DOCUMENTATION

Language :TEMP

Language Creators

- Pranay Tarigopula (2018A7PS0237H)
- Dhruv Adlakha (2018A7PS0303H)
- Pranav Reddy Pesaladinne (2018A7PS0238H)
- Donkada Vishal Dheeraj (2018A7PS0239H)

How to Run the Lexer

- The executable has been provided and can be run by typing ./Lexer into the terminal. Note that this is only applicable to Linux systems.
- If on windows, you will first need to compile the program on your own. The Lexer was written in C++ so it must be compiled with g++ or an equivalent.
- The executable must be run with filename as the only command line argument.
- Note that the filename must have an extension of .temp, otherwise it will throw an error.
- Ex: The command “./Lexer filename.temp” (Without quotes) would run the Lexer on the file “filename.temp”.

Basic program

```
int main (int argv, char argc)
{
        print(“Hello world”);
        return 0;
}
```

Output: Hello world

Keywords

- int - integers
- char - characters
- bool - true / false values
- float - numerical values having decimal points
- string - string data type
- if - conditional execution
- else - condition executed if ‘if’ statement condition fails
- print - prints to stdout
- for - initiates for loop
- true - condition is correct
- false - condition is incorrect

- return - return the calling function
- function - represents the function start
- main - main function called by the operating system for execution.

Data types

- int - integers
- bool - boolean values
- float - decimal values
- char - characters
- string - “ “ double quotes for representing string literals

Identifiers

- Contains alpha-numeric values and underscores.
- Can start with alphabets or underscore.
- Keywords are not allowed.

Operators

- Arithmetic Operators
    - \+ Addition
    - \- Subtraction
    - * Multiplication
    - / Division
    - % Remainder (Modulo)
    - << Left shift
    - >> Right shift
    - // Divide and take floor (Integer division)
    - ** Exponentiation

- Logical Operators
    - && Logical and (returns true if both conditions are true)
    - || Logical or (returns true if atleast one condition is true)

- Unary Operators

    - ++ i=i+1
    - -- i=i-1
    - + Unary plus (e.g +10)
    - - Unary minus(e.g -10)
    - ! Not operator

- Comparators
    - < a<b
    - > a>b
    - == a==b
    - >= a>=b
    - <= a<=b
    - != a!=b

- Assignment
    - = a=b (Assigns a the value of b)

- Special symbols
    - ( ) parentheses-used in functions & multileveled expressions
    - *{ }* curly braces (function bodies, loop bodies)
    - *;* Semicolon (end of statement)
    - *,* Comma - used to separate parameters in functions

Conditional and iterative operations

- If :

```
if(condition){
        //statements
}
```

- If-else :

```
if(condition){
        //statements
}
else{
        //statements
}
```

- for loop :

```
for(initialisation; condition; assignment operations){
    //statements
}
```

Functions

- Function declaration

```
function functionName(parameter list){
    //statements
    [return statement];
}
```

- Function calls

```
functionName(argument list)
```

Comments

- #....# Comment start - end (will be ignored by the Lexer)

Additional Information

- Anything outside the alphabet will throw a lexical error (`^@$).
- String literals **must** be enclosed within “ ”.
- There cannot be any leading zeros in Integer and Float literals (023 or 01.23) unless the numeric part of the number is equal to 0 (0.0).
- There **must** be at least one digit after a decimal point for floating points numbers (1. Is invalid, 1.0 is valid).
- The Lexer will differentiate a unary operator from an arithmetic operator based on the context (eg: ‘+’ and ‘-’ can either be unary or arithmetic).

Sample Program

```
function factorial (n) {
    int factorial_value=1;
    for (int i=1; i<=n; ++i)
    {
        factorial_value=factorial_value * i;
    }
```

```
        return factorial_value;
}
int main(){
        print("Finding the factorial of 10");
        int result = factorial(10);
        print(result);
}
```