Київський національний університет імені
Тараса Шевченка
Факультет комп'ютерних наук та кібернетики

Лабораторна робота **№5**
З дисципліни “Системне програмування”

Виконав студент 3-го курсу
групи МІ-31 Гришечкін Тихон

Київ – 2023

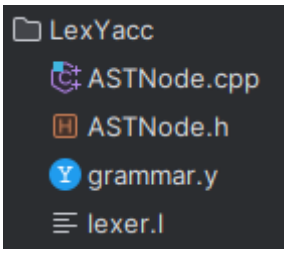
Умова лабораторної

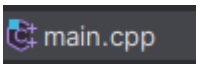
На базі Lex/YACC (або аналога) розробити синтаксичний аналізатор (з генерацією коду або обчисленнями - опціонально) для С або іншої сучасної імперативної (ООП, процедурної, тощо) мови програмування, або для арифметичних виразів (спрощений варіант).

Оцінювання:
2 бали - коректний розв'язок (працюючий проєкт)
2 бали - відповіді на питання по коду
+2 бали - для граматики мови С або С++
+2 бали - за відповіді на питання по теорії: LALR-аналізатор, принципи роботи yacc, як розв'язуються конфлікти в ході розбору...
+3 бали - за додавання семантики до AST (реалізацію обчислень)
+4 бали - за генерацію коду на цільовій мові в результаті трансляції
+3 бали - за візуалізацію AST (d3.js, або будь-яким зручним для користувача та наочним засобом)
за невчасну здачу - 50% від отриманої оцінки
при виявленні плагіату (суттєвих запозичень в коді) - оцінка 0 балів за всю роботу

Деталі реалізації

Мова виконання - **c++**.





Проект складається з lex, yacc файлів для побудови лексичного та синтаксичного аналізатора. Класс ASTNode що використовується під час аналізу для побудови AST дерева та результатів обчислень на основі аналізу. У функції main відбувається генерація зображення побудованого AST дерева за допомогою Graphviz.

```
[0-9]+"."[0-9]+    { yylval.node=new ASTNode("Number",atof(yytext)); return NUMBER; }
[0-9]+             { yylval.node=new ASTNode("Number",atoi(yytext)); return NUMBER; }
[-]         { yylval.node=new ASTNode("-"); return MINUS; }
[+]         { yylval.node=new ASTNode("+"); return PLUS; }
[*]         { yylval.node=new ASTNode("*"); return MULTIPLY; }
[/]         { yylval.node=new ASTNode("/"); return DIVIDE; }
[(]         { yylval.node=new ASTNode("(");}
[)]         { yylval.node=new ASTNode(")");}
[ \t]       ; // Пропуск пробілів та табуляцій
```

Розбиття на токени за допомогою lex.

```
exxpresionsList: exxpresionsList finalexpr {...}
    | finalexpr {...}
    ;

finalexpr: expr NEWLINE {$$=new ASTNode("FinalExpr", $1->getValue());$$->addChild($1); $$ -> print(0);}
    ;

expr: NUMBER        {...}
    | expr PLUS expr    {...}
    | expr MINUS expr   {...}
    | expr MULTIPLY expr    {...}
    | expr DIVIDE expr  {...}
    | '(' expr ')'      {...}
    ;
```

```
%union
{
    class ASTNode *node;
}
```

```
%start exxpresionsList
```

```
expr: NUMBER        { $$ = new ASTNode("Expression", $1->getValue()); $$ -> addChild($1);}
    | expr PLUS expr    { $$ = evaluate($1, '+', $3); }
    | expr MINUS expr   { $$ = evaluate($1, '-', $3); }
    | expr MULTIPLY expr    { $$ = evaluate($1, '*', $3); }
    | expr DIVIDE expr  { $$ = evaluate($1, '/', $3); }
```

Стартовий нетермінал exxpressionsList, програма очікує що на вход програма отримує арифметичні вирази розділені переходами на новий рядок.
Кожен нетермінал зберігає клас ASTNode, що має такі поля.

```
private:
    std::string label;
    double value;
    std::vector<ASTNode*> children;
```

```
BISON_TARGET(MyParser LexYacc/grammar.y ${CMAKE_CURRENT_BINARY_DIR}/grammar.tab.cpp)
FLEX_TARGET(MyScanner LexYacc/lexer.l ${CMAKE_CURRENT_BINARY_DIR}/lexer.yy.cpp)
```

```
add_executable(parser_c__ ${BISON_MyParser_OUTPUTS} ${FLEX_MyScanner_OUTPUTS} main.cpp LexYacc/ASTNode.cpp)
```

Виконуємо команди для генерації парсера.

```
extern int yyparse();
extern ASTNode* start_token_value;

void generateJson(const ASTNode* node, std::ostream& output) {...}

void generateDotHelper(const ASTNode* node, std::ostream& output) {...}

void generateDot(const ASTNode* node, std::ostream& output) {...}
```

під час синтаксичного аналізу зберігаємо ноду стартового нетермінала, для побудови AST дерева.

```
generateDot( node: root, &: dotFile);

dotFile.close();


std::system( Command: "\"C:\\Program Files\\Graphviz\\bin\\dot.exe\" -Tpng -o ast.png ast.dot");
```

Генеруємо dot файл та використовуємо його для побудови зображення нашого AST

Результати виконання

input.txt

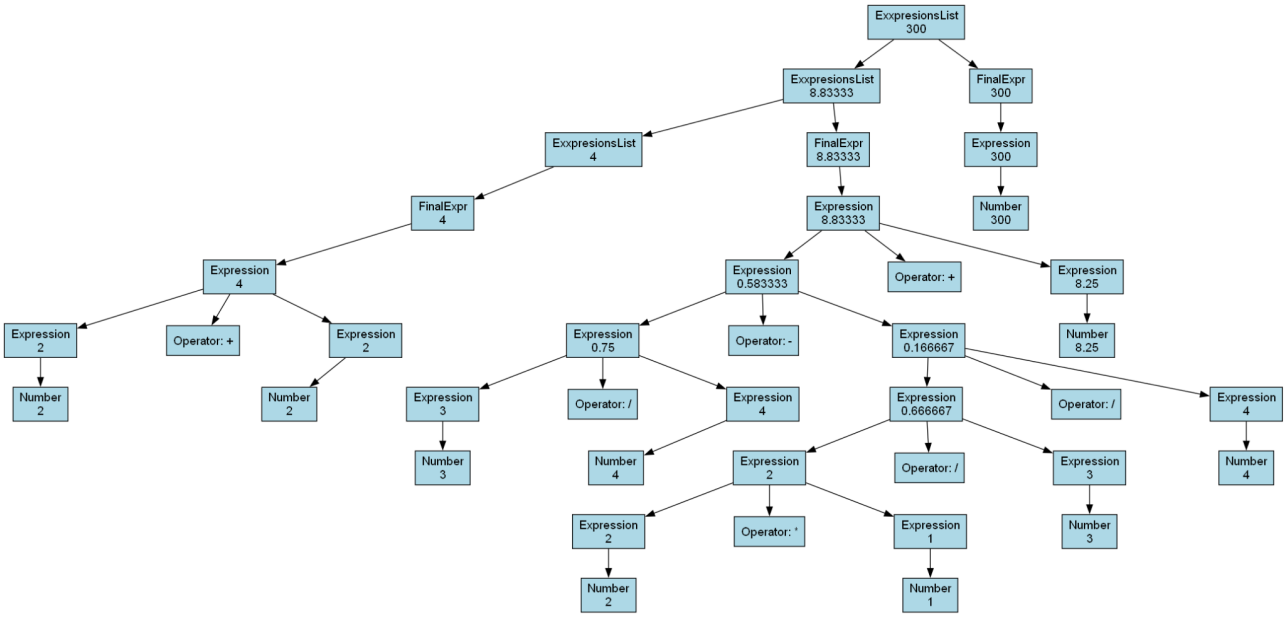
```
2 + 2
3/4 - 2*(1/3/4) + 8.25
300
```

Вхідний файл отримує три арифметичних вирази.

```
FinalExpr 4
```

```
FinalExpr 8.83333
```

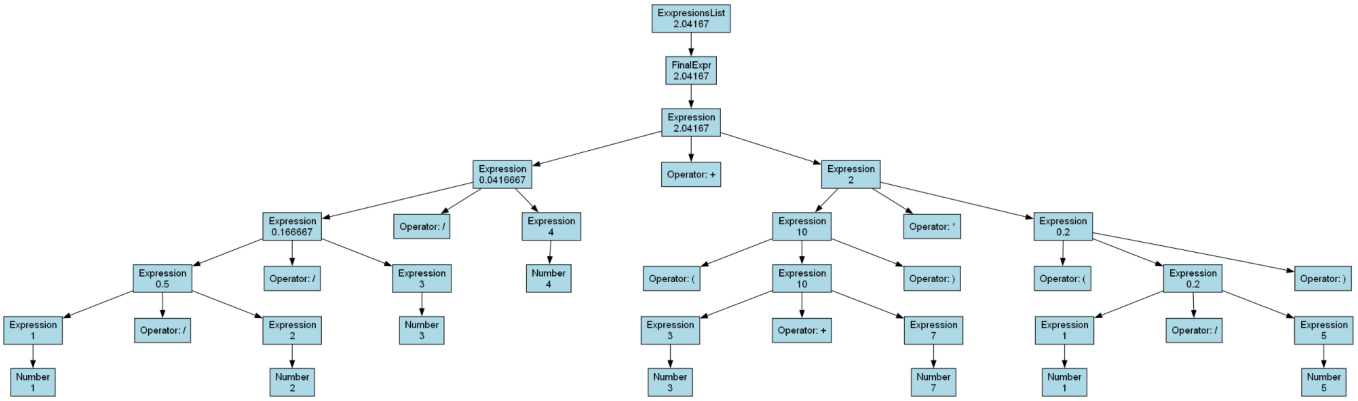
```
FinalExpr 300
```



Побудоване AST дерево.

---

input.txt `1/2/3/4 + (3+7)*(1/5)`



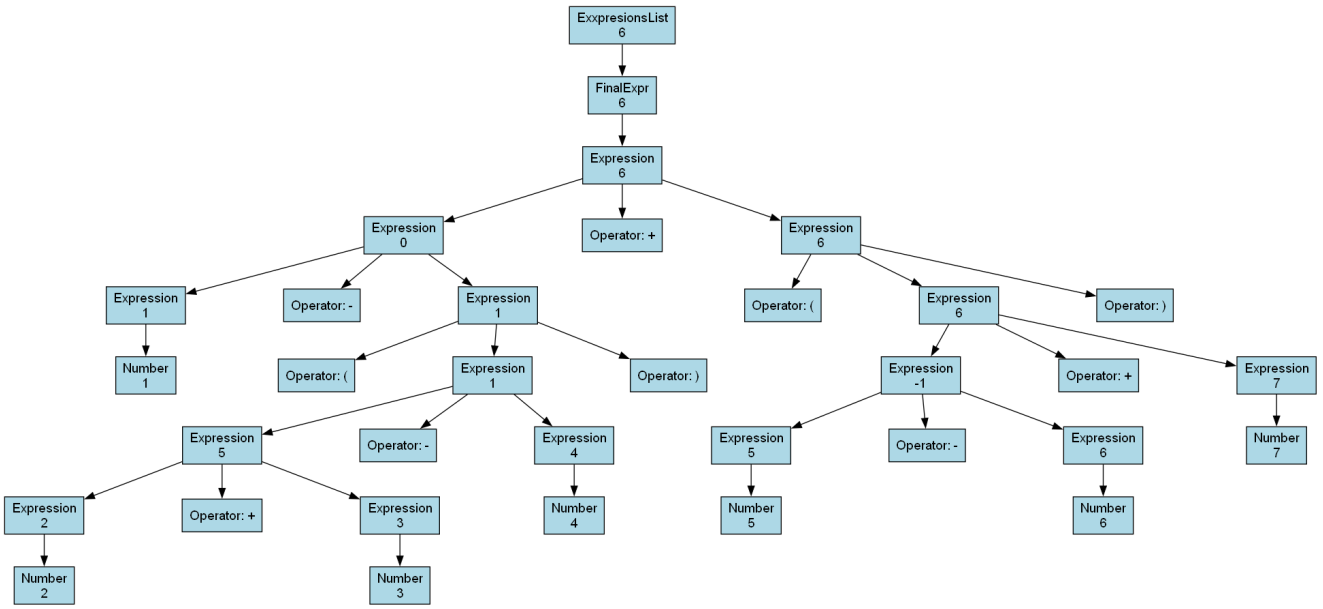

```
Result: 2.04167
AST image generation complete.
```

Дерево побудовано правильно.

---

input.txt `1 - (2 + 3 - 4) + (5 - 6 + 7)`

```
Result: 6
AST image generation complete.
```



Github: https://github.com/DeDTihoN/LexYaccParser