**Тема 10:** Интерпретатор за функционалния език **ThisFunc**.

Интерпретаторът трябва да може да се пуска в интерактивен режим, в който позволява на потребителя да пише ред код, който интерпретаторът да оценява и да принтира резултат. Също така вашият интерпретатор трябва да може да се стартира върху файлове които да може да се изпълняват и резултатът от тях да се отпечатва на изхода на вашата програма.

Нека разгледаме как се дефинира езика **ThisFunc**. В този език има само един тип литерали и те са реални числа които ще отбелязваме в това описание с <real-number>. За да е функционален този език ще трябва да може да дефинираме и изпълняваме функции. Изпълнението на една функция ще става чрез нейното име, и списъкът от аргумент на функцията заграден в кръгли скоби. Като може да има функции без аргументи (с празен списък с аргументи).

```
<expression> ::= <real-number> | <function-call>
<function-call> ::= <function-name>([<expression>, ...])
```

Вашият интерпретатор трябва да осигури няколко на брой предварително имплементирани функции, които могат да се извикват наготово: **add, sub, mul, div, pow, sqrt, sin, cos, eq, le, nand**.

- **nand**(#0, #1) връща булевата оценка на `!#0 || !#1`
- **eq**(#0, #1) връща булевата оценка на `#0 == #1`

Освен това езикът трябва да може да позволява да се декларират функции - с име и израз съдържащ техните аргументи. Аргументите на функцията ще се дефинират с цяло число, индексът на аргумента, предхождащ се от символа ‘#’

```
<param-expression> ::= <expression> | #integer | <function-name>([<param-expression>
, ...])

<function-declaration> ::= <function-name> <- <param-expression>
```

Оценката на декларация на функция не връща резултат.

Трябва да поддържате и няколко специални функции:

1. Функцията **if,** приемаща 3 аргумента: `<test>, <if-true>, <if-false>`. Тази функция трябва да изчисли първия аргумент и ако той е различен от 0, то резултатът трябва да е втория аргумент. Ако първият аргумент се оценява на 0, то функцията трябва да върне третия си аргумент.
2. Функцията **list,** приемаща произволен брой аргументи. Тя се оценява на списък от аргументите си.
3. Функцията **head,** която приема един аргумент от тип списък и връща първия елемент на списъка.
4. Функцията **tail,** която приема един аргумент от тип списък и връща нов списък, чиито елементи са елементите на подаденият но без първият.
5. Функцията **map,** която приема 2 аргумента `<function-name>, <list>`. Резултатът от тази функция с списък с брой елементи колкото елементи има в подадения списък (втория аргумент), и стойността на всеки елемент е резултата от оценяването на подадената функция върху стойността от входния списък.
6. Функцията **concat,** която приема 2 аргумента от тип списък и връща нов списък, чиито елементи са елементите на първия списък и към тях са добавени елементите от втория списък

Във някои случаи функциите като **if** и **nand** не е нужно да изчисляват стойността на всичките си аргументи за да изчислят резултатът си.

При четене и изпълнение на кода могат да възникнат няколко типа грешки - грешки при използването на не-декларирани функции, използването на невалидни символи в имената на функциите, или грешки при изпълнение на кода (например деление на нула). За всяка грешка трябва да се погрижите да се изпише подходящо съобщение.

Обърнете внимание при декларирането и оценяване на функциите е възможно да се получи рекурсия която трябва да поддържате.

Нека разгледаме няколко примера:

- Извикване на функцията за събиране на две числа:
  ```
  add(3, 7)
  > 10
  ```
- Вложено извикване на функцията за събиране:
  ```
  add(add(3, 5), add(10, 10))
  > 28
  ```
- Деклариране и извикване на функция без аргументи, която винаги връща константа:
  ```
  myConst <- 7
  >
  myConst()
  > 7
  ```
- Деклариране и извикване на функция с аргументи:
  ```
  doubleArg <- add(#0, #0)
  >
  doubleArg(5)
  > 10
  ```
- Деклариране и извикване на по сложна функция с аргументи:
  ```
  sumSqr <- add(mul(#0, #0), mul(#1, #1))
  >
  sumSqr(5, 10)
  > 125
  ```
- Използване на условната функция:
  ```
  if(1, 7, 11)
  > 7
  if(0, 7, 11)
  > 11
  if(add(5, -5), 3, 5)
  > 5
  ```
- Използване на списъци:
  ```
  myList <- list(1, 2, 3, 4)
  >
  myList
  > [1, 2, 3, 4]
  double <- mul(#0, #0)
  >
  map(double, myList)
  > [1, 4, 9, 16]
  concat(list(1, 2), list(3, 4))
  ```

```
> [1, 2, 3, 4]
```

- Рекурсия:

```
fact <- if(eq(#0, 0), 1, mul(#0, fact(sub(#0, 1))))
>
fact(0)
> 1
fact(5)
> 120
```