Compilers

Lab Session 2

Advanced calculator grammar

Expr.g4

```
prog:  stat+ EOF ;

stat:  expr NEWLINE          # print
    |  ID '=' expr NEWLINE   # assign
    |  NEWLINE               # blank
    ;

expr:  expr MUL expr         # prod
    |  expr ADD expr         # plus
    |  INT                   # int
    |  ID                    # id
    ;

MUL :  '*' ;
ADD :  '+' ;
ID  :  [a-zA-Z]+ ;       // match identifiers
INT :  [0-9]+ ;          // match integers
NEWLINE: '\r'? '\n' ;    // return newlines to parser
WS  :  [ \t]+ -> skip ;  // toss out whitespace
```

Advanced calculator grammar

Expr.g4

```
prog:  stat+ EOF ;

stat:  expr NEWLINE          # print
    |  ID '=' expr NEWLINE   # assign
    |  NEWLINE               # blank
    ;

expr:  expr MUL expr         # prod
    |  expr ADD expr         # plus
    |  INT                   # int
    |  ID                    # id
    ;

MUL :  '*' ;
ADD :  '+' ;
ID  :  [a-zA-Z]+ ;      // match identifiers
INT :  [0-9]+ ;         // match integers
NEWLINE: '\r'? '\n' ;   // return newlines to parser
WS  :  [ \t]+ -> skip ; // toss out whitespace
```

Rule labels
(Not comments!)

Those are comments

Advanced calculator grammar

main.cpp

```
//////////////////////////////////////////////////////////////////
// Sample "calculator" using visitors
class Calculator : public CalcBaseVisitor {
public:

  // stat : expr NEWLINE    # print
  antlrcpp::Any visitPrint(CalcParser::PrintContext *ctx) {
    int value = visit(ctx->expr());  // evaluate the 'expr' child
    std::cout << value << endl;      // print resulting value
    return 0;                        // return dummy value
  }


  // expr : INT    # int
  antlrcpp::Any visitInt(CalcParser::IntContext *ctx) {
    return std::stoi(ctx->INT()->getText());  // get integer value
  }


 // expr : expr MUL expr    # prod
  antlrcpp::Any visitProd(CalcParser::ProdContext *ctx) {
    int left  = visit(ctx->expr(0));  // get value of left subexpression
    int right = visit(ctx->expr(1));  // get value of right subexpression
    return left*right;                // compute and return result
  }
};
```

Advanced calculator grammar

main.cpp

```
//////////////////////////////////////////////////////////////////
// Sample "calculator" using visitors
class Calculator : public CalcBaseVisitor {
public:

  // stat : expr NEWLINE    # print
  antlrcpp::Any visitPrint(CalcParser::PrintContext *ctx) {
    int value = visit(ctx->expr());  // evaluate the 'expr' child
    std::cout << value << endl;      // print resulting value
    return 0;                        // return dummy value
  }


  // expr : INT    # int
  antlrcpp::Any visitInt(CalcParser::IntContext *ctx) {
    return std::stoi(ctx->INT()->getText());  // get integer value
  }


 // expr : expr MUL expr    # prod
  antlrcpp::Any visitProd(CalcParser::ProdContext *ctx) {
    int left  = visit(ctx->expr(0));  // get value of left subexpression
    int right = visit(ctx->expr(1));  // get value of right subexpression
    return left*right;                // compute and return result
  }
};
```

Rule labels generate diferent visitors for each subrule

Advanced calculator grammar

main.cpp

```
//////////////////////////////////////////////////////////////////
// Sample "calculator" using visitors
class Calculator : public CalcBaseVisitor {
public:
  // "memory" for the calculator; stores current value for each variable
  std::map<std::string, int> memory;

  // stat : ID '=' expr NEWLINE    # assign
  antlrcpp::Any visitAssign(CalcParser::AssignContext *ctx) {
    std::string id = ctx->ID()->getText();  // id is left-hand side of '='
    int value = visit(ctx->expr());         // compute value of expression on right
    memory[id] = value;                     // store it in the memory
    return 0;                               // return dummy value
  }

  // expr : ID    # id
  antlrcpp::Any visitId(CalcParser::IdContext *ctx) {
    std::string id = ctx->ID()->getText();
    if (memory.find(id) != memory.end())
       return memory[id];  // retrieve current variable value
    else
       return 0;           // ...or zero if it does not exist
  }

};
```

We need to store and retrieve values for variables

Advanced calculator grammar

Exercise 4

- Complete the grammar using what you did in exercise 3
  Add missing operators and others like comparisons, conditional ternary, postfix factorial !, sum(e1,e2,...), ...
- Extend the Calculator **visitor** to handle the missing operators and compute the result in each case. Use **rule labels.**
- Extend your Calc language with additional statements (IF, WHILE, ...)

Summary

Key concepts learnt in this session

- How to use rule labels in antlr4 to get cleaner code for our visitors or listeners
- How to use attributes (e.g. `memory` map) to store information that persists and is accesible from any node.