JADAVPUR UNIVERSITY

Faculty of Engineering & Technology

...CSE/PC/B/S/322 Compiler Design Lab...**Engg. Laboratory**

Class....CSE, UG3**... Sec.....**A1**.....**
Date of Experiment.........................
Date of Submission..........................
Marks Obtained........................
Signature of Examiner..........................

NAME	**Roll**
......Jeetesh Abrol.............	.........002210501021........
......Saiful Hossain............	.........002210501009.......
......Akshat Jaiswal...........	.........002210501039........
...Abrar Ahmed Ansari....	..........002210501005......
....	....
....	....

Title.... Mini Project

Commence at.................. **Completed at............................**

Name of Teacher concerned: Prof. Nandini Mukherjee

Features of Input Code:

- **Data Types:** int, float
- **Variables are assigned with values at the time of declaration only.**
- **Loop Constructs: while (condition) {S}.** (Nested Loops are not supported)
- **Within the Loop:** (i) Arithmetic Expressions assigning values to variables (ii) Increment Decrement Statements
- **Variables may be declared inside the loop as well.**
- **Supported Relational Operator:** Less Than (<) and Greater Than (>)
- **Supported Arithemetic operators:** PLUS(+), MINUS(-), MULTIPLY(*), DIVIDE(/), INCREMENT(++), DECREMENT(--)
- **Only function is main(), there is no other function. The main() function does not have arguments and return statements.**

Required tasks:

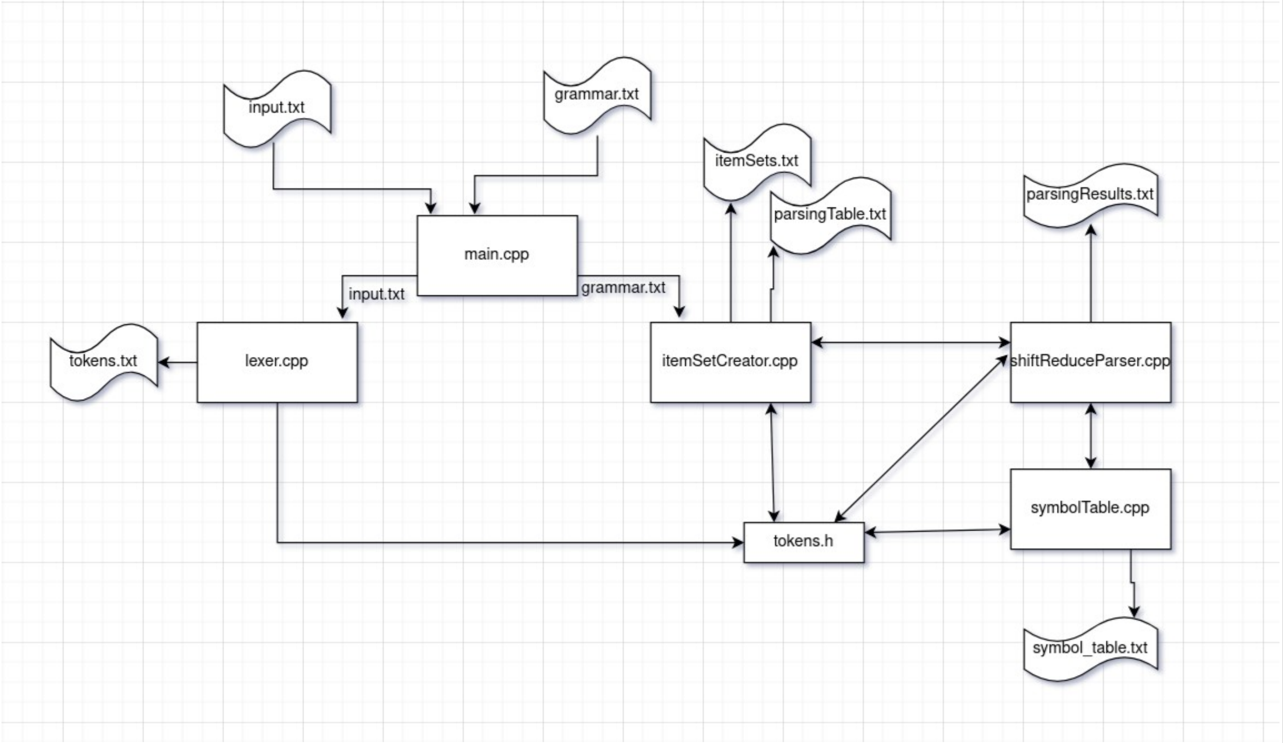
Part I – Construct a CFG for this language.

Part II – Write a lexical analyser to scan the stream of characters from a program written in the above language and generate stream of tokens.

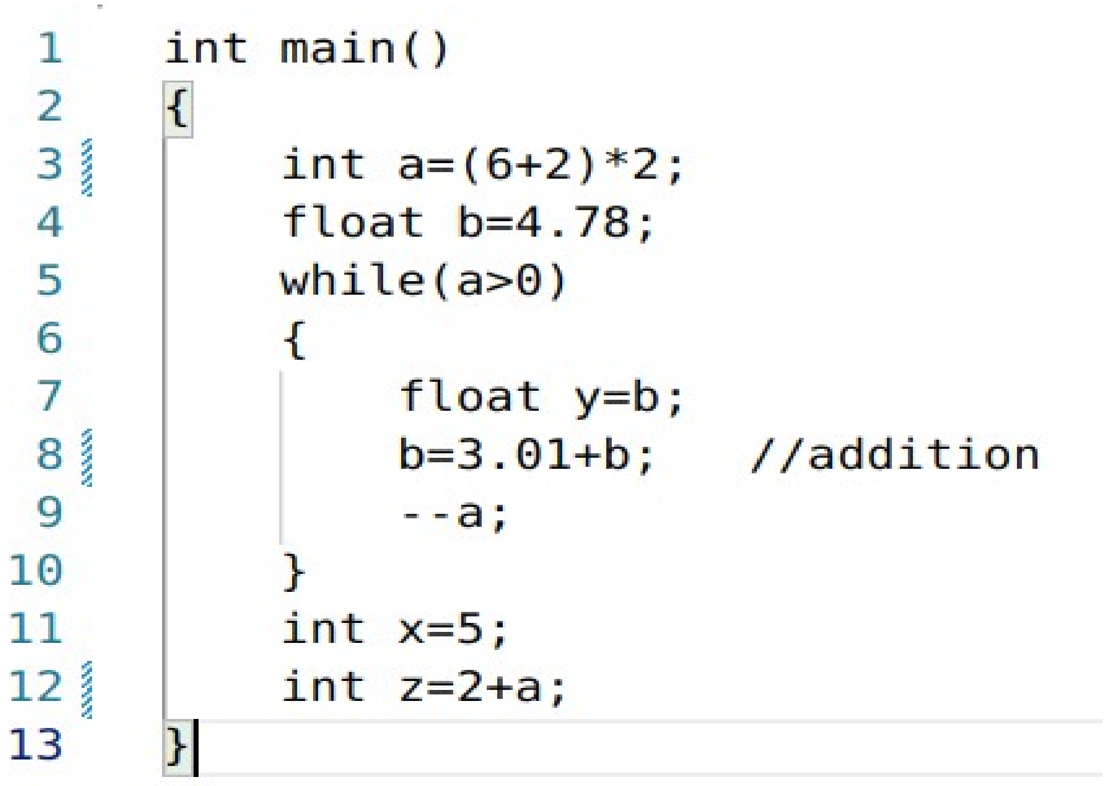
Part III – Maintain a symbol table with appropriate data structures.

Part IV – Write a bottom-up parser for this language (modules include Item-set construction, computation of FOLLOW, parsing table construction and parsing).

Flow Diagram:



Input Code:



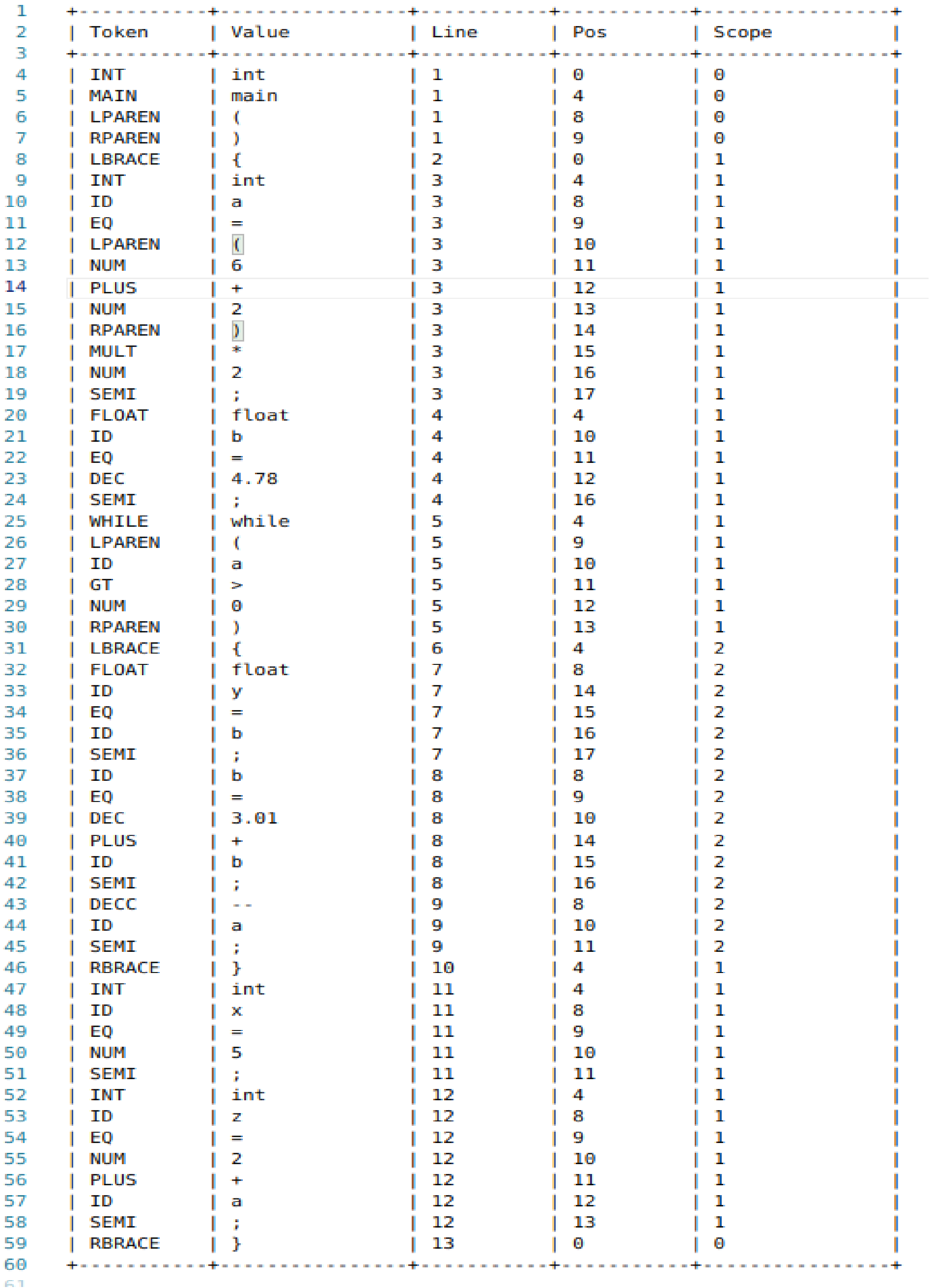
```
int main()
{
    int a=(6+2)*2;
    float b=4.78;
    while(a>0)
    {
        float y=b;
        b=3.01+b;   //addition
        --a;
    }
    int x=5;
    int z=2+a;
}
```

Context Free Grammar:

```
%tokens INT FLOAT MAIN WHILE EQ LT GT INC DECC PLUS MINUS MULT DIV MOD LPAREN RPAREN
LBRACE RBRACE SEMI COMMA ID DEC NUM

%%

S: INT MAIN LPAREN RPAREN compound_stmt
   ;

compound_stmt: LBRACE statement_list while_stmt statement_list RBRACE
   |   LBRACE statement_list while_stmt  RBRACE
   |   LBRACE statement_list RBRACE
   ;

declaration: type_specifier ID EQ literal SEMI
   | type_specifier ID EQ expression SEMI
   ;

type_specifier: INT
   | FLOAT
   ;

statement_list: statement
   |   statement_list statement
   ;

statement: assignment
   | declaration
   | inc_dec_statement
   ;

inc_dec_statement: ID INC SEMI
   | ID DECC SEMI
   | DECC ID SEMI
   | INC ID SEMI
   ;

assignment: ID EQ expression SEMI
   ;

expression: expression PLUS expression
   | expression MINUS expression
   | expression MULT expression
   | expression DIV expression
   | expression MOD expression
   | LPAREN expression RPAREN
   | ID
   | literal
   ;

literal: NUM
   | DEC
   ;

while_stmt: WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE
   ;
```

```
condition: expression LT expression
    | expression GT expression
    ;

%%
```

Tokens Generated:

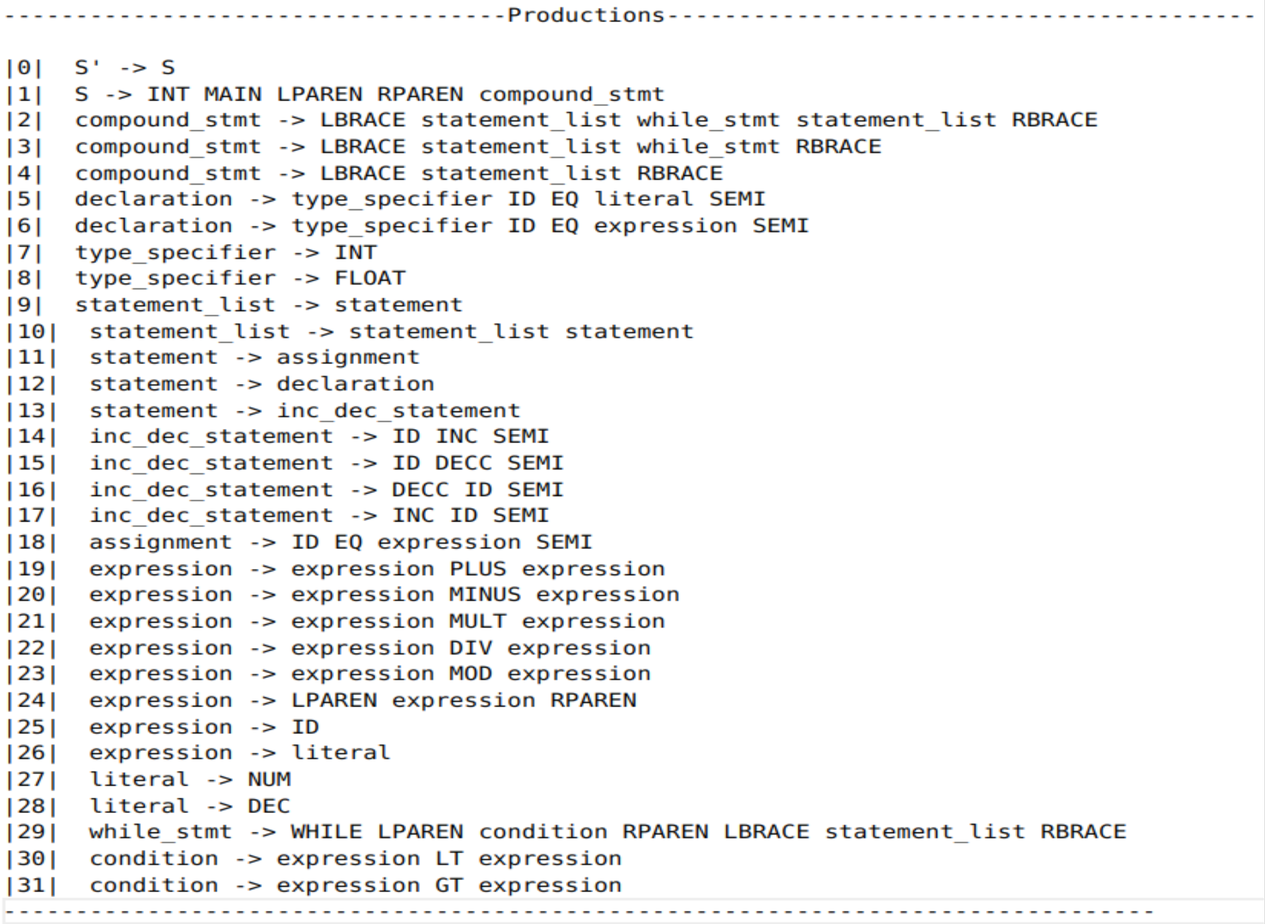


```
+------------+-----------------+------------+------------+-----------------+
| Token      | Value           | Line       | Pos        | Scope           |
+------------+-----------------+------------+------------+-----------------+
| INT        | int             | 1          | 0          | 0               |
| MAIN       | main            | 1          | 4          | 0               |
| LPAREN     | (               | 1          | 8          | 0               |
| RPAREN     | )               | 1          | 9          | 0               |
| LBRACE     | {               | 2          | 0          | 1               |
| INT        | int             | 3          | 4          | 1               |
| ID         | a               | 3          | 8          | 1               |
| EQ         | =               | 3          | 9          | 1               |
| LPAREN     | (               | 3          | 10         | 1               |
| NUM        | 6               | 3          | 11         | 1               |
| PLUS       | +               | 3          | 12         | 1               |
| NUM        | 2               | 3          | 13         | 1               |
| RPAREN     | )               | 3          | 14         | 1               |
| MULT       | *               | 3          | 15         | 1               |
| NUM        | 2               | 3          | 16         | 1               |
| SEMI       | ;               | 3          | 17         | 1               |
| FLOAT      | float           | 4          | 4          | 1               |
| ID         | b               | 4          | 10         | 1               |
| EQ         | =               | 4          | 11         | 1               |
| DEC        | 4.78            | 4          | 12         | 1               |
| SEMI       | ;               | 4          | 16         | 1               |
| WHILE      | while           | 5          | 4          | 1               |
| LPAREN     | (               | 5          | 9          | 1               |
| ID         | a               | 5          | 10         | 1               |
| GT         | >               | 5          | 11         | 1               |
| NUM        | 0               | 5          | 12         | 1               |
| RPAREN     | )               | 5          | 13         | 1               |
| LBRACE     | {               | 6          | 4          | 2               |
| FLOAT      | float           | 7          | 8          | 2               |
| ID         | y               | 7          | 14         | 2               |
| EQ         | =               | 7          | 15         | 2               |
| ID         | b               | 7          | 16         | 2               |
| SEMI       | ;               | 7          | 17         | 2               |
| ID         | b               | 8          | 8          | 2               |
| EQ         | =               | 8          | 9          | 2               |
| DEC        | 3.01            | 8          | 10         | 2               |
| PLUS       | +               | 8          | 14         | 2               |
| ID         | b               | 8          | 15         | 2               |
| SEMI       | ;               | 8          | 16         | 2               |
| DECC       | --              | 9          | 8          | 2               |
| ID         | a               | 9          | 10         | 2               |
| SEMI       | ;               | 9          | 11         | 2               |
| RBRACE     | }               | 10         | 4          | 1               |
| INT        | int             | 11         | 4          | 1               |
| ID         | x               | 11         | 8          | 1               |
| EQ         | =               | 11         | 9          | 1               |
| NUM        | 5               | 11         | 10         | 1               |
| SEMI       | ;               | 11         | 11         | 1               |
| INT        | int             | 12         | 4          | 1               |
| ID         | z               | 12         | 8          | 1               |
| EQ         | =               | 12         | 9          | 1               |
| NUM        | 2               | 12         | 10         | 1               |
| PLUS       | +               | 12         | 11         | 1               |
| ID         | a               | 12         | 12         | 1               |
| SEMI       | ;               | 12         | 13         | 1               |
| RBRACE     | }               | 13         | 0          | 0               |
+------------+-----------------+------------+------------+-----------------+
```

Data structure used to store tokens is:

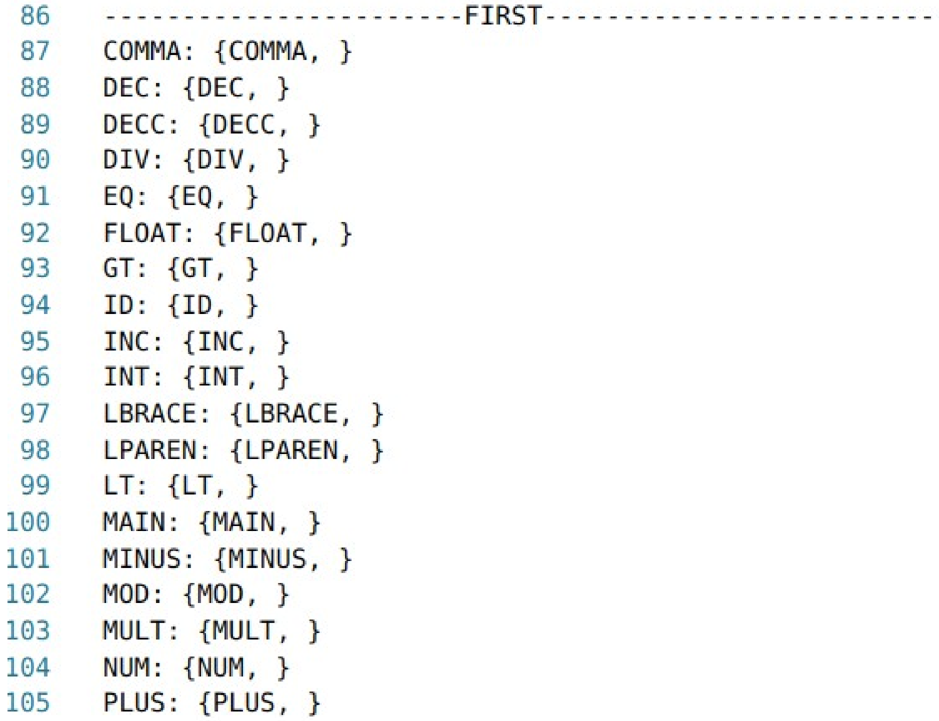
```
//<TokenType,value,line,pos,scope level>
vector<tuple<TokenType, string, int, int, int>> tokens;
```

Productions:

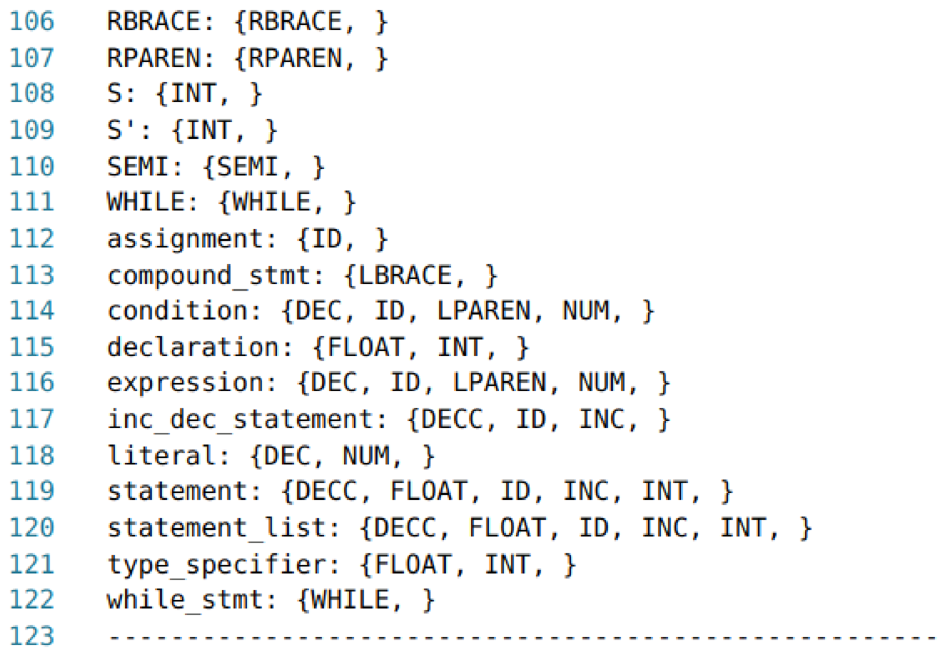


```
----------------------------------Productions----------------------------------------

|0|  S' -> S
|1|  S -> INT MAIN LPAREN RPAREN compound_stmt
|2|  compound_stmt -> LBRACE statement_list while_stmt statement_list RBRACE
|3|  compound_stmt -> LBRACE statement_list while_stmt RBRACE
|4|  compound_stmt -> LBRACE statement_list RBRACE
|5|  declaration -> type_specifier ID EQ literal SEMI
|6|  declaration -> type_specifier ID EQ expression SEMI
|7|  type_specifier -> INT
|8|  type_specifier -> FLOAT
|9|  statement_list -> statement
|10|  statement_list -> statement_list statement
|11|  statement -> assignment
|12|  statement -> declaration
|13|  statement -> inc_dec_statement
|14|  inc_dec_statement -> ID INC SEMI
|15|  inc_dec_statement -> ID DECC SEMI
|16|  inc_dec_statement -> DECC ID SEMI
|17|  inc_dec_statement -> INC ID SEMI
|18|  assignment -> ID EQ expression SEMI
|19|  expression -> expression PLUS expression
|20|  expression -> expression MINUS expression
|21|  expression -> expression MULT expression
|22|  expression -> expression DIV expression
|23|  expression -> expression MOD expression
|24|  expression -> LPAREN expression RPAREN
|25|  expression -> ID
|26|  expression -> literal
|27|  literal -> NUM
|28|  literal -> DEC
|29|  while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE
|30|  condition -> expression LT expression
|31|  condition -> expression GT expression
--------------------------------------------------------------------------------
```

First Sets:

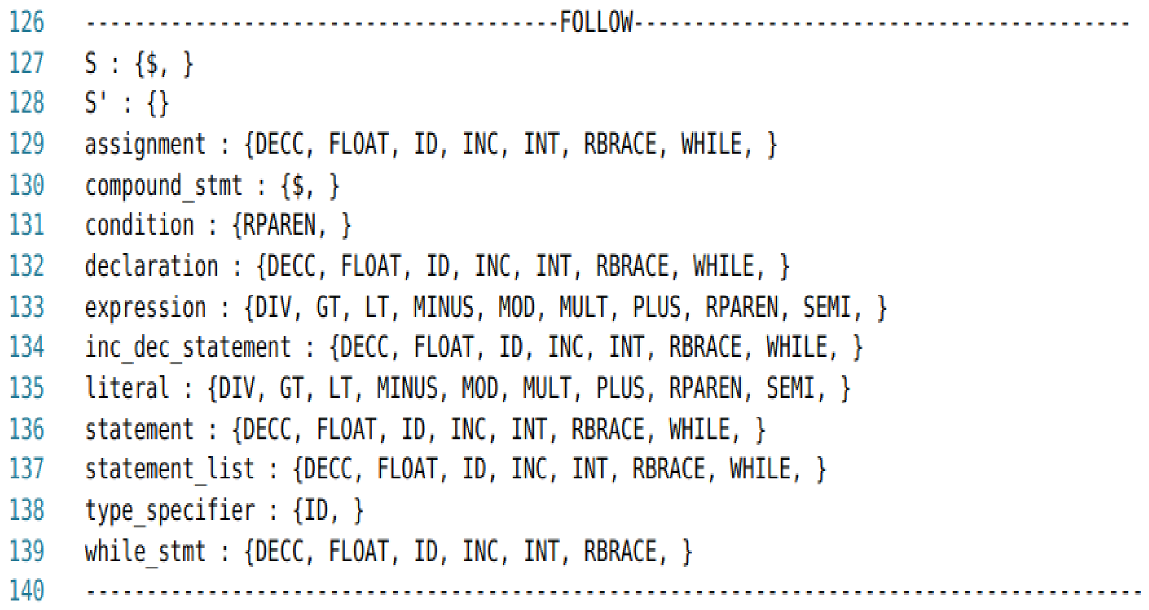


```
-----------------------FIRST------------------------
COMMA: {COMMA, }
DEC: {DEC, }
DECC: {DECC, }
DIV: {DIV, }
EQ: {EQ, }
FLOAT: {FLOAT, }
GT: {GT, }
ID: {ID, }
INC: {INC, }
INT: {INT, }
LBRACE: {LBRACE, }
LPAREN: {LPAREN, }
LT: {LT, }
MAIN: {MAIN, }
MINUS: {MINUS, }
MOD: {MOD, }
MULT: {MULT, }
NUM: {NUM, }
PLUS: {PLUS, }
```



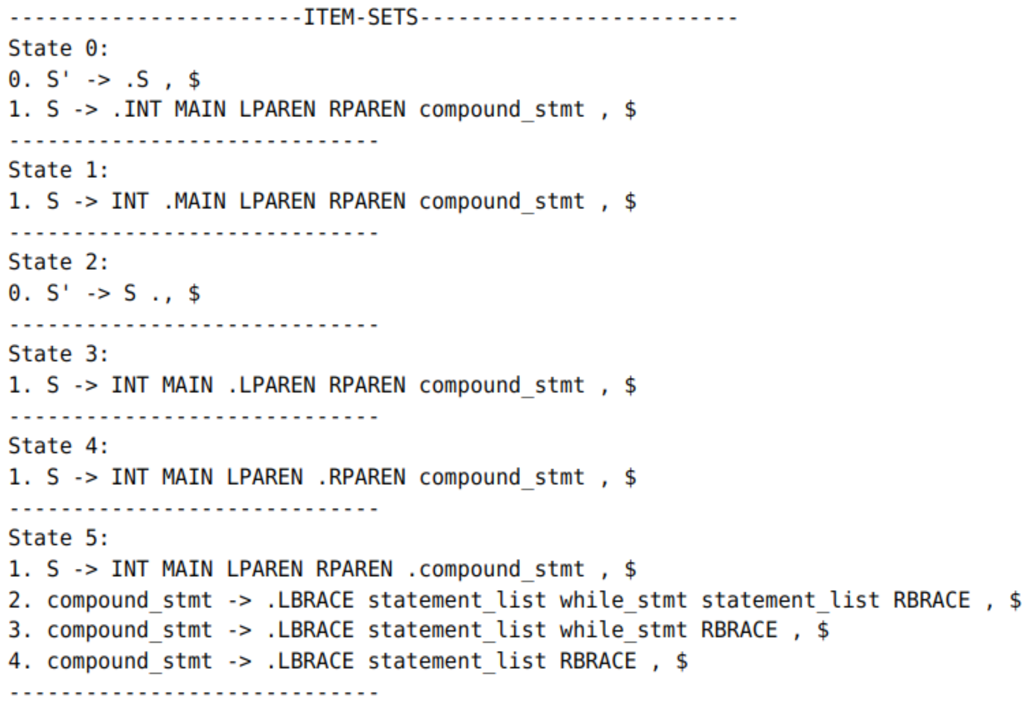
```
RBRACE: {RBRACE, }
RPAREN: {RPAREN, }
S: {INT, }
S': {INT, }
SEMI: {SEMI, }
WHILE: {WHILE, }
assignment: {ID, }
compound_stmt: {LBRACE, }
condition: {DEC, ID, LPAREN, NUM, }
declaration: {FLOAT, INT, }
expression: {DEC, ID, LPAREN, NUM, }
inc_dec_statement: {DECC, ID, INC, }
literal: {DEC, NUM, }
statement: {DECC, FLOAT, ID, INC, INT, }
statement_list: {DECC, FLOAT, ID, INC, INT, }
type_specifier: {FLOAT, INT, }
while_stmt: {WHILE, }
-----------------------------------------------------
```

Follow Sets:



```
------------------------------------FOLLOW-------------------------------------
S : {$, }
S' : {}
assignment : {DECC, FLOAT, ID, INC, INT, RBRACE, WHILE, }
compound_stmt : {$, }
condition : {RPAREN, }
declaration : {DECC, FLOAT, ID, INC, INT, RBRACE, WHILE, }
expression : {DIV, GT, LT, MINUS, MOD, MULT, PLUS, RPAREN, SEMI, }
inc_dec_statement : {DECC, FLOAT, ID, INC, INT, RBRACE, WHILE, }
literal : {DIV, GT, LT, MINUS, MOD, MULT, PLUS, RPAREN, SEMI, }
statement : {DECC, FLOAT, ID, INC, INT, RBRACE, WHILE, }
statement_list : {DECC, FLOAT, ID, INC, INT, RBRACE, WHILE, }
type_specifier : {ID, }
while_stmt : {DECC, FLOAT, ID, INC, INT, RBRACE, }
-------------------------------------------------------------------------------
```

Item Sets:



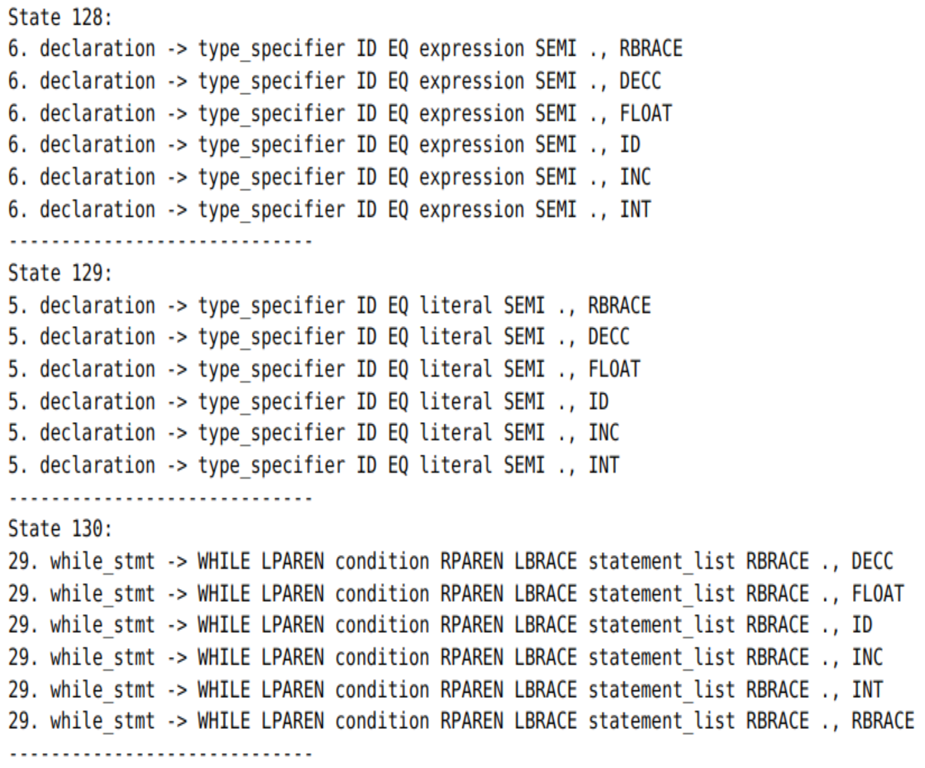
```
-----------------------ITEM-SETS------------------------
State 0:
0. S' -> .S , $
1. S -> .INT MAIN LPAREN RPAREN compound_stmt , $
-----------------------------
State 1:
1. S -> INT .MAIN LPAREN RPAREN compound_stmt , $
-----------------------------
State 2:
0. S' -> S ., $
-----------------------------
State 3:
1. S -> INT MAIN .LPAREN RPAREN compound_stmt , $
-----------------------------
State 4:
1. S -> INT MAIN LPAREN .RPAREN compound_stmt , $
-----------------------------
State 5:
1. S -> INT MAIN LPAREN RPAREN .compound_stmt , $
2. compound_stmt -> .LBRACE statement_list while_stmt statement_list RBRACE , $
3. compound_stmt -> .LBRACE statement_list while_stmt RBRACE , $
4. compound_stmt -> .LBRACE statement_list RBRACE , $
-----------------------------
```

• • • •

• • • •

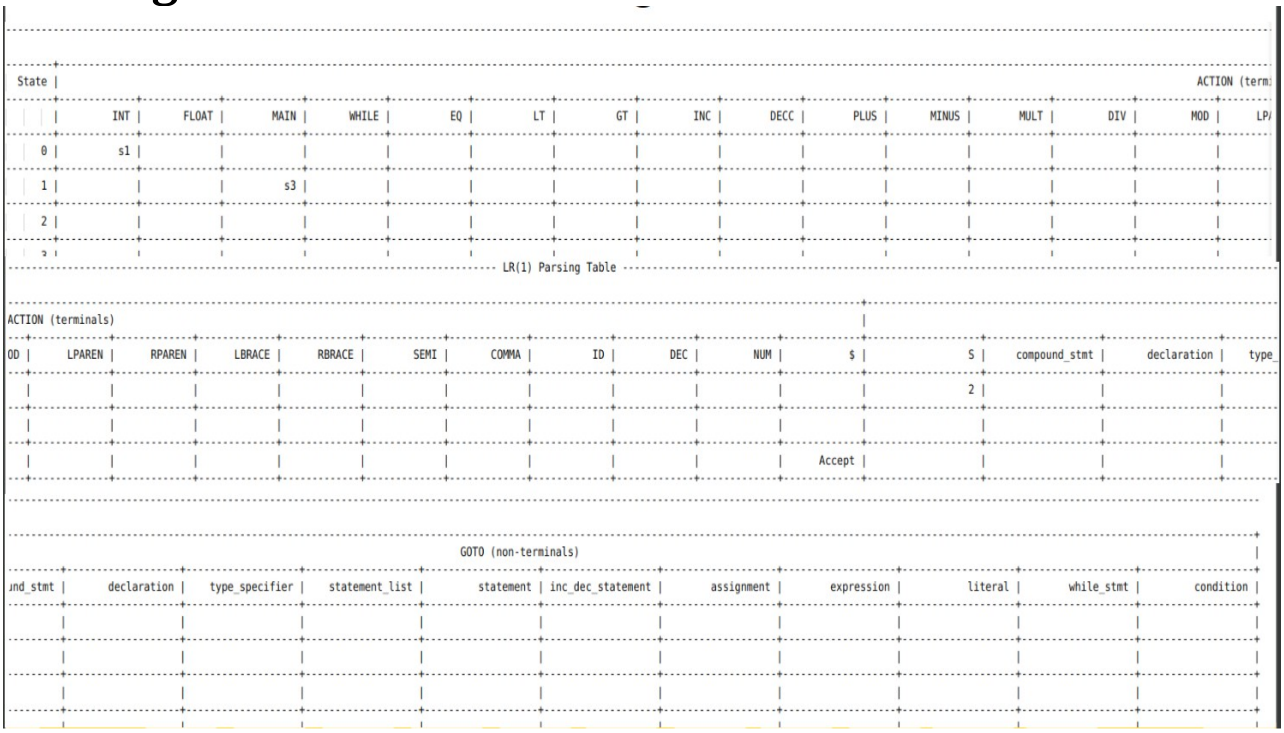
• • • •

• • • •



```
State 128:
6. declaration -> type_specifier ID EQ expression SEMI ., RBRACE
6. declaration -> type_specifier ID EQ expression SEMI ., DECC
6. declaration -> type_specifier ID EQ expression SEMI ., FLOAT
6. declaration -> type_specifier ID EQ expression SEMI ., ID
6. declaration -> type_specifier ID EQ expression SEMI ., INC
6. declaration -> type_specifier ID EQ expression SEMI ., INT
-----------------------------
State 129:
5. declaration -> type_specifier ID EQ literal SEMI ., RBRACE
5. declaration -> type_specifier ID EQ literal SEMI ., DECC
5. declaration -> type_specifier ID EQ literal SEMI ., FLOAT
5. declaration -> type_specifier ID EQ literal SEMI ., ID
5. declaration -> type_specifier ID EQ literal SEMI ., INC
5. declaration -> type_specifier ID EQ literal SEMI ., INT
-----------------------------
State 130:
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., DECC
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., FLOAT
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., ID
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., INC
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., INT
29. while_stmt -> WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE ., RBRACE
-----------------------------
```

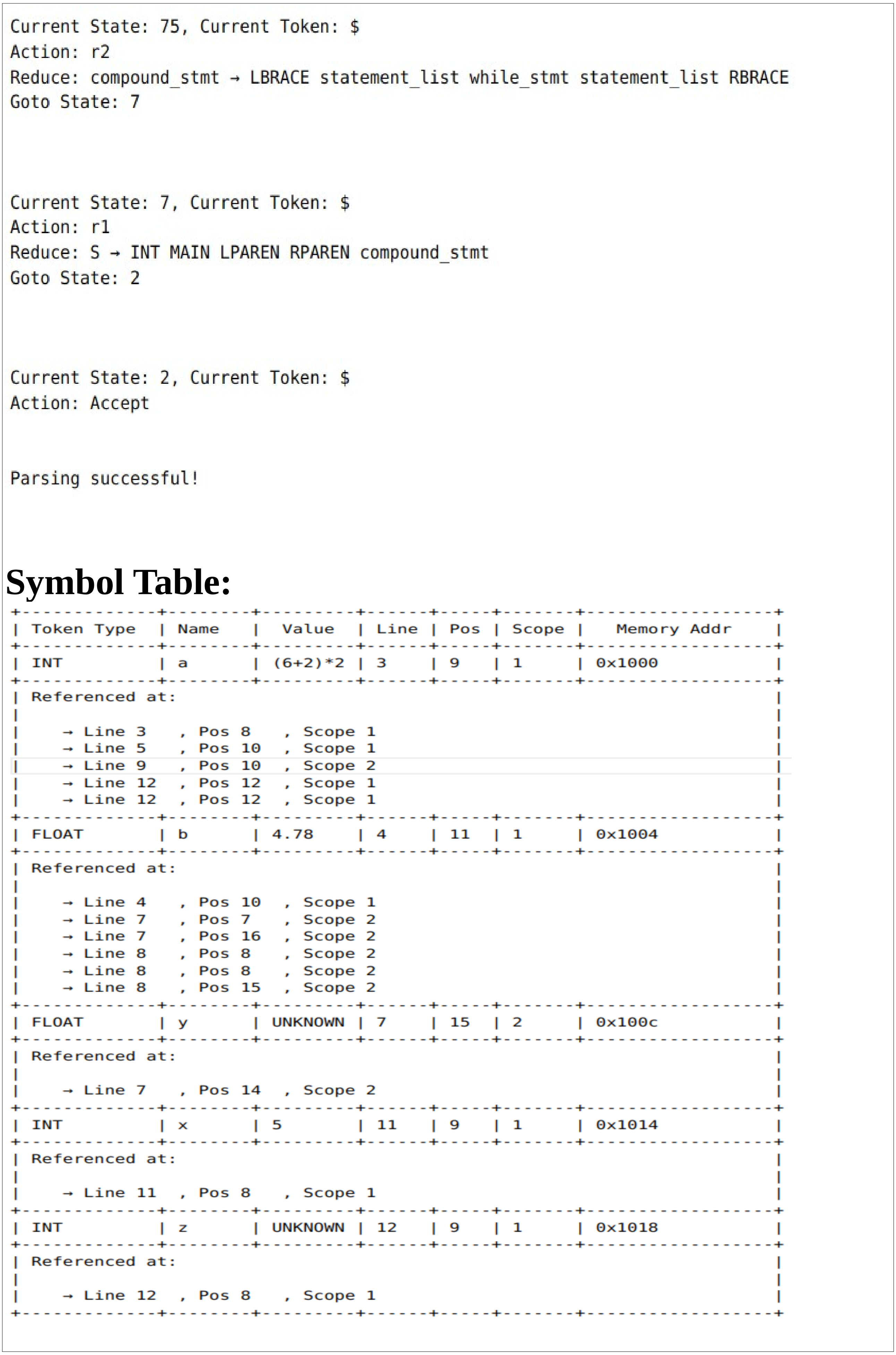
Parsing Table:



```
State |                                                                                                                  ACTION (term
      |    INT |  FLOAT |   MAIN |  WHILE |     EQ |     LT |     GT |    INC |   DECC |   PLUS |  MINUS |   MULT |    DIV |    MOD |   LPA
    0 |     s1 |        |        |        |        |        |        |        |        |        |        |        |        |        |
    1 |        |        |     s3 |        |        |        |        |        |        |        |        |        |        |        |
    2 |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
    3 |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
------------------------------------------------------ LR(1) Parsing Table ------------------------------------------------------

ACTION (terminals)                                                                             |
OD |  LPAREN |  RPAREN |  LBRACE |  RBRACE |    SEMI |   COMMA |      ID |     DEC |     NUM |       $ |          S |  compound_stmt |    declaration |  type_
   |         |         |         |         |         |         |         |         |         |         |          2 |                |                |
   |         |         |         |         |         |         |         |         |         |         |            |                |                |
   |         |         |         |         |         |         |         |         |         |         |  Accept |            |                |                |

                                           GOTO (non-terminals)                                                                   |
und_stmt |  declaration | type_specifier | statement_list |      statement | inc_dec_statement |  assignment |  expression |  literal |  while_stmt |  condition |
         |              |                |                |                |                   |             |             |          |             |            |
         |              |                |                |                |                   |             |             |          |             |            |
         |              |                |                |                |                   |             |             |          |             |            |
```

Parsing Results:

```
Current State: 0, Current Token: INT
Action: s1
Shift: INT


Current State: 1, Current Token: MAIN
Action: s3
Shift: MAIN


Current State: 3, Current Token: LPAREN
Action: s4
Shift: LPAREN


Current State: 4, Current Token: RPAREN
Action: s5
Shift: RPAREN


Current State: 5, Current Token: LBRACE
Action: s6
Shift: LBRACE


Current State: 6, Current Token: INT
Action: s12
Shift: INT
```



```
Current State: 75, Current Token: $
Action: r2
Reduce: compound_stmt → LBRACE statement_list while_stmt statement_list RBRACE
Goto State: 7



Current State: 7, Current Token: $
Action: r1
Reduce: S → INT MAIN LPAREN RPAREN compound_stmt
Goto State: 2



Current State: 2, Current Token: $
Action: Accept


Parsing successful!
```

Symbol Table:

```
+-------------+--------+---------+------+-----+-------+-----------------+
| Token Type  | Name   |  Value  | Line | Pos | Scope |   Memory Addr   |
+-------------+--------+---------+------+-----+-------+-----------------+
| INT         | a      | (6+2)*2 | 3    | 9   | 1     | 0x1000          |
+-------------+--------+---------+------+-----+-------+-----------------+
| Referenced at:                                                        |
|                                                                       |
|    → Line 3   , Pos 8   , Scope 1                                   |
|    → Line 5   , Pos 10  , Scope 1                                   |
|    → Line 9   , Pos 10  , Scope 2                                   |
|    → Line 12  , Pos 12  , Scope 1                                   |
|    → Line 12  , Pos 12  , Scope 1                                   |
+-------------+--------+---------+------+-----+-------+-----------------+
| FLOAT       | b      | 4.78    | 4    | 11  | 1     | 0x1004          |
+-------------+--------+---------+------+-----+-------+-----------------+
| Referenced at:                                                        |
|                                                                       |
|    → Line 4   , Pos 10  , Scope 1                                   |
|    → Line 7   , Pos 7   , Scope 2                                   |
|    → Line 7   , Pos 16  , Scope 2                                   |
|    → Line 8   , Pos 8   , Scope 2                                   |
|    → Line 8   , Pos 8   , Scope 2                                   |
|    → Line 8   , Pos 15  , Scope 2                                   |
+-------------+--------+---------+------+-----+-------+-----------------+
| FLOAT       | y      | UNKNOWN | 7    | 15  | 2     | 0x100c          |
+-------------+--------+---------+------+-----+-------+-----------------+
| Referenced at:                                                        |
|                                                                       |
|    → Line 7   , Pos 14  , Scope 2                                   |
+-------------+--------+---------+------+-----+-------+-----------------+
| INT         | x      | 5       | 11   | 9   | 1     | 0x1014          |
+-------------+--------+---------+------+-----+-------+-----------------+
| Referenced at:                                                        |
|                                                                       |
|    → Line 11  , Pos 8   , Scope 1                                   |
+-------------+--------+---------+------+-----+-------+-----------------+
| INT         | z      | UNKNOWN | 12   | 9   | 1     | 0x1018          |
+-------------+--------+---------+------+-----+-------+-----------------+
| Referenced at:                                                        |
|                                                                       |
|    → Line 12  , Pos 8   , Scope 1                                   |
+-------------+--------+---------+------+-----+-------+-----------------+
```

Symbol table is using the Data Structures:

```
struct Symbol
{
string tokenType;
string name;
string value;
bool isUsed;
int line;
int pos;
int scope;
uintptr_t memoryAddress;

//line,position,scope
vector<tuple<int, int, int>> references;
};
```

the above stores the details for a symbol

Symbol Table uses

```
//scope, name, Symbol
unordered_map<int, unordered_map<string, Symbol>> table;
```

This stores the symbol of one scope with a certain name in the symbol table.

Learnings from the Project:

1. **Context-Free Grammar (CFG) Design:** Gained experience in defining a formal grammar for a restricted programming language, including handling loops, expressions, and declarations.
2. **Lexical Analysis:** Learned how to tokenize a stream of characters into meaningful tokens (keywords, identifiers, operators, etc.), ensuring strict type recognition and handling of all valid constructs.
3. **Symbol Table Management:** Understood how to build and maintain a symbol table to store and track variable names, types, and scopes for semantic validation.
4. **Bottom-Up Parsing:** Implemented an LR(1) parser from scratch, including item-set construction, FOLLOW set computation, and building the parsing table.
5. **Parsing and Evaluation:** Developed logic to parse input programs and evaluate expressions inside the loop, respecting the grammar and type rules.
6. **Strict Type Checking:** Enforced strong typing and learned how to detect and handle type mismatches in expressions.
7. **Compiler Construction Skills:** Gained foundational knowledge of how a simple compiler front-end works, from scanning to parsing and semantic analysis.