University of Central Punjab

UCP

Lahore, Pakistan 2002

Name:	**Mahnoor Feroz**
Registration No:	**L1F17BSCS0106**
Section:	**“G”**
Submitted to:	**Sir Usman Tariq**

Complete Grammar :

```
Function         ⇒    Type identifier ( ArgList ) CompoundStmt        ----------- (Rule 1)
ArgList          ⇒    Arg | ArgList , Arg                             ----------- (Rule 2)
Arg              ⇒    Type identifier                                 ----------- (Rule 3)
Declaration      ⇒    Type IdentList ;                                ----------- (Rule 4)
Type             ⇒    int | float                                     ----------- (Rule 5)
IdentList        ⇒    identifier ,IdentList | identifier              ----------- (Rule 6)
Stmt             ⇒    ForStmt | WhileStmt | Expr ;
                      | IfStmt | CompoundStmt |Declaration | ;        ----------- (Rule 7)
ForStmt          ⇒    for < Expr ; OptExpr ; OptExpr > Stmt           ----------- (Rule 8)
OptExpr          ⇒    Expr | ε                                        ----------- (Rule 9)
WhileStmt        ⇒    while < Expr > Stmt                             ----------- (Rule 10)
IfStmt           ⇒    if < Expr > Stmt ElsePart                       ----------- (Rule 11)
ElsePart         ⇒    else Stmt | ε                                   ----------- (Rule 12)
CompoundStmt     ⇒     [ StmtList ]                                   ----------- (Rule 13)
StmtList         ⇒    StmtList Stmt | ε                               ----------- (Rule 14)
Expr             ⇒    identifier := Expr | Rvalue                     ----------- (Rule 15)
Rvalue           ⇒     Rvalue Compare Mag | Mag                       ----------- (Rule 16)
Compare          ⇒    == | < | > | <= | >= | != | <>                  ----------- (Rule 17)
Mag              ⇒     Mag + Term | Mag - Term | Term                 ----------- (Rule 18)
Term             ⇒    Term * Factor | Term / Factor | Factor          ----------- (Rule 19)
Factor           ⇒    ( Expr ) | identifier | number                  ----------- (Rule 20)
```

Converting Complete Grammar into LL(1) Grammar

Step 1: Removing ambiguity

There is no ambiguity in this grammar.

Step 2: Removing Left recursion

- *Rule 1* is a valid rule because there is no direct leftmost recursion.
- *Rule 2* has left direct recursion, therefore:

  α= ,Arg and **β**= Arg

  Thus,

  Arglist ⇒ Arg ArgList’

  ArgList’ ⇒ ,Arg ArgList’ | ***ε***
- *Rule 3* is a valid rule because there is no left direct or indirect recursion.
- *Rule 4* is a valid rule because there is no left direct or indirect recursion.
- *Rule 5* is a valid rule because there is no left direct or indirect recursion.
- *Rule 6* is a valid rule because there is no left direct or indirect recursion.
- *Rule 7* is a valid rule because there is no left direct or indirect recursion.
- *Rule 8* is a valid rule because there is no left direct or indirect recursion.
- *Rule 9* is a valid rule because there is no left direct or indirect recursion.
- *Rule 10* is a valid rule because there is no left direct or indirect recursion.
- *Rule 11* is a valid rule because there is no left direct or indirect recursion.
- *Rule 12* is a valid rule because there is no left direct or indirect recursion.
- *Rule 13* is a valid rule because there is no left direct or indirect recursion.
- *Rule 14* has direct left most derivation. Therefore,

  α= Stmt and **β**= ***ε***

  But **β** can not be null. Therefore:

  Removing null-production:

  StmtList ⇒ StmtList Stmt | Stmt

  And

  CompundStmt ⇒ [StmtList] | [ ]

  Thus, removing left direct recursion;

  α= Stmt and **β**= Stmt

  So,

  StmtList ⇒ Stmt StmtList’

  StmtList’ ⇒ Stmt StmtList’ | ***ε***

- *Rule 15* is a valid rule because there is no left direct or indirect recursion.
- *Rule 16* has direct leftmost recursion. Therefore:

  α= Compare Mag and **β**= Mag

  Thus,

  Rvalue ⇒ Mag Rvalue’

  Rvalue’ ⇒ Compare Mag Rvalue’ | ***ε***
- *Rule 17* is a valid rule because there is no left direct or indirect recursion.
- *Rule 18* has direct recursion. Therefore,

  α= + Term ; **α**= - Term and **β**= Term

  Thus,

Mag ⇒ Term Mag’

Mag’ ⇒ + Term Mag’ | -Term Mag’ | ***ε***

- *Rule 19* has direct recursion. Therefore,

  α= * Factor ; α= / Factor and β= Factor

  Thus,

  Term ⇒ Factor Term’

  Term’ ⇒ * Factor Term’ | / Factor Term’ | ***ε***

- *Rule 20* is a valid rule because there is no left direct or indirect recursion.

<u>Valid Grammer after removal of Left Recursion:</u>

Function	⇒	Type identifier (ArgList) CompoundStmt	----------- *(Rule 1)*
Arglist	⇒	Arg ArgList’	----------- *(Rule 2)*
ArgList’	⇒	‚Arg ArgList’ | ***ε***	----------- *(Rule 3)*
Arg	⇒	Type Identifier	----------- *(Rule 4)*
Declaration	⇒	Type IdentList ;	----------- *(Rule 5)*
Type	⇒	int | float	----------- *(Rule 6)*
IdentList	⇒	identifier ,IdentList | identifier	----------- *(Rule 7)*
Stmt	⇒	ForStmt | WhileStmt | Expr ;	
		| IfStmt | CompoundStmt |Declaration | ;	----------- *(Rule 8)*
ForStmt	⇒	for < Expr ; OptExpr ; OptExpr > Stmt	----------- *(Rule 9)*
OptExpr	⇒	Expr | ***ε***	----------- *(Rule 10)*
WhileStmt	⇒	while < Expr > Stmt	----------- *(Rule 11)*
IfStmt	⇒	if < Expr > StmtElsePart	----------- *(Rule 12)*
ElsePart	⇒	else Stmt | ***ε***	----------- *(Rule 13)*
CompoundStmt	⇒	[StmtList] | [ ]	----------- *(Rule 14)*
StmtList	⇒	Stmt StmtList’	----------- *(Rule 15)*
StmtList’	⇒	Stmt StmtList' | ***ε***	----------- *(Rule 16)*
Expr	⇒	identifier := Expr | Rvalue	----------- *(Rule 17)*
Rvalue	⇒	Mag Rvalue’	----------- *(Rule 18)*
Rvalue’	⇒	Compare Mag Rvalue’ | ***ε***	----------- *(Rule 19)*
Compare	⇒	== | < | > | <= | >= | != | <>	----------- *(Rule 20)*
Mag	⇒	Term Mag’	----------- *(Rule 21)*
Mag’	⇒	+ Term Mag’ | -Term Mag’ | ***ε***	----------- *(Rule 22)*
Term	⇒	Factor Term’	----------- *(Rule 23)*
Term’	⇒	* Factor Term’ | / Factor Term’ | ***ε***	----------- *(Rule 24)*
Factor	⇒	(Expr) | identifier | number	----------- *(Rule 25)*

Step 3: Removing Left factoring

- No left Factoring in Rule 1.
- No left Factoring in Rule 2.

- No left Factoring in Rule 3.
- No left Factoring in Rule 4.
- No left Factoring in Rule 5.
- No left Factoring in Rule 6.
- Left Factoring in Rule 7, thus:

  α=identifier and β_1= ,identList ; β_2= ε

  So:

  IdenList ⇒ identifier IdenList'

  IdentList' ⇒ ,identList | ε

- No left Factoring in Rule 8.
- No left Factoring in Rule 9.
- No left Factoring in Rule 10.
- No left Factoring in Rule 11.
- No left Factoring in Rule 12.
- No left Factoring in Rule 13.
- Left Factoring in Rule 14, thus

  α=[and β_1= StmtList] ; β_2=]

  So,

  CompoundStmt ⇒ [CompoundStmt'

  CompoundStmt' ⇒ StmtList] |]

- No left Factoring in Rule 15.
- No left Factoring in Rule 16.
- No left Factoring in Rule 17.
- No left Factoring in Rule 18.
- No left Factoring in Rule 19.
- No left Factoring in Rule 20.
- No left Factoring in Rule 21.
- No left Factoring in Rule 22.
- No left Factoring in Rule 23.
- No left Factoring in Rule 24.

LL(1) Grammar:

Function	⇒	Type identifier (ArgList) CompoundStmt	----------- *(Rule 1)*
Arglist	⇒	Arg ArgList'	----------- *(Rule 2)*
ArgList'	⇒	, Arg ArgList'	----------- *(Rule 3)*
ArgList'	⇒	ε	----------- *(Rule 4)*
Arg	⇒	Type Identifier	----------- *(Rule 5)*
Declaration	⇒	Type IdentList ;	----------- *(Rule 6)*
Type	⇒	int	----------- *(Rule 7)*
Type	⇒	float	----------- *(Rule 8)*
IdenList	⇒	identifier IdenList'	----------- *(Rule 9)*

IdenList’	⇒	,idenList	----------- *(Rule 10)*
IdenList’	⇒	***ε***	----------- *(Rule 11)*
Stmt	⇒	ForStmt	----------- *(Rule 12)*
Stmt	⇒	WhileStmt	----------- *(Rule 13)*
Stmt	⇒	Expr ;	----------- *(Rule 14)*
Stmt	⇒	IfStmt	----------- *(Rule 15)*
Stmt	⇒	CompoundStmt	----------- *(Rule 16)*
Stmt	⇒	Declaration	----------- *(Rule 17)*
Stmt	⇒	;	----------- *(Rule 18)*
ForStmt	⇒	for < Expr ; OptExpr ; OptExpr > Stmt	----------- *(Rule 19)*
OptExpr	⇒	Expr	----------- *(Rule 20)*
OptExpr	⇒	***ε***	----------- *(Rule 21)*
WhileStmt	⇒	while < Expr > Stmt	----------- *(Rule 22)*
IfStmt	⇒	if < Expr > Stmt ElsePart	----------- *(Rule 23)*
ElsePart	⇒	else Stmt	----------- *(Rule 24)*
ElsePart	⇒	***ε***	----------- *(Rule 25)*
CompoundStmt	⇒	[CompoundStmt’	----------- *(Rule 26)*
CompoundStmt’	⇒	StmtList]	----------- *(Rule 27)*
CompoundStmt’	⇒	]	----------- *(Rule 28)*
StmtList	⇒	Stmt StmtList’	----------- *(Rule 29)*
StmtList’	⇒	Stmt StmtList’	----------- *(Rule 30)*
StmtList'	⇒	***ε***	----------- *(Rule 31)*
Expr	⇒	identifier := Expr	----------- *(Rule 32)*
Expr	⇒	Rvalue	----------- *(Rule 33)*
Rvalue	⇒	Mag Rvalue’	----------- *(Rule 34)*
Rvalue’	⇒	Compare Mag Rvalue’	----------- *(Rule 35)*
Rvalue’	⇒	***ε***	----------- *(Rule 36)*
Compare	⇒	==	----------- *(Rule 37)*
Compare	⇒	<	----------- *(Rule 38)*
Compare	⇒	>	----------- *(Rule 39)*
Compare	⇒	<=	----------- *(Rule 40)*
Compare	⇒	>=	----------- *(Rule 41)*
Compare	⇒	!=	----------- *(Rule 42)*
Compare	⇒	<>	----------- *(Rule 43)*
Mag	⇒	Term Mag’	----------- *(Rule 44)*
Mag’	⇒	+ Term Mag’	----------- *(Rule 45)*
Mag’	⇒	-Term Mag’	----------- *(Rule 46)*
Mag’	⇒	***ε***	----------- *(Rule 47)*
Term	⇒	Factor Term’	----------- *(Rule 48)*
Term’	⇒	* Factor Term’	----------- *(Rule 49)*
Term’	⇒	/ Factor Term’	----------- *(Rule 50)*
Term’	⇒	***ε***	----------- *(Rule 51)*
Factor	⇒	(Expr)	----------- *(Rule 52)*
Factor	⇒	identifier	----------- *(Rule 53)*
Factor	⇒	number	----------- *(Rule 54)*

Rule	States	First Set	Follow Set
1	Function	int, float	$
2	Arglist	int, float	)
3	ArgList’	“ , ” , ε	)
4	Arg	int, float	“ , ” ,)
5	Declaration	int, float	; , for , while, identifier, (, number , if , [, int , float ,] , else
6	Type	int, float	identifier
7	IdenList	identifier	;
8	IdentList’	“ , ” , ε	;
9	Stmt	; , for , while, identifier, (, number , if , [, int , float	; , for , while, identifier, (, number , if , [, int , float ,] , else
10	ForStmt	for	; , for , while, identifier, (, number , if , [, int , float ,] , else
11	OptExpr	identifier, (, number , ε	; , >
12	WhileStmt	while	; , for , while, identifier, (, number , if , [, int , float ,] , else
13	IfStmt	if	; , for , while, identifier, (, number , if , [, int , float ,] , else
14	ElsePart	else, ε	; , for , while, identifier, (, number , if , [, int , float ,] , else
15	CompoundStmt	[	; , for , while, identifier, (, number , if , [, int , float ,] , else , $
16	CompoundStmt’	; , for , while, identifier, (, number , if , [, int , float ,]	; , for , while, identifier, (, number , if , [, int , float ,] , else , $
17	StmtList	; , for , while, identifier, (, number , if , [, int , float	]
18	StmtList’	; , for , while, identifier, (, number , if , [, int , float , ε	]
19	Expr	identifier, (, number	) , > , ;
20	Rvalue	identifier, (, number	) , > , ;
21	Rvalue’	=, < , > , ! , ε	) , > , ;

22	Compare	=, < , > , !	(, identifier , number
23	Mag	identifier, (, number	= , < , > , ! ,) , ;
24	Mag’	+ , - , **ε**	= , < , > , ! ,) , ;
25	Term	identifier, (, number	+ , - ,= , < , > , ! ,) , ;
26	Term’	* , / , **ε**	+ , - ,= , < , > , ! ,) , ;
27	Factor	identifier, (, number	+ , - ,= , < , > , ! , ; , * , /

First Set :

- First(Factor) = First(() ∪ First(identifier) ∪ First(number) = { (, identifier , number }.
- First(Term’) = First(*) ∪ First(/) = { * , / }
    - Term’ is nullable.
- First (Term) = First(Factor) = { (, identifier , number }.
- First (Mag‘) = First(+) ∪ First(-) = { + , - }
    - Mag’ is nullable.
- First (Mag) = First(Term) = { (, identifier , number }.
- First(Compare) = First(=) ∪ First(<) ∪ First(>) ∪ First(<) ∪ First(>) ∪ First(!) ∪ First(<)
  = { =, < , > , ! }
- First(Rvalue’) = First(Compare) = { =, < , > , ! }
    - Rvalue’ is nullable.
- First(Rvalue) = First(Mag) = { (, identifier, number }.
- First(Expr) = First(identifier) ∪ First(Rvalue) = { identifier, (, number }
- First(StmtList’)= First(Stmt) = { ; , for , while, identifier, (, number , if , [, int , float }
    - StmtList’ is nullable.
- First(StmtList) = First(Stmt) = { ; , for , while, identifier, (, number , if , [, int , float }
- First(CompundStmt’) = First(StmtList) ∪ First(])
  = { ; , for , while, identifier, (, number , if , [, int , float ,] }
- First(CompundStmt) =First([ ) = { [}
- First(ElsePart) = First(else) = { else }
    - ElsePart is nullable
- First(IfStmt) = First(if) ={ if }
- First(WhileStmt) = First(while) = { while }
- First(OptExpr) = First(Expr) = { identifier, (, number }
    - OptExpr is nullable.
- First(ForStmt) = First(for) = { for }
- First(Stmt) = First(ForStmt) ∪ First(WhileStmt) ∪ First(Expr)

∪ First(IfStmt) ∪ First(CompundStmt)
∪ First(Declaration) ∪ First(;)
= { ; , for , while, identifier, (, number , if , [, int , float }

- First(IdentList') = First(,) = { , }
    - IdentList is nullable
- First(IdentList) = First(Identifier) = { Identifier }
- First(Type) = First(int) ∪ First(float) = { int , float }
- First(Declaration) = First(Type) = { int , float }
- First(Arg) = First(Type) = { int , float }
- First(ArgList') = First(,) = { , }
    - ArgList' is nullable
- First(ArgList) = First(Arg) = { int , float }
- First(Function) = First(Type) = { int , float }

Follow Set:

- Follow(Function) = { $ }
- Follow(Arglist) = First()) = {) }
- Follow(ArgList') = {) }
    - Follow(ArgList') = Follow(ArgList')
    - Follow(ArgList') = Follow(ArgList) = {) }
- Follow(Arg) = { , ,) }
    - Follow(Arg) = First(ArgList') ∪ Follow(ArgList') = { , ,) }
    - Follow(Arg) = First(ArgList') ∪ Follow(ArgList) = { , ,) }
- Follow(Declaration) = Follow (Stmt)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }
- Follow(Type) = { Identifier }
    - Follow(Type) = First(IdentList) = { Identifier }
    - Follow(Type) = First(identifier) = { Identifier }
- Follow(IdenList) = { ; , Follow(IdenList') } = { ; }
    - Follow(IdenList) = Follow(IdenList')
    - Follow(IdenList) = First(;) = { ; }
- Follow(IdenList') = Follow(IdenList) = { ; , Follow(IdenList') } = { ; }
- Follow(Stmt) = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }
    - Follow(Stmt) = First(StmtList') ∪ Follow(StmtList')

      = { ; , for , while, identifier, (, number , if , [, int , float ,

      Follow(StmtList') }

= { ; , for , while, identifier, (, number , if , [, int , float ,] }

- ○ Follow(Stmt) = First(StmtList') ∪ Follow(StmtList)

  = { ; , for , while, identifier, (, number , if , [, int , float , Follow(StmtList)}

  = { ; , for , while, identifier, (, number , if , [, int , float ,] }

- ○ Follow(Stmt) = Follow(ElsePart)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- ○ Follow(Stmt) = First(ElsePart) ∪ Follow(IfPart)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- ○ Follow(Stmt) = Follow(WhileStmt)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- ○ Follow(Stmt) = Follow(ForStmt)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- Follow(ForStmt) = Follow(Stmt)

  ={ ; , for , while, identifier, (, number , if , [, int , float ,] , else , Follow(ForStmt) }

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- Follow(OptExpr) = First(;) ∪ First(>) ={ ; , > }
- Follow(WhileStmt) = Follow(Stmt)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- Follow(IfStmt) = Follow(Stmt) =

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else , Follow(ElsePart) }

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- Follow(ElsePart) = Follow(IfStmt)

  ={ ; , for , while, identifier, (, number , if , [, int , float ,] , else , Follow(ElsePart) }

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

- Follow(CompoundStmt)

  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else , $ }

  - ○ Follow(CompoundStmt) = Follow(Stmt)
    = { ; , for , while, identifier, (, number , if , [, int , float ,] , else }

    - Follow(CompoundStmt) = Follow(Function) = { $ }
- Follow(CompoundStmt’) = Follow(CompoundStmt)
  = { ; , for , while, identifier, (, number , if , [, int , float ,] , else , $ }
- Follow(StmtList) = First( ]) = {] }
- Follow(StmtList’) = Follow(StmtList) = {] }
    - Follow(StmtList’) = Follow(StmtList’)
    - Follow(StmtList’) = Follow(StmtList) = {] }
- Follow(Expr) = {) , > , ; }
    - Follow(Expr) = First( )) = {) }
    - Follow(Expr) = Follow(Expr)
    - Follow(Expr) = First(>) = { > }
    - Follow(Expr) = First(;) = { ; }
- Follow(Rvalue) = Follow(Expr) = {) , > , ; }
- Follow(Rvalue’) = {) , > , ; }
    - Follow(Rvalue’) = Follow(Rvalue’)
    - Follow(Rvalue’) = Follow(Rvalue) = {) , > , ; }
- Follow(Compare) = First(Mag) = { (, identifier , number }.
- Follow(Mag) = { = , < , > , ! ,) , ; }
    - Follow(Mag) = First(Rvalue’) ∪ Follow(Rvalue’) = { = , < , > , ! ,) , ; }
    - Follow(Mag) = First(Rvalue’) ∪ Follow(Rvalue) = { = , < , > , ! ,) , ; }
- Follow(Mag’) = { = , < , > , ! ,) , ; }
    - Follow(Mag’) = Follow(Mag’)
    - Follow(Mag’) = Follow(Mag) = { = , < , > , ! ,) , ; }
- Follow(Term) = { + , - ,= , < , > , ! ,) , ; }
    - Follow(Term) = First(Mag’) ∪ Follow(Mag’) = { + , - ,= , < , > , ! ,) , ; }
    - Follow(Term) = First(Mag’) ∪ Follow(Mag) = { + , - ,= , < , > , ! ,) , ; }
- Follow(Term’)= { + , - ,= , < , > , ! ,) , ; }
    - Follow(Term’)= Follow(Term’)
    - Follow(Term’)= Follow(Term) = { + , - ,= , < , > , ! ,) , ; }
- Follow(Factor) = { + , - ,= , < , > , ! , ; , * , / }
    - Follow(Factor) = First(Term’) ∪ Follow(Term’)
      = { + , - ,= , < , > , ! , ; }
    - Follow(Factor) = First(Term’) ∪ Follow(Term)
      = { * , / , + , - ,= , < , > , ! ,) , ; }