Compilers – II: Parser Generator

Team 4:

Aman Panwar - CS20BTECH11004

Pranav K Nayak - ES20BTECH11035

Shambhu Kavir - CS20BTECH11045

Shreya Kumar - ES20BTECH11026

Taha Adeel Mohammed - CS20BTECH11052

Vikhyath - CS20BTECH11056

Goals of Assignment

- Write a parser generator written in Bison for our language ‘Tangent’
- Parts of Assignment:
  - Create Grammar for the language
  - Write parser for the grammar
  - Integrate lexer from previous assignment to work with parser
  - Debug the grammar

About our Parser

- It is an **LALR(1)** parser
- There is **no IF-ELSE ambiguity** in our grammar.
- Our grammar is ambiguous, but bison deals with the shift-reduce conflicts by reducing as default. Despite this, we have kept the number of shift-reduce conflicts to a minimum.
- We took structural reference from the C grammar, but built upon this by adding **OOPS features** and **other features specific to our language - Tangent.**

Grammar:

We wrote the Context Free Grammar for our language.

Since it is too big to fit in the presentation, use the following command in the root directory to generate the grammar description:

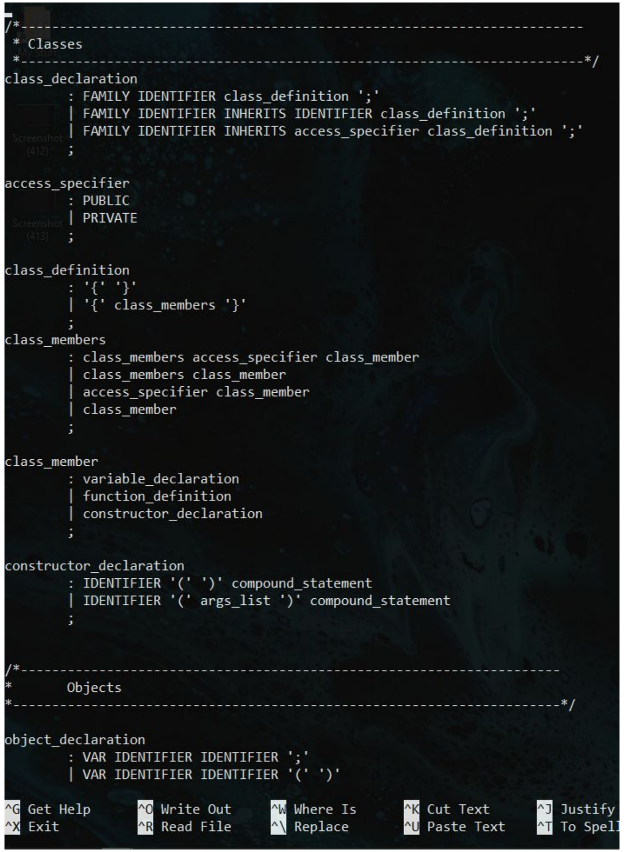
```
bison -t -v --debug -d src/parser.y; vim parser.output
```

Challenges Faced

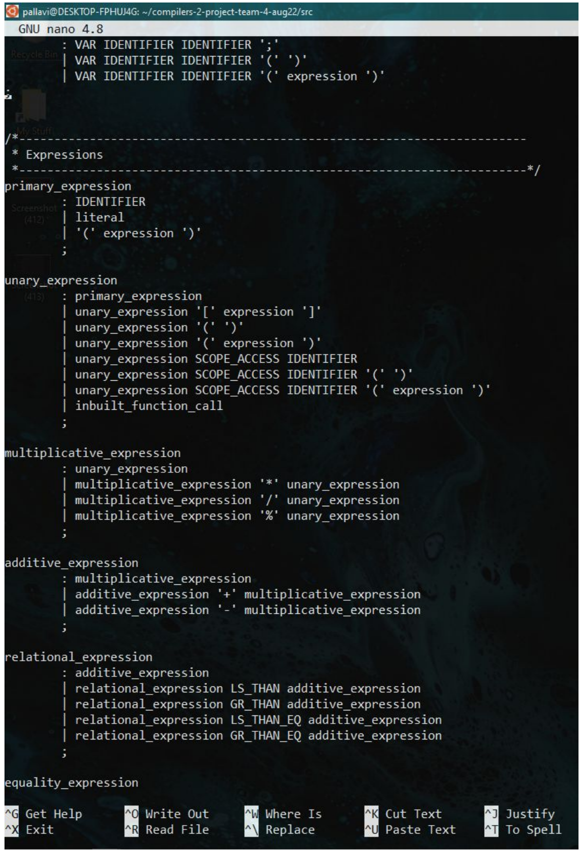
- While generating our parser, using Bison from our parser.y file, we had several **shift-reduce conflicts.** We were unable to fully eliminate all conflicts, but we managed to bring the total count of shift-reduce conflicts to **4.**
- Another challenge that we faced was while writing the grammar rules for the **OOPS principles** such as inheritance, object creation, and member access. Finally, we managed to implement these to some extent in our grammar.
- Since, it was our first time working with bison, we initially had issues with how to debug our code. We used the following flags to compiler our parser ***-t -v --debug -d*** , with this we were able to create output files with which we could check the correctness of our grammar.

Next Steps...

- We will generate a **Parse Tree** and a **Symbol Table** and move on to our **Semantic Analysis Phase.**



```
/*------------------------------------------------------------------------
 * Classes
 *------------------------------------------------------------------------*/
class_declaration
        : FAMILY IDENTIFIER class_definition ';'
        | FAMILY IDENTIFIER INHERITS IDENTIFIER class_definition ';'
        | FAMILY IDENTIFIER INHERITS access_specifier class_definition ';'
        ;

access_specifier
        : PUBLIC
        | PRIVATE
        ;

class_definition
        : '{' '}'
        | '{' class_members '}'
        ;
class_members
        : class_members access_specifier class_member
        | class_members class_member
        | access_specifier class_member
        | class_member
        ;

class_member
        : variable_declaration
        | function_definition
        | constructor_declaration
        ;

constructor_declaration
        : IDENTIFIER '(' ')' compound_statement
        | IDENTIFIER '(' args_list ')' compound_statement
        ;


/*---------------------------------------------------------------------
*       Objects
*---------------------------------------------------------------------*/

object_declaration
        : VAR IDENTIFIER IDENTIFIER ';'
        | VAR IDENTIFIER IDENTIFIER '(' ')'
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell
```



```
pallavi@DESKTOP-FPHUJ4G: ~/compilers-2-project-team-4-aug22/src
 GNU nano 4.8
        : VAR IDENTIFIER IDENTIFIER ';'
        | VAR IDENTIFIER IDENTIFIER '(' ')'
        | VAR IDENTIFIER IDENTIFIER '(' expression ')'
;


/*------------------------------------------------------------------------
 * Expressions
 *------------------------------------------------------------------------*/
primary_expression
        : IDENTIFIER
        | literal
        | '(' expression ')'
        ;

unary_expression
        : primary_expression
        | unary_expression '[' expression ']'
        | unary_expression '(' ')'
        | unary_expression '(' expression ')'
        | unary_expression SCOPE_ACCESS IDENTIFIER
        | unary_expression SCOPE_ACCESS IDENTIFIER '(' ')'
        | unary_expression SCOPE_ACCESS IDENTIFIER '(' expression ')'
        | inbuilt_function_call
        ;

multiplicative_expression
        : unary_expression
        | multiplicative_expression '*' unary_expression
        | multiplicative_expression '/' unary_expression
        | multiplicative_expression '%' unary_expression
        ;

additive_expression
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
        ;

relational_expression
        : additive_expression
        | relational_expression LS_THAN additive_expression
        | relational_expression GR_THAN additive_expression
        | relational_expression LS_THAN_EQ additive_expression
        | relational_expression GR_THAN_EQ additive_expression
        ;

equality_expression
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell
```

CODE SNIPPETS