Compiler project
Phase 2 parser generator

Team members:

Ahmed Talaat Noser (6)

Salma Ragab Gad (32)

Data structures used:

- **Unordered map** with string key (non-terminals) and its value is vector of vector of strings (productions) to store the context free grammar.
- Two **unordered sets** to store the first and follow sets.
- **Unordered map** with string key (non-terminals) and its value is another map with string key (terminals) and vector of strings value (one production) to store represents the parsing table.

Algorithms and techniques used:

- **First:** to get the set of the terminal symbols which occur as first symbols in strings derived from any string of the grammar symbols.
- **Follow:** to get the set of the terminals which occur immediately after a non-terminal in the strings derived from the starting symbol.
- **Construct parsing table:** constructed using first and follow sets to help in the left most derivation prediction.
- **LL(1) parser algorithm:** generates the left most derivation of the input iteratively using the previously generated parsing table and explicit stack.

Our functions explanation:

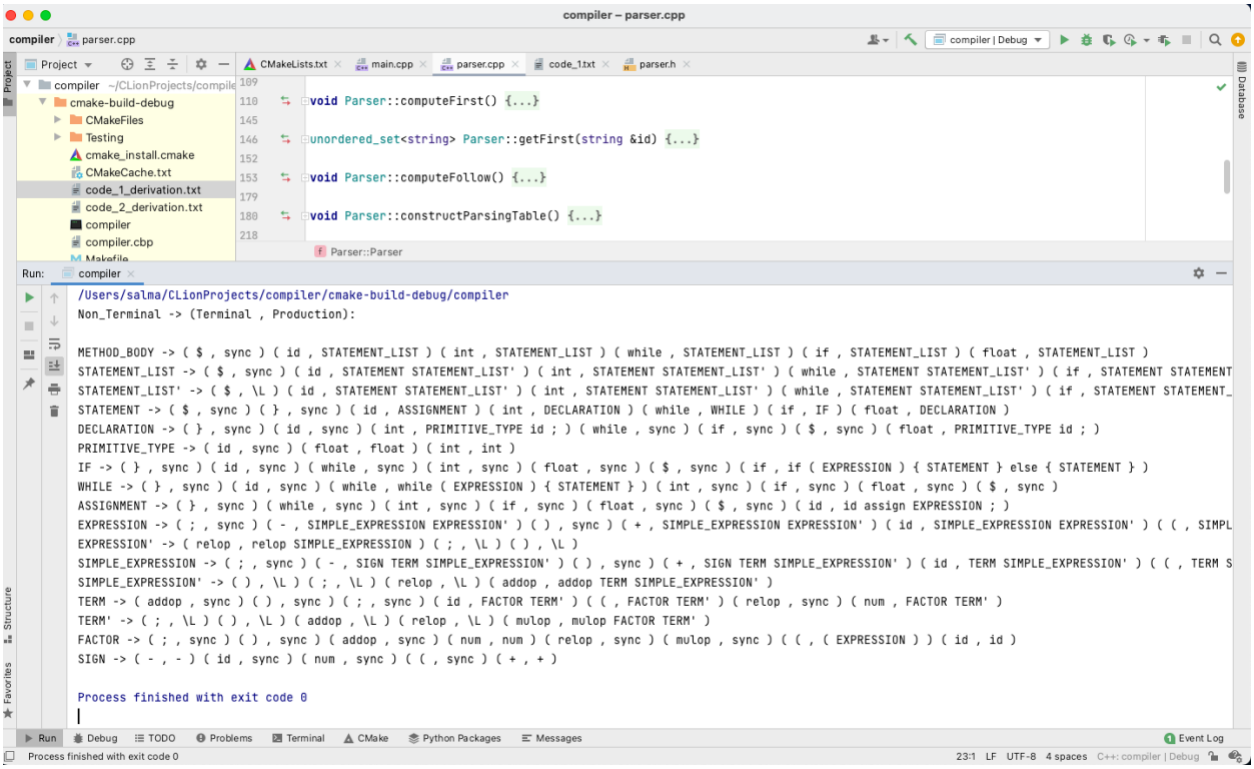
- **getCFG:** reads the CFG input file line by line.
- **getSent:** takes each production as a parameter and split it into its sentential.
- **parseCFG:** takes a vector of string (CFG) as a parameter and split it into non-terminals and their productions and store them in the unordered map explained above.
- **computeFirst:** uses **First** algorithm to compute the first set of each non-terminal and store them in the unordered map (first).
- **getFirst:** used to return the first set of any string of the grammar symbols (terminals or non-terminals).
- **computeFollow:** uses **Follow** algorithm to compute the follow set of each non-terminal and store them in the unordered map (follow).
- **constructParsingTable:** uses the **construct parsing table** algorithm to construct the parsing table, adding the synchronizing terminals to help in error recovery and also discover if the grammar is not LL(1) to produce the appropriate error message and terminate.
- **parse:** generates the left most derivation using the iterative top-down **LL(1) parser algorithm** and the previously generated parsing table.

Assumptions:

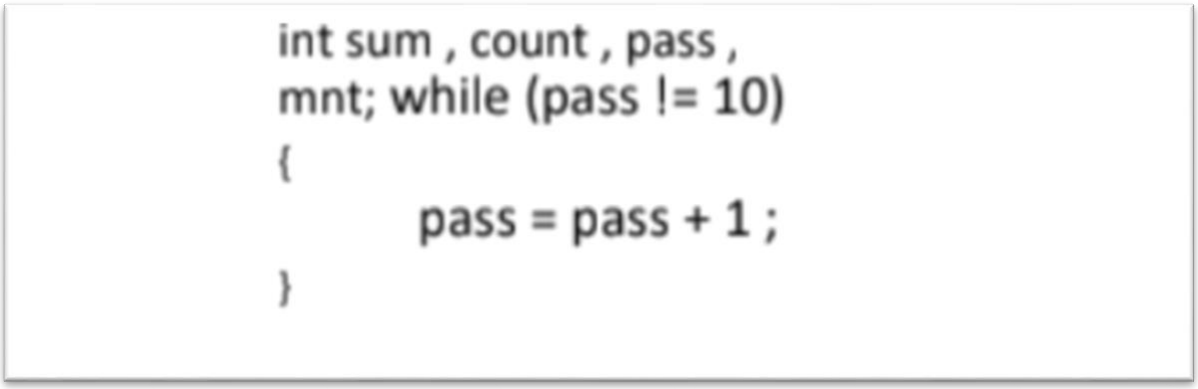
- CFG format like in the input file example.
- The ‘=’ sign is changed to ‘assign’ to match with the rules of phase 1.
- Left recursion is eliminated and left factoring is done manually before reading the input file.
  The edited CFG:

```
# METHOD_BODY = STATEMENT_LIST
# STATEMENT_LIST = STATEMENT STATEMENT_LIST'
# STATEMENT_LIST' = STATEMENT STATEMENT_LIST' | \L
# STATEMENT = DECLARATION
| IF
| WHILE
| ASSIGNMENT
# DECLARATION = PRIMITIVE_TYPE 'id' ';'
# PRIMITIVE_TYPE = 'int' | 'float'
# IF = 'if' '(' EXPRESSION ')' '{' STATEMENT '}' 'else' '{' STATEMENT '}'
# WHILE = 'while' '(' EXPRESSION ')' '{' STATEMENT '}'
# ASSIGNMENT = 'id' 'assign' EXPRESSION ';'
# EXPRESSION = SIMPLE_EXPRESSION EXPRESSION'
# EXPRESSION' = \L | 'relop' SIMPLE_EXPRESSION
# SIMPLE_EXPRESSION = TERM SIMPLE_EXPRESSION' | SIGN TERM SIMPLE_EXPRESSION'
# SIMPLE_EXPRESSION' = 'addop' TERM SIMPLE_EXPRESSION' | \L
# TERM = FACTOR TERM'
# TERM' = 'mulop' FACTOR TERM' | \L
# FACTOR = 'id' | 'num' | '(' EXPRESSION ')'
# SIGN = '+' | '-'
```

Parsing Table for the given CFG:



Output file for the code given in phase 1:



```
METHOD_BODY
STATEMENT_LIST
STATEMENT STATEMENT_LIST'
DECLARATION STATEMENT_LIST'
PRIMITIVE_TYPE id ; STATEMENT_LIST'
int id ; STATEMENT_LIST'
id ; STATEMENT_LIST'
; STATEMENT_LIST'
------------------------ERROR! missing ;. Inserted.------------------------
STATEMENT_LIST'
------------------------ERROR! Illegal STATEMENT_LIST'. Discard ,.------------------------
STATEMENT_LIST'
STATEMENT STATEMENT_LIST'
ASSIGNMENT STATEMENT_LIST'
id assign EXPRESSION ; STATEMENT_LIST'
assign EXPRESSION ; STATEMENT_LIST'
------------------------ERROR! missing assign. Inserted.------------------------
EXPRESSION ; STATEMENT_LIST'
------------------------ERROR! Illegal EXPRESSION. Discard ,.------------------------
EXPRESSION ; STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ; STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
id TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
------------------------ERROR! Illegal TERM'. Discard ,.------------------------
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
------------------------ERROR! Illegal TERM'. Discard id.------------------------
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
EXPRESSION' ; STATEMENT_LIST'
; STATEMENT_LIST'
STATEMENT_LIST'
STATEMENT STATEMENT_LIST'
WHILE STATEMENT_LIST'
while ( EXPRESSION ) { STATEMENT } STATEMENT_LIST'
( EXPRESSION ) { STATEMENT } STATEMENT_LIST'
EXPRESSION ) { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
id TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
relop SIMPLE_EXPRESSION ) { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION ) { STATEMENT } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
num TERM' SIMPLE_EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION' ) { STATEMENT } STATEMENT_LIST'
) { STATEMENT } STATEMENT_LIST'
{ STATEMENT } STATEMENT_LIST'
STATEMENT } STATEMENT_LIST'
ASSIGNMENT } STATEMENT_LIST'
id assign EXPRESSION ; } STATEMENT_LIST'
assign EXPRESSION ; } STATEMENT_LIST'
EXPRESSION ; } STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ; } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
id TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
```

```
id assign EXPRESSION ; } STATEMENT_LIST'
assign EXPRESSION ; } STATEMENT_LIST'
EXPRESSION ; } STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ; } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
id TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
addop TERM SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
num TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; } STATEMENT_LIST'
EXPRESSION' ; } STATEMENT_LIST'
; } STATEMENT_LIST'
} STATEMENT_LIST'
STATEMENT_LIST'

----------------------------------------------------Accepted----------------------------------------------------
```

Output file for the code given in phase 2:

```
int x;
x = 5;
if (x > 2)
{
        x = 0;
}
```

```
METHOD_BODY
STATEMENT_LIST
STATEMENT STATEMENT_LIST'
DECLARATION STATEMENT_LIST'
PRIMITIVE_TYPE id ; STATEMENT_LIST'
int id ; STATEMENT_LIST'
id ; STATEMENT_LIST'
; STATEMENT_LIST'
STATEMENT_LIST'
STATEMENT STATEMENT_LIST'
ASSIGNMENT STATEMENT_LIST'
id assign EXPRESSION ; STATEMENT_LIST'
assign EXPRESSION ; STATEMENT_LIST'
EXPRESSION ; STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ; STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
num TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; STATEMENT_LIST'
EXPRESSION' ; STATEMENT_LIST'
; STATEMENT_LIST'
STATEMENT_LIST'
STATEMENT STATEMENT_LIST'
IF STATEMENT_LIST'
if ( EXPRESSION ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
( EXPRESSION ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
EXPRESSION ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
id TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
relop SIMPLE_EXPRESSION ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
num TERM' SIMPLE_EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION' ) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
) { STATEMENT } else { STATEMENT } STATEMENT_LIST'
{ STATEMENT } else { STATEMENT } STATEMENT_LIST'
STATEMENT } else { STATEMENT } STATEMENT_LIST'
ASSIGNMENT } else { STATEMENT } STATEMENT_LIST'
id assign EXPRESSION ; } else { STATEMENT } STATEMENT_LIST'
assign EXPRESSION ; } else { STATEMENT } STATEMENT_LIST'
EXPRESSION ; } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
TERM SIMPLE_EXPRESSION' EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
FACTOR TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
num TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
TERM' SIMPLE_EXPRESSION' EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
SIMPLE_EXPRESSION' EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
EXPRESSION' ; } else { STATEMENT } STATEMENT_LIST'
; } else { STATEMENT } STATEMENT_LIST'
} else { STATEMENT } STATEMENT_LIST'
else { STATEMENT } STATEMENT_LIST'
-------------------------ERROR! missing else. Inserted.-------------------------
{ STATEMENT } STATEMENT_LIST'
-------------------------ERROR! missing {. Inserted.-------------------------
STATEMENT } STATEMENT_LIST'
} STATEMENT_LIST'
-------------------------ERROR! missing }. Inserted.-------------------------
STATEMENT_LIST'

--------------------------------------------------Accepted--------------------------------------------------
```