Construction of data buildings:

First, you must implement the following data structure (the same Data structure!) according to the instructions given (!!).

do it**In implementations, you can use common data types () in the class definition use Generic**

1- Implement the Linked List data structure according to the topics of the lesson. The linked list is created.

It should have the following methods:

- Adds to the beginning of the linked list element e : Void addFirst(Element e)
- Adds to the end of the linked list element e : Void addLast(Element e)
- Deletes the first element of the list (if there is an element): Void deleteFirst( )
- Deletes the elements of the list (if there is a member): Void deleteLast( )
- NULL returns the first element of the list (if the list is empty: Element getFirst() returns).
- NULL returns the element at the end of the list (if the list is empty: Element getLast() returns).
- Returns the number of elements in the list: Long int getSize( )
- It clears the list: Void clear( )
- returns false otherwise true if the list is empty: Boolean isEmpty( )

All linked list operations must have time complexity)O(1**be done.**

2- Using the linked list, create the Queue data structure with the following functions:

- Adds element e to the beginning of the queue: Void enqueue(Element e)
- Removes the element at the beginning of the queue and returns it (if there is an existing element: Element dequeue() be).
- NULL returns the first element of the queue (if the queue is empty: Element getFirst() returns).
- Returns the number of members in the queue: Long int getSize( )
- It clears the queue: Void clear( )
- returns false otherwise true if the queue is empty: Boolean isEmpty( )

All queue operations must have time complexity)O(1**be done.**

3- You can use a queue or a linked list to build a stack. The stack should support the following functions slow:

- Removes the top element of the stack and returns it (if the stack is not empty): Element pop()
- puts the element on the top of the stack: Void push(Element e)
- .) returns NULL, if the stack is empty (returns the top element of the stack: Element peek()
- Returns the number of members in the stack: Long int getSize( )
- Empty the stack: Void clear( )
- returns false otherwise true if the stack is empty: Boolean isEmpty( )

All stack operations must have time complexity)O(1**be done.**

The main parts of the program:

The program consists of a number of control commands. Control commands start with # character and everything in The continuation of the command in the same line is related to that command. If you press the Enter key, the current command program considered finished. To create a new command, you have to go to the next line. After running the program, we can use Use the following control commands:

1- The command new# creates a new work environment. All definitions of variables and calculations of previous environments are lost. Pay attention that after running your program, you will not start working with the program without writing the command new#.

2- The define# command is used to define integer and decimal variables and rational functions. Functions have arguments Note that in the definition of variables and functions, in addition to operators and operands, parentheses to determine priority There are also operators and you have to use the expression tree and its algorithms to assign the correct value to functions and variables attribute. Another point is that the variables must be initialized when they are defined.

Mathematical expressions that we know are usually in the formMediationare written**The expressions in this Programs are given by one or more operands, with a number of operators and parentheses. Operands consist of numbers Integer or decimal are functions for a specific value, or variables that have integer or decimal values. are**Operators canInclude Binary. Binary or Unary operators**plus(+)**And**minus(-)**And**multiply(*)**And **Division(/)**And**power(^)**are the only operatorUnary symbol is used in this program**negative (-)**Is.

The general form of using this command is as follows:

<variable name or function name> = <arithmetic expression>#define

Regarding this order, the following rules must be observed:

- If the mathematical expression (both variable initialization and function definition) is meaningless, an error message should be displayed

  Suitable for printing according to the following format:

```
>>err1: wrong exp
```

  In addition, this entire line of the program should be ignored and the normal process of the program should be followed.

- For spacing after writing the word define, only one space is enough, but more spaces

  It is not considered wrong.

- Pay attention that if you use a variable or a function in the program, you must have defined it before

  If a variable or function is used before the definition, the entire executive order should be ignored

  The appropriate error message should be printed on the next line in the following format:

```
>>err2 : undefined exp
```

- Only uppercase and lowercase letters of the English alphabet are allowed in the naming of variables and functions.

- Argument variables of functions are valid only inside the function, and their names are the same as other variables, other functions or

  The variables of other functions do not mean that they are the same.

- The value of variables and the definition of functions can be changed during the execution of the program and after their initial definition; Suppose

  You define a variable named x for the first time and give it a value; For example, the :

```
#define x = 7
```

  Therefore, from now on, x is a variable with a value of 7. Now if the following command is written:

```
#define x = 3
```

  The new value of x is equal to 3. Therefore, changing the value of the variables is also done with the help of this command. This point

  The same is true for functions.

- It is possible that the parentheses or the use of operators and operands in define commands are incorrect

  In this case, the program should display the following error and not consider this line of code and execute it

  The program continues:

```
>>err3: incorrect syntax
```

- In this program, argument functions can also accept values   outside the scope of their definition, and the "definition" value

  In this case, to display the undefined value of the corresponding undefined value expression

  The following format is used:

```
>> undefined value
```

Examples of the define command:

```
#define x = 12
#define y = 13.77
#define f(x) = 3*x – 12/4*3
#define g(x) = 2^x – 1
#define g(x,t) = x*t
#define P(x,y,z) = x-y+z
#define arashYousefi(x,t) = 1/2*̂x 2 + 3*t-1
...
```

3- print # command which is called as follows:

*<variable name>*or*<arithmetic expression>*#print

This command prints the final value of the expression (value) before print in the next line with the following format:

```
>>value
```

As in the previous sections, to output, you must go to the next line and after displaying the "<<" sign, the desired output printed

The statement in front of the print command is a mathematical statement containing a number of mathematical calculations. All kinds of mathematical calculations in this

The software can be used in the following categories:

A) Bracketed expressions including operators (which were explained earlier) and operands. For example:

```
#print 12*3+2
38
#print (2+3)*(3/2)
7.5>>
```

b) Mathematical operations that are performed on exponential functions. Including the following:

- Contraction or expansion and transfer of functions. For example, consider the following piece of code:

```
#define f(x) = 9*x-1
#print 7-f(1)*1/2
#print f(-1)/2-1
```

whose output is as follows:

```
3>>
6->>
```

- Combining functions. In this case, instead of a variable or a number in the argument, a function with a specific value or a mathematical expression complex is placed in the argument. The combination of functions can be done in large numbers. For example, suppose two Let's have function (x) f and (x) g. To combine them from the symbol (f) (x).ºWe do not use and for this operation (g The symbol)g(f(x) and similar is used. For more examples, see the input and output examples.

- addition, subtraction, multiplication and division functions for specific values that are passed to them. For further study, Heresee.

4- The command solve variable # is used to solve equations of the 1st and 2nd degree that are only in terms of one variable. The equation will contain one of the lowercase letters of the English alphabet, which will sit instead of variable in the general format. Form General use of the command:

<variable> <arithmetic equation>#solve

After solving the equation, this command will print the answer or its answers (if any in the set of real numbers) in the form below he does :

>> variable = undefined in real numbers**or**>>variable = answer1, answer2**or**variable = answer

Examples of this command:

```
#solve x x̂ 2-4=0
>>x = 2, -2
#solve x 7-3*x+1 = -1
>>x = 3
```

4- command end# is used to exit the program.

Input and output format:

The program must be in constant communication with the user and receive appropriate responses to various control commands

Also, for some commands that need output, in the Console environment, you must go to the next line and

After displaying the "<<" sign, it printed the desired output.

Input and output examples:

OUTPUT	INPUT
1>> >>err3: incorrect syntax 7>>	#new #define x = 7 #define y = 1 #print y^x #print (2+3)*3/(2) #print x #end
	#define u = 10 #print u+1 #print u*u
8>>	#new #define f(x) = x + 2 #define t = 1 #print f(f(f(f(0)))) #end

3>> 3>>	#new #define g(t) = 2^t-1 #print g(2̂2-1*2) #print g(2̂1^3^3̂12) #end
1>> 1->>	#new #define e(u) = u -1 #define g(t) = 1 #define A(t) = t #print 2-3*1/1 #print A(g(e(1))) #end
1>> >> undefined value 1>>	#new #define r(t) = 1/(x̂ 2-2*x) # print 1 #print r(2) #print 3-2*1 #end

Implementation details:

You must implement the linked list and queue and stack data structures with the desired properties; Even if you don't use them in the project process! Using arrays for implementations is seamless.

Pay attention that it is mandatory to observe one space between the sections mentioned in the general formats of the commands; But the number

More is not considered a wrong distance.