Natural Language Calculator

Formal Languages and Compilers Project 2024/2025

Team members: Elisa Ciardulli, Elisabeth Gruber, Johanna Kümmerer

General explanation:

We built a calculator, which does not get its input as numbers and operations, but as textual commands in (American) English natural language.
Next to the standard operations, our compiler provides the possibility of assignments and insertions, Boolean expressions and relations, while and if functions and measuring units at the end of a number.
Since we extended the files of the lecture, we also implemented a list which works as symbol table and in which Tokens are stored with their associated attributes. The symbol table also contains a type-checker for the tokens which should be stored there. All basic operations like insert, delete and lookup are provided.

Grammar explanation and input examples:

Our compiler can process the following features:

- **Integers** (positive and negative) up to 999,999,999,999
  - `nine hundred ninety nine billion nine hundred ninety nine million nine hundred ninety nine thousand nine hundred ninety nine`
    - the numbers need to be written separately
- **Double**-precision floating-point numbers
  - `nine point nine six five`
    - the number after the point have to be single digits
- **Booleans**
  - `true`
- **Control statements**:
  - While
    - `while open parenthesis true close parenthesis six plus seven`
  - If/else
    - `if open parenthesis true close parenthesis five minus two else seven times four`
- **Mathematical relations**:
  - Smaller (<)
    - `nine smaller than three`
  - Equal (=)
    - `nine equals ten`
  - Greater (>)
    - `seven greater than fifty five`
  - Smaller equal (<=)
    - `seven smaller equal sixty nine`
  - Greater equal (>=)
    - `seven greater equal sixty nine`
  - Different (!=)
    - `seven greater equal sixty nine`

- **Computations**:
  - Addition
    - `six hundred five plus ten`
  - Subtraction
    - `six hundred five minus ten`
  - Multiplication
    - `one hundred times six`
  - Division
    - `one hundred one divided by six`
  - Exponentiation
    - `six point zero three to the power of two`
  - Factorial
    - `twelve factorial`
  - Negative numbers
    - `negative twelve`
      - notice the difference between minus and negative
- **Measurements**:
  - Weight (kilograms, grams, milligrams, pounds, tons)
    - `three pounds`
  - Time (days, hours, seconds, milliseconds)
    - `five seconds`
  - Space (kilometers, centimeters, millimeters, miles, feet, inches)
    - `three point three meters to the power of two`
      - this is 3.3m^2
  - Volume (liters, milliliters)
    - `twenty five liters`
  - It also supports mathematical operations:
    - `one meter plus three centimeters`
    - `one second minus sixty minutes`
    - `seven meters times two miles`
    - `seven meters divided by seven meters`
    - `seven factorial meters`
    - `negative five seconds`
    - `five times five seconds`
    - `five divided by five seconds`
    - `five seconds times five`
    - `five times five seconds`
    - `open parenthesis three meters close parenthesis to the power of two`
- **Percentage calculations**
- **Parenthesis hierarchy**
  - `three times open parenthesis one plus three close parenthesis`
- **Variable assignments** via symbol table, supporting:
  - Integers
    - `integer a is two`
  - Doubles
    - `double a is three point three`
  - Booleans
    - `boolean aB is true`

        - Boolean variables have to end in B
    - Computations
        - `a plus b`
    - Identifiers
        - `integer a`
    - Control statements
        - `if open parenthesis a equals one close parenthesis a plus one else a plus two`
    - Any of the above expressions enclosed in brackets

Where you can:

- Add new variables
    - `integer a is two`
        - the name of the variable cannot be a reserved word, e.g. number, operation etc.
- Change existing variables
    - `a is five`
- Retrieve existing variables
    - `a`

Keep in mind that our grammar is case sensitive.

How to start using our compiler:

1. Make sure all necessary packages are installed
   ```
   sudo apt update
   sudo apt install flex bison build-essential
   ```
2. Build the project
   ```
   Make
   ```
3. Run the project
   ```
   ./natLangCalc
   ```
4. Type the desired input
   ```
   Twenty five plus twenty five
   ```
5. Wait for the result
   ```
   Result: fifty
   ```
6. Keep inserting expressions and click ctrl-C to exit

Our grammar:

line → assignment | line assignment | insertation | line insertation | statements | line statements | expr | line expression | quantity | line quantity | boolexpr | line boolexpr

number → integer | integer POINT fraction

integer → group | group hundrets | group hundrets integer

group → UNIT | TEEN | TEN | TEN UNIT | UNIT HUNDRED | UNIT HUNDRED group

hundrets → THOUSAND | MILLION | BILLION

fraction → UNIT | UNIT UNIT | UNIT UNIT UNIT

expr → number | expr PLUS expr | expr MINUS expr | expr TIMES expr | expr DIVIDE expr | Expr FACT | NEG expr | MINUS expr | expr EXP expr | LPAREN expr RPAREN | ID

quantity → number MEASURE | quantity PLUS quantity | quantity MINUS quantity | quantity TIMES quantity | quantity DIVIDE quantity | quantity EXP expr | expr TIMES quantity | expr DIVIDE quantity | quantity TIMES expr |

quantity DIVIDE expr | expr FACT MEASURE | NEG expr MEASURE |
MINUS expr MEASURE | LPAREN parenquantity RPAREN | parenquantity EXP expr

parenquantity → LPAREN quantity RPAREN

assignment → ID ASSIGN expr | BOOL_ID ASSIGN boolexpr

insertation → INT ID ASSIGN expr | DOU ID ASSIGN expr | BOOL BOOL_ID ASSIGN boolexpr

statements → WHILE LPAREN boolexpr RPAREN expr |
IF LPAREN boolexpr LPAREN expr ELSE expr

boolexpr → BOOLEAN | BOOL_ID | expr RELATION expr