Programming Language Project

Name: Saar Amikam

ID: 314967472

Course: Programming Language

Lecturer: Dr. Sharon Yalov-Handzel

BNF Grammer

<program> ::= <statement_list>

<statement_list> ::= <statement> {";"} <statement_list> | ε

<statement> ::= <if_statement> | <while_statement> | <assignment_statement> | <expr>

<if_statement> ::= "if" <expr> "then" <expr> ["else" <expr>]

<while_statement> ::= "while" <expr> "do" "{" <statement_list> "}"

<assignment_statement> ::= <identifier> "=" <expr>

<expr> ::= <expr> "+" <term>| <expr> "-" <term>| <term>

<term> ::= <term> "*" <factor>| <term> "/" <factor> | <factor>

<factor> ::= <factor> ">" <primary>| <factor> "<" <primary>| <factor> "==" <primary>| <primary>

<primary> ::= <number>| <identifier>| "(" <expr> ")"

<identifier> ::= [a-zA-Z_]{1,4} // Restricts identifiers to 1 or 4 letters as per your specification

<number> ::= [0-9]+

Memory Management Specifications

1. Maximum Code Length

The programming environment is designed to handle programs with a maximum length of 100 commands per line. This limit ensures that the program remains manageable and that the parsing process is efficient, preventing potential performance degradation due to excessively long lines of code.

Specification:

- **Maximum Commands per Line:** 100

2. Maximum Calculation Result

The environment restricts the size of calculation results to prevent integer overflow and ensure consistent computational accuracy across different systems. The maximum allowable result of any single calculation cannot exceed the value of 2,147,483,647, which is the upper limit for a 32-bit signed integer.

Specification:

- **Maximum Calculation Value:** 2,147,483,647

3. Maximum Variable Count

To prevent excessive memory usage and ensure efficient memory allocation, the programming environment limits the number of variables that can be declared within a single session. This limit helps maintain an optimal balance between resource usage and program complexity.

Specification:

- **Maximum Number of Variables:** 50

4. Overflow Handling

In cases where these limits are exceeded, the programming environment is designed to handle overflows gracefully:

- **Code Length Exceedance:** The system will reject input lines that exceed the maximum command count, issuing a syntax error notification to the user.
- **Calculation Overflow:** Any calculation exceeding the defined maximum value will result in an error, preventing the operation from being carried out and alerting the user to the invalid operation.
- **Variable Count Exceedance:** If a program attempts to declare more than the allowed number of variables, an exception will be raised, informing the user that the maximum variable limit has been reached.

5. Variable Naming Rules

Variables within the programming environment must adhere to the following naming conventions:

- **Length:** Variable names can consist of only one or two English letters (a-z, A-Z).
- **Characters:** Only alphabetical characters are permitted. Numeric or special characters are not allowed in variable names.

Discussion of design decisions and trade-offs

1. Control Structures

- Decision: The inclusion of traditional control structures like if-then-else and while loops.
- Trade-off: These structures are familiar and intuitive to programmers coming from imperative backgrounds, making the language more accessible. However, this might limit the language's expressiveness and elegance in handling more complex functional programming patterns, such as those involving recursion or higher-order functions.

2. Syntax and Semantic Choices

- Decision: The syntax is designed to be straightforward with a clear resemblance to established languages, using recognizable symbols and constructs (if, while, braces for blocks, etc.).
- Trade-off: This makes the language easier to learn and use, but it might also inherit some of the limitations and complexities of traditional languages, such as error-prone block management with braces and semicolons.

3. Memory Management

- Decision: Implementing limits on variables, calculation lengths, and program size.
- Trade-off: Setting these limits helps in managing the language's complexity and ensuring that programs do not consume excessive resources, which is particularly important for languages used in constrained environments. On the downside, these limits might restrict the development of more complex programs or algorithms.

4. Error Handling

- Decision: The language incorporates basic error handling by throwing exceptions for syntax and runtime errors.
- Trade-off: While exceptions can make error handling cleaner by separating error management from business logic, they also require the programmer to have a good understanding of exception management techniques, which can complicate program design and testing.

Example programs and test cases

1. Setting Variables

```
x = 10; y = 20;
```

2.printing Variables Values

```
x=5;y=7;z=8;
x;y;z; // // {'x': 5, 'y': 7, 'z': 8}
```

3. Addition and Subtraction Operations Without Variables

```
3 + 5; 12 - 7; // [8, 5]
```

4. Multiplication and Division Operations Without Variables

```
6 * 8; 49 / 7; // [48, 7]
```

5. Addition and Subtraction Operations With Variables

```
a = 15;b = 5;
resA = a + b;  // resA should be 20
resB = a - b;  // resB should be 10
```

6. Multiplication and Division Operations With Variables

```
m = 9; n = 3;
resA = m * n;  // resA should be 27
resB = m / n;  // resB should be 3
```

7. Using operators '>','<','=='

```
x=5; y=3;
x==y // false
x > y // true
y < 8 // true
```

8. if-then Command

```
x = 5;
if x > 3 then y = 2;  // {'x': 5, 'y': 2}
```

9. if-then-else Command

```
x = 2;
if x > 3 then y = 2 else y = 3;  // {'x': 2, 'y': 3}
```

10.if-then 3 levels deep Command.

```
if 5>3 then if 3>2 then if 4>5 then x=2 else x=0  // (works also with variables) {'x': 0}
```

11. while-do Command

```
x = 0;
while x < 5 do {x = x + 1; } // increments x until x equals 5
```

12. while-do Command with a statement_list (Using Brackets {})

```
x = 0; z = 10;
while x < 5 do {x = x + 1;  z = z - 1;}
```

13.More complex Commands

```
x=0;
if 5>3 then while x<3 do {x=x+1; y=x*2; z=y+x;}    // {'x': 3, 'y': 6, 'z': 9}
```

```
x=0; y=2;
while x<3 do{if y > 3 then y=y+1 else x=x+1}   // {'x': 3, 'y': 2}
```

```
x=0; y=0; z=0;
while x < 2 do {x=x+1;z=z+1; while y < 2 do { y=y+1;z=z+1}} //{'x': 2, 'y': 2, 'z': 4}
```