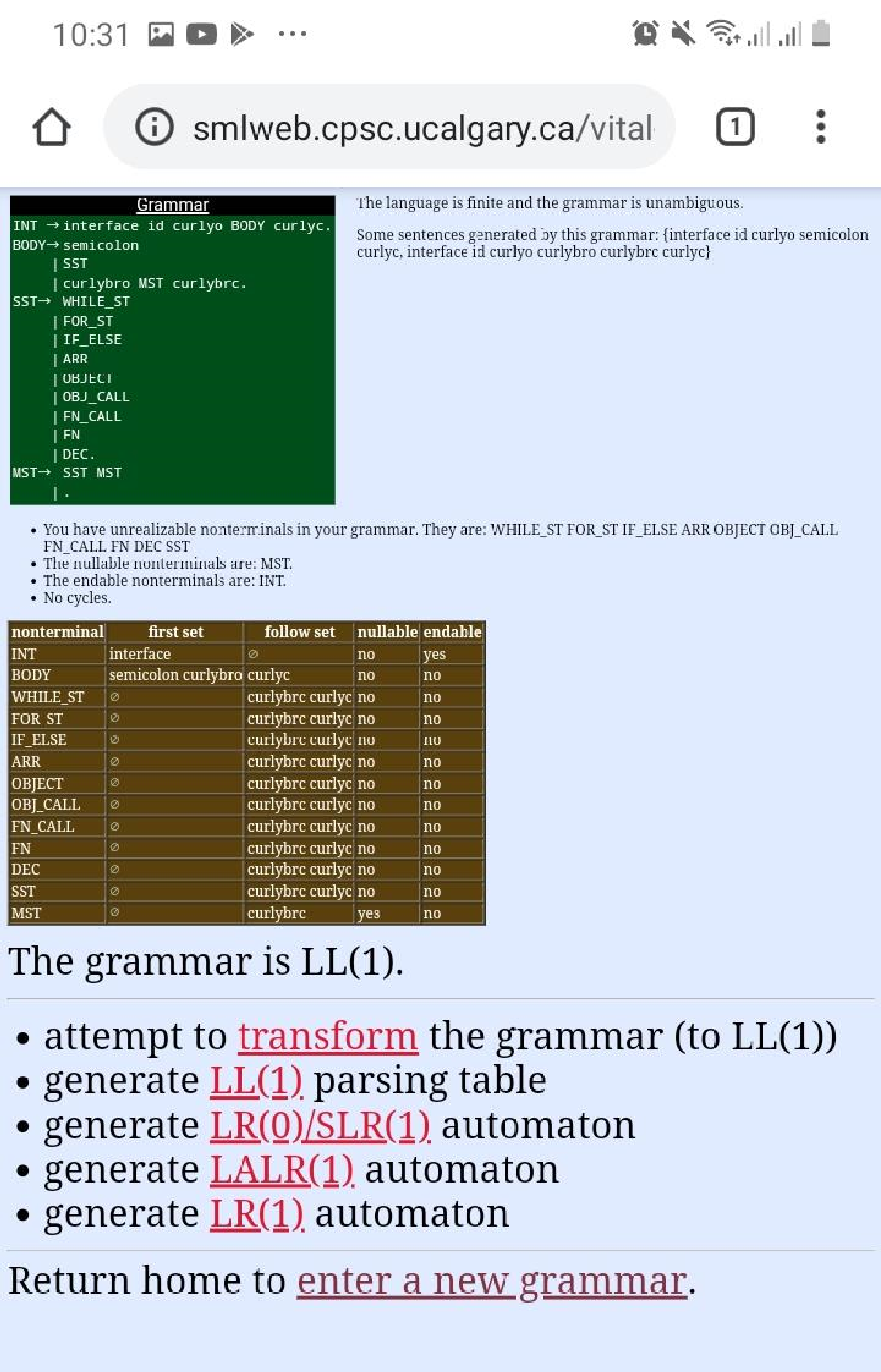
| OBJ\_CALL

| FN\_CALL

| FN

| DEC. MST-> SST MST

| .



# OBJECT DECLARATION

OBJECT-> CLASS BR OBJ\_NAME LIST2.

CLASS-> id.

OBJ\_NAME -> id. BR-> [ SIZE ]

| .

SIZE-> INT\_CONST

| .

ID\_CONST-> id | CONST.

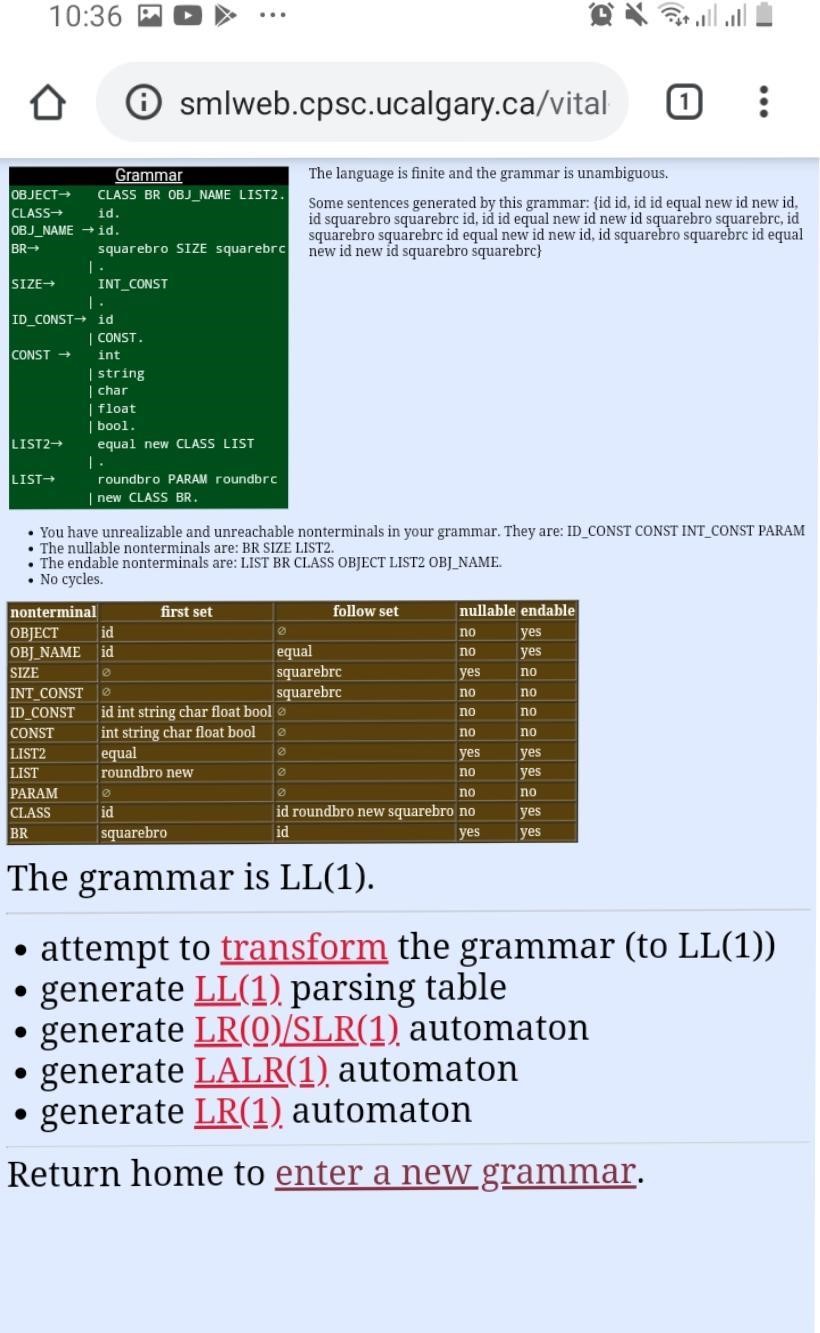
CONST -> int

| string

| char

| float | bool. LIST2-> = new CLASS LIST

| .

LIST-> ( PARAM ) | new CLASS BR. 

# OBJECT CALL CFG

OBJ\_CALL-> OBJ\_NAME BR VALUE LIST3 ;.

VALUE -> dot VALUE2

| . VALUE2 -> id

| FN\_CALL VALUE | OBJECT VALUE. LIST3 -> = LIST4

| .

OBJ\_NAME -> id. BR-> [ SIZE ]

| .

SIZE-> INT\_CONST

| .

ID\_CONST-> id | CONST.

CONST -> int

| string

| char

| float | bool.

LIST4-> new CLASS ( PARAM )

| CONST

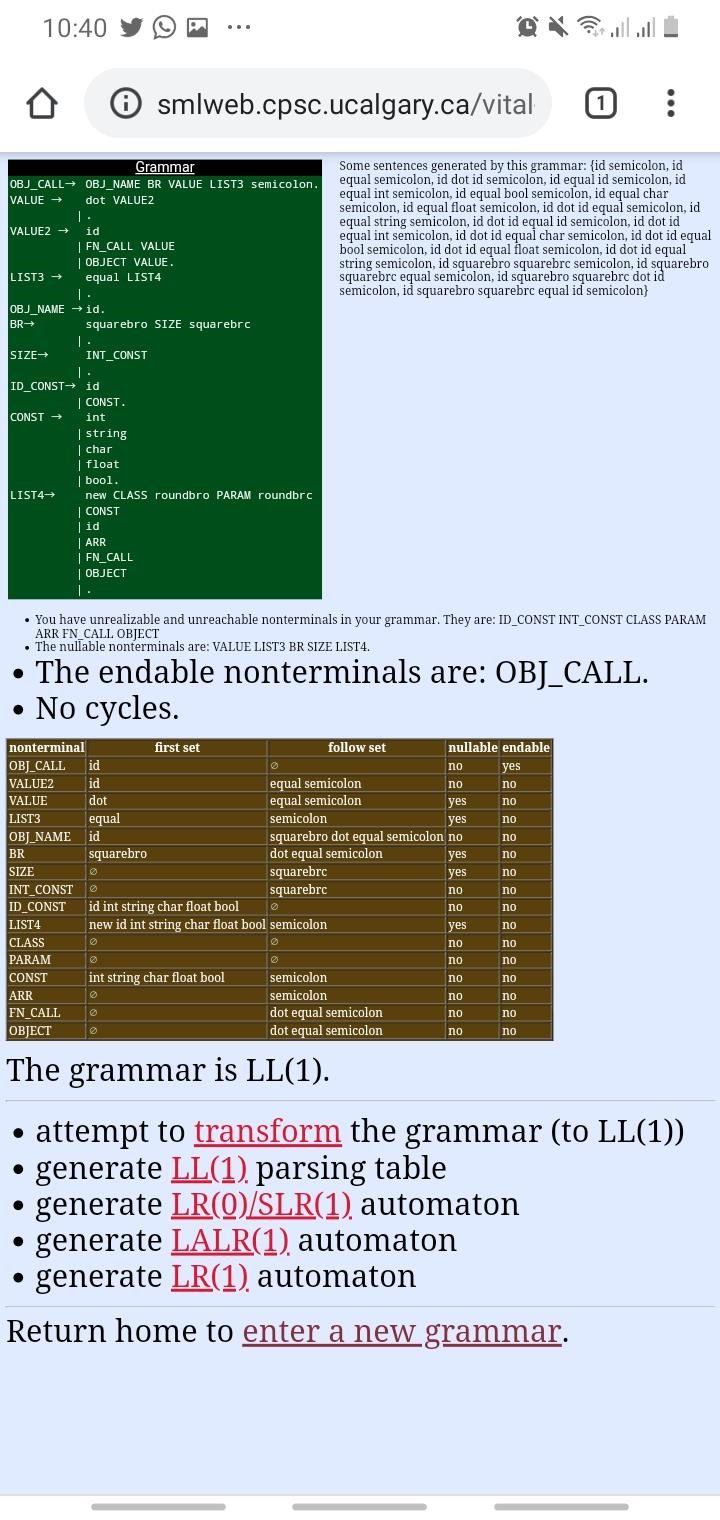
| id

| ARR

| FN\_CALL

| OBJECT

| .



# CLASS CFG

CLASS-> M class id B.

M-> abstract

| sealed | partial | .

B-> : id { BODY }.

BODY-> ;

| SST | { MST }.

SST-> WHILE\_ST

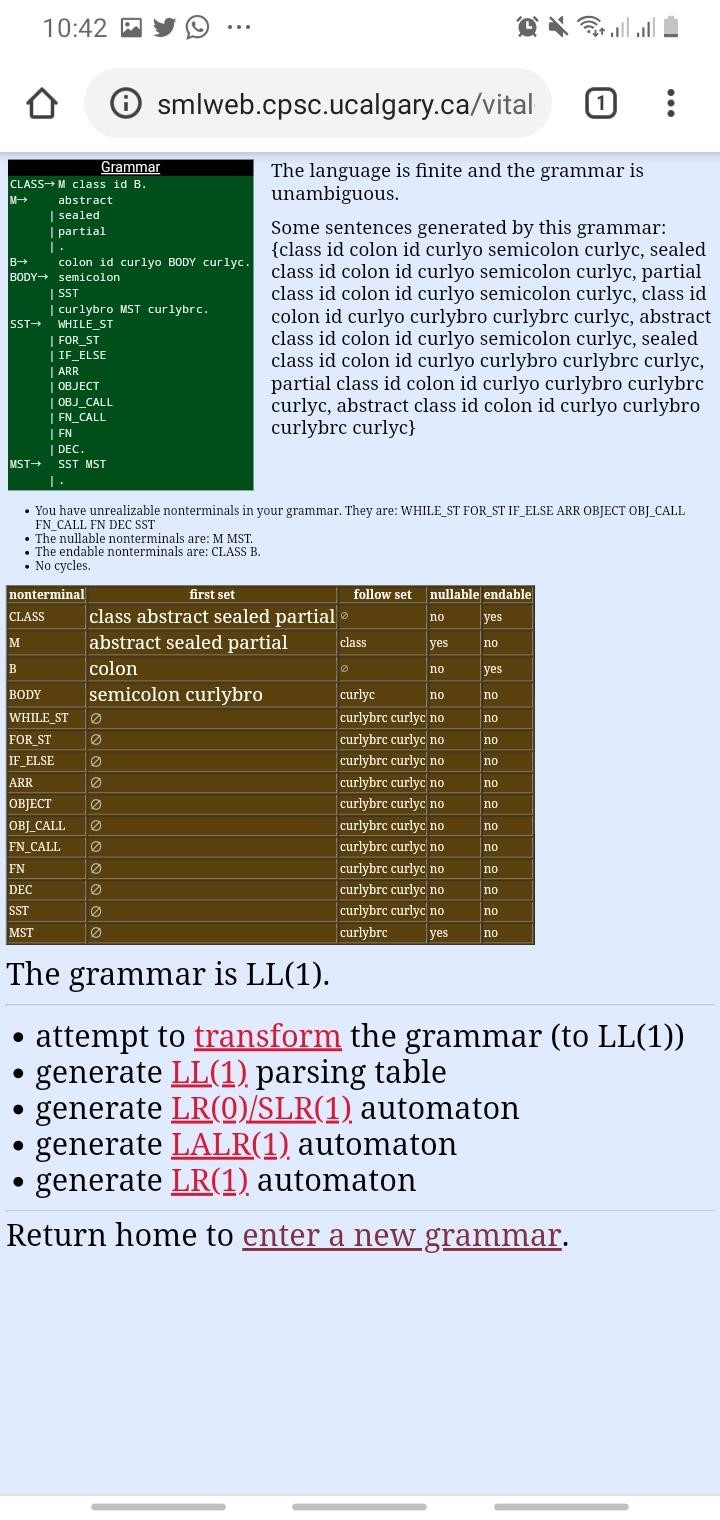
| FOR\_ST | IF\_ELSE | ARR

| OBJECT

| OBJ\_CALL

| FN\_CALL

| FN | DEC. MST->SST MST | .



# FUNCTION

FUNCTION -> AM S M RT id ( PARAM ) { BODY }.

AM->public | protected | private.

S-> static | .

M-> abstract

| sealed | private

| . RT-> dt | void.

BODY-> MST return ( ID\_CONST ) ;. MST-> SST MST.

PARAM-> dt id A

| ARR | .

ID\_CONST -> id | CONST. A-> PARAM | ARR.

SST->WHILE\_ST

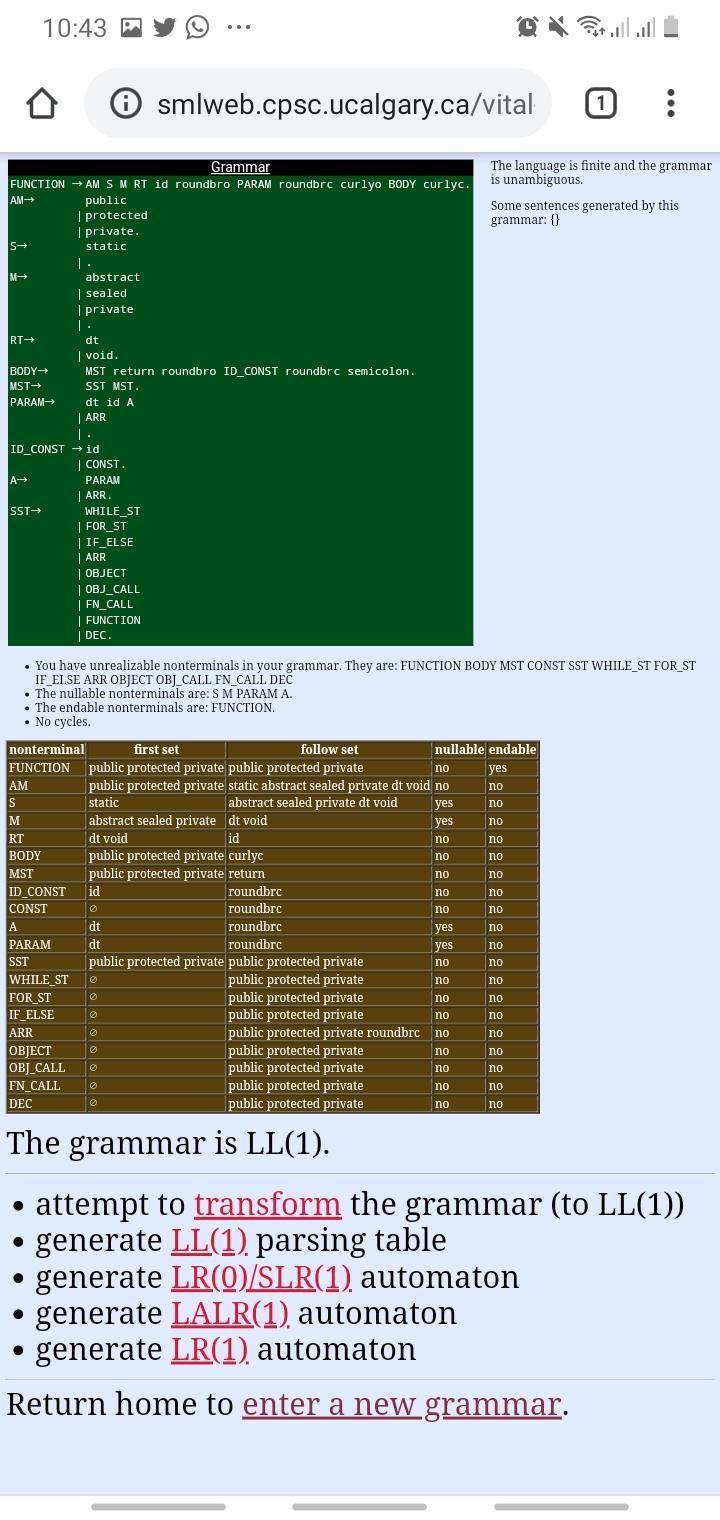
| FOR\_ST

| IF\_ELSE | ARR

| OBJECT

| OBJ\_CALL

| FN\_CALL | FUNCTION | DEC.



# FUNCTION CALL

FN\_CALL-> ASSIGN id ( PARAM ). ASSIGN-> id LF. AOP-> COMP\_AOP

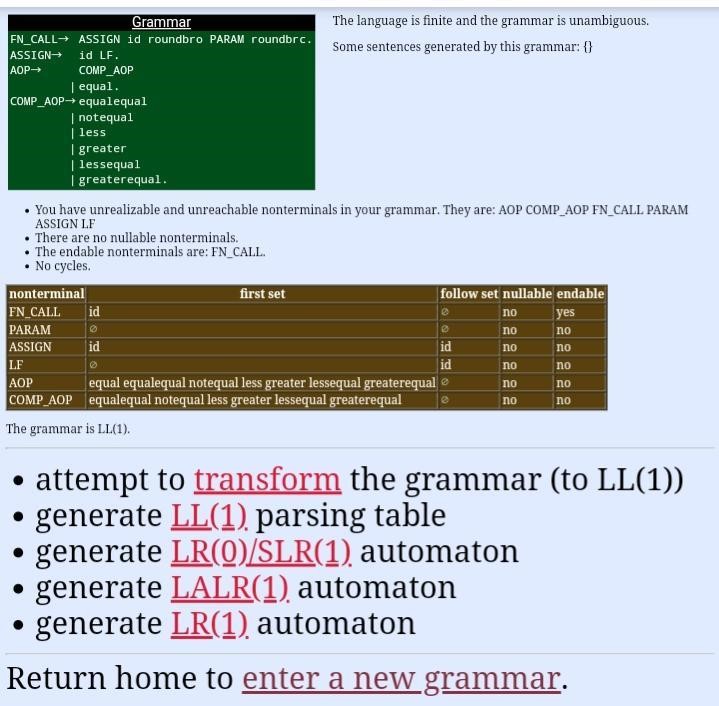
|=.

COMP\_AOP-> ==

|not=

|less

|greater |less= |greater=.



# ARRAY

ARRAY-> DT [ ] id T ;.

T-> B

| = new DT [ SIZE ] B.

B-> { LIST }

| .

SIZE-> INT\_CONST

| .

LIST->INT\_CONST coma LIST | INT\_CONST. ID\_CONST-> id | CONST.

CONST-> int

| string

| char

| float | bool.

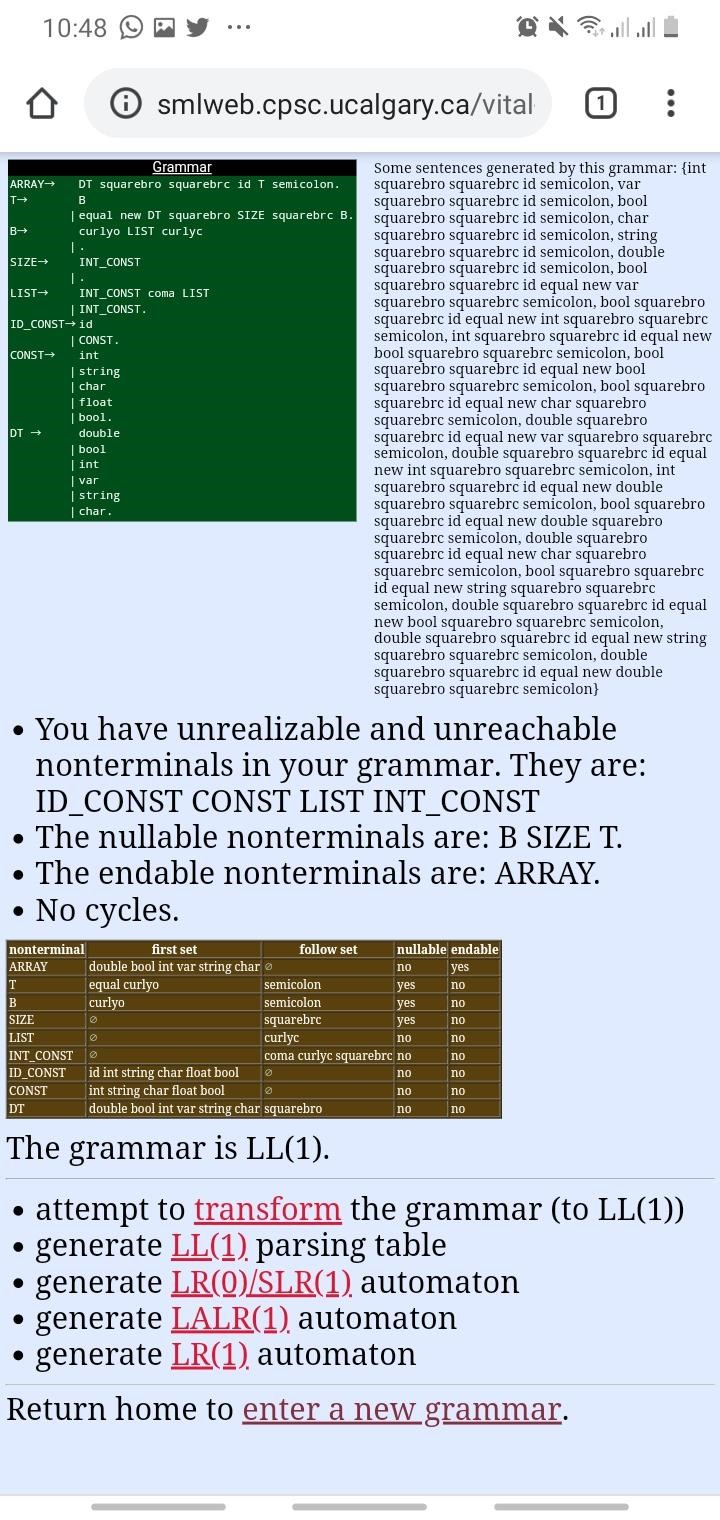
DT -> double

| bool

| int

| var

| string | char.



# FOR LOOP

FOR\_ST -> for ( F1 F2 ; F3 ) BODY.

F1-> ; |DEC |ASSIGN. F2->COND

|.

DEC -> DT id INIT LIST.

DT -> double

|bool

|int

|var

|string |char.

LIST -> ;

|, id INIT LIST.

INIT -> = IT.

IT ->id INIT |CONST.

COND->id LF1. LF1->ROP ID\_CONST

|. F3->INC\_DEC

|ASSIGN

|.

INC\_DEC\_ASSIGN-> id LF |OP id. LF-> OP |x AOP EXP.

OP-> plusplus

|minusminus.

ASSIGN-> id LF. AOP-> COMP\_AOP

|=.

COMP\_AOP-> ==

|not=

|less

|greater |less=

|greater=. EXP-> T E1. E1-> pm T E1

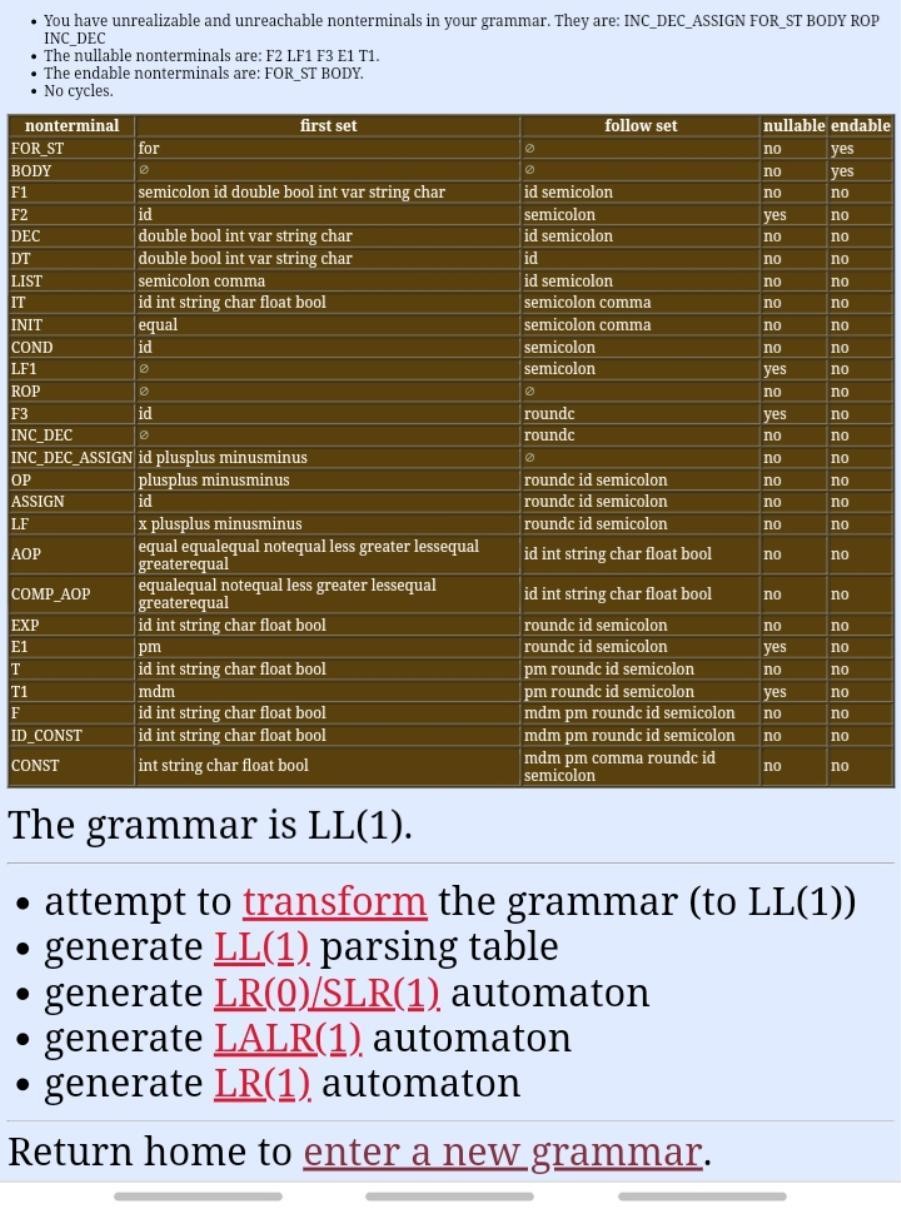
|.

T-> F T1. T1-> mdm F T1 |. F-> ID\_CONST. ID\_CONST->id |CONST.

CONST->int |string

|char |float |bool.





PART V: ATTRIBUTED GRAMMAR

DECLERATION ATTRIBUTED GRAMMAR:

DEC -> DT T=Token[index].VP id N=Token[index].VP INIT LIST.

DT -> double | bool | int | var | string | char.

LIST -> ; | , id N=Token[index].VP INIT LIST.

INIT -> = IT.

IT -> id N=Token[index].VP INIT|CONST.

CONST -> int | string | char | float | bool.

WHILE LOOP ATTRIBUTED GRAMMAR:

WHILE\_ST -> until T= ”until” BODY.

COND-> ID LF.

LF-> ROP ID\_CONST | . ID\_CONST-> ID | CONST.

BODY-> ; | SST | { MST }.

SST->WHILE\_ST | FOR\_ST | IF\_ELSE | ARR | OBJECT | OBJ\_CALL | FN\_CALL | FN DEC. MST-> SST MST | .

CONST -> int | string | char | float | bool.

IF ELSE ATTRIBUTED GRAMMAR:

IF\_ELSE -> when T= “when” BODY OELSE.

OELSE -> else BODY. COND-> ID LF.

LF -> ROP ID\_CONST| . ID\_CONST-> ID | CONST.

BODY-> ; | SST | { MST }.

SST->WHILE\_ST | FOR\_ST | IF\_ELSE | ARR | OBJECT | OBJ\_CALL | FN\_CALL | FN DEC.

MST-> SST MST | .

INTERFACE ATTRIBUTED GRAMMAR:

INT -> interface T=”interface” id N=Token[index].VP { BODY }.

{ createScope() destroyScope() }

BODY-> ; | SST | { MST }.

SST->WHILE\_ST | FOR\_ST | IF\_ELSE | ARR | OBJECT | OBJ\_CALL| FN\_CALL | FN DEC.

MST-> SST MST | .

OBJECT DECLERATION ATTRIBUTED GRAMMAR:

OBJECT-> CLASS BR OBJ\_NAME LIST2.

CLASS-> id. N=Token[index].VP

OBJ\_NAME -> id. N=Token[index].VP

BR-> [ SIZE ] | .

SIZE-> INT\_CONST | .

ID\_CONST-> id N=Token[index].VP | CONST.

CONST -> int | string | char | float | bool.

LIST2-> = new CLASS LIST | .

LIST-> ( PARAM ) | new CLASS BR.

OBJECT CALL ATTRIBUTED GRAMMAR:

OBJ\_CALL-> OBJ\_NAME BR VALUE LIST3 ;.

VALUE -> dot VALUE2 | .

VALUE2 -> id N=Token[index].VP | FN\_CALL VALUE | OBJECT VALUE.

LIST3 -> = LIST4| .

OBJ\_NAME -> id. N=Token[index].VP

BR-> [ SIZE ] | .

SIZE-> INT\_CONST | .

ID\_CONST-> id N=Token[index].VP | CONST.

CONST -> int | string | char | float | bool.

LIST4-> new CLASS ( PARAM ) | CONST | id N=Token[index].VP |ARR |FN\_CALL |OBJECT | .

CLASS DECLERATION ATTRIBUTED GRAMMAR:

CLASS-> M class T=”class” id N=Token[index].VP B.

{ createScope() destroyScope() }

M-> abstract | sealed | partial | .

B-> : id N=Token[index].VP { insert\_mt BODY }.

BODY-> ; | SST | { MST }.

SST-> WHILE\_ST | FOR\_ST | IF\_ELSE | ARR | OBJECT | OBJ\_CALL | FN\_CALL

| FN | DEC.

MST->SST MST | .

FUNCTION DECLERATION ATTRIBUTED GRAMMAR:

FUNCTION -> AM S M RT id N=Token[index].VP INSERT\_CT ( PARAM INSERT\_FT ) { BODY }.

{ createScope() destroyScope() }

AM->public | protected | private.

S-> static | .

M-> abstract | sealed | private | .

RT-> dt T=Token[index].VP | void. T= “void”

BODY-> MST return ( ID\_CONST ) ;.

MST-> SST MST.

PARAM-> dt T=Token[index].VP id N=Token[index].VP A | ARR | .

ID\_CONST -> id N=Token[index].VP | CONST.

A-> PARAM | ARR.

SST->WHILE\_ST | FOR\_ST | IF\_ELSE | ARR | OBJECT | OBJ\_CALL | FN\_CALL |FUNCTION | DEC.

FUNCTION CALL ATTRIBUTED GRAMMAR:

FN\_CALL-> ASSIGN id N=Token[index].VP (INSERT\_FT PARAM ).

ASSIGN-> id N=Token[index].VP LF.

AOP-> COMP\_AOP |=.

COMP\_AOP-> == |not= |less |greater |less= |greater=.

Parameter List:

<PL> -> <OE><PL2> | ε

<PL2> -> , <OE><PL2> | ε

ARRAY DECLERATION ATTRIBUTED GRAMMAR:

ARRAY-> DT T=Token[index].VP [ ] id N=Token[index].VP T ;.

T-> B | = new DT T=Token[index].VP [ SIZE ] B.

B-> { LIST } | .

SIZE-> INT\_CONST N=Token[index].VP | .

LIST->INT\_CONST N=Token[index].VP coma LIST | INT\_CONST. N=Token[index].VP

ID\_CONST-> id N=Token[index].VP | CONST. N=Token[index].VP

CONST-> int | string | char |float|bool.

DT -> double | bool | int | var | string | char.

FOR LOOP ATTRIBUTED GRAMMAR:

FOR\_ST -> for ( F1 F2 ; F3 ) BODY.

F1-> ; |DEC |ASSIGN.

F2->COND |.

DEC -> DT t=Token[index].VP id N=Token[index].VP INIT LIST.

DT -> double |bool |int |var |string |char.

LIST -> ; |, id N=Token[index].VP INIT LIST.

INIT -> = IT.

IT ->id N=Token[index].VP INIT |CONST.

COND->id N=Token[index].VP LF1.

LF1->ROP ID\_CONST N=Token[index].VP |.

F3->INC\_DEC |ASSIGN |.

INC\_DEC\_ASSIGN-> id N=Token[index].VP LF |OP id. N=Token[index].VP

LF-> OP |x AOP EXP.

OP-> plusplus |minusminus.

ASSIGN-> id N=Token[index].VP LF.

AOP-> COMP\_AOP |=.

COMP\_AOP-> == |not= |less |greater |less= |greater=.

EXP-> T E1.

E1-> pm T E1 |.

T-> F T1.

T1-> mdm F T1 |.

F-> ID\_CONST. N=Token[index].VP

ID\_CONST->id N=Token[index].VP |CONST. N=Token[index].VP

CONST->int |string |char |float |bool.

EXPRESSION:

<OE> -> <AE> <OE’>

<OE’> -> OR T=”OR” <AE> <OE’> | ε

<AE> -> <RE> <AE’>

<AE’> -> AND T=”AND” <RE><AE’> | ε

<RE> -> <E> <RE’>

<RE’> -> ROP OP=Token[index].VP <E> <RE’> | ε

<E> -> <T> <E’>

<E’> -> PM OP=Token[index].VP <T> <E’> | ε

<T> -> <F> <T’>

<T’> -> MDM OP=Token[index].VP <F> <T’> | ε

<F> -> ID N=Token[index].VP <F’> | <const> N=Token[index].VP | (<OE>) | ! <F> | <inc\_dec>

ID N=Token[index].VP <X>

<F’> ->. ID N=Token[index].VP <F’> | (<PL>) <F’\_list> |<inc\_dec> | ε

<F’\_list> ->. ID N=Token[index].VP <F’> | ε

<const> -> int\_const N=Token[index].VP | deci\_const N=Token[index].VP | char\_const N=Token[index].VP | str\_const N=Token[index].VP | bool\_const N=Token[index].VP |none N=Token[index].VP