Exercise 10. *Deadline: 25 May 2009.*

COURSE OF C++ PROGRAMMING LANGUAGE

reverse Polish notation

University of Wrocław
Institute of Computer Science *Paweł Rzechonek*

Exercise

Reverse Polish notation (or just *RPN*) is a postfix notation introduced in 1920 by the Polish mathematician *Jan Łukasiewicz*. He developed a formal logic system which allowed mathematical expressions to be specified without parentheses by placing the operators after the operands.

Write a program, which will read, interpret, and evaluate any postfix expression. Your calculator should read commands from the standard input stream `cin` (each command in a separate line), interpret and execute them, and write results to the standard output `cout`. All comments and messages (when problems occurs) should be sent to the standard output for errors `cerr`.

Your program should recognize four commands:

- `set` *var expr*
  Create a new variable *var*, if it doesn't exist yet, and set it to the value of *expr*. You should compute the RPN expression *expr* first and then assign its value to the variable *var*. Use an associative container `map<string,double>` to store variables with their values.

- `print` *expr*
  Compute the value of RPN expression *expr* and write it to the standard output stream. To do so partition the expression *expr* into tokens and place them into the `queue<Computable>` queue (`Computable` is an abstract class that serves as the interface for the derived classes). Use the `stack<double>` stack to store intermediate values during the computation.

- `clear`
  Remove all variables from the associative container. The names of variables have to be different from the names of functions defined in your program (for example: `+`, `-`, `*`, `/`, `pi`, `e`, `abs`, `floor`, `ceil`, `frac`, `min`, `max`, `sin`, `cos`, `atan`, `acot`, `log`, `ln`, `power`, `exp`, etc). You can collect the pairs associatting a function names (an object `string` from STL) with a functions (an object `Function` defined by a programmer) in the container `map<string,Function*>`.

- `end`
  Exit the program.

If an RPN expression is incorrect (the command `set` or `print`) you should throw an exception. Define a class hierarchy for exceptions rooted in the standard library exception class `logic_error` (this exception class has only a constructor `logic_error (const string &what_arg)`).

Create a class hierarchy for computable objects. The class `Computable` should define the basic interface to all `Computable`s and specifies a default implementation that more specific kinds of `Computable`s must override with their own versions (the pure virtual method `value()`). The class `Computable` should be a basic class for all functions, operators, variables and constants. Each class derived from `Computable` has a definition of the method `value()`, which compute a value of type `double`. We can safely invoke this method after accumulating its arguments.

```
class Computable
{
    // ...
public:
    virtual double value () throw (logic_error) = 0;
};
```

Below is a definition of the `Function` class:

```
class Function
{
protected:
    int *arr; // the array of arguments
    int args; // the current number of arguments
public:
    const int arity;
public:
    Function (int ar) throw (logic_error) : arity(ar)
    {
        if (arity<0) throw NegativeNumberOfArguments();
        if (arity>0) arr = new double[arity];
        else arr = 0;
        args = arity;
    }
    virtual ~Function () // virtual destructor
    {
        if (arr) delete []arr;
    }
    Function (const Function &fun) throw () : arity(fun.arity)
    {
        if (arity>0) arr = new double[arity];
        else arr = 0;
        args = arity;
    }
public:
    bool need_arg () const throw ()
    {
        return args>0;
    }
    void add_arg (double a) throw (logic_error)
    {
        if (!need_arg()) throw TooManyArguments();
        arr[--args] = a;
    }
    virtual double value () throw (logic_error) = 0; // the pure virtual method
    /*
        if (need_arg()) throw NeedArguments();
        double result = (compute a value); // a code appropriate to a function
        args = arity;
        return result;
    */
};
```

Suggestion

You can use the *stack* and *queue* data structure from previous exercise. Alternatively, you can use the classes from STL.

Suggestion

Partition your code into the header and source files.

Suggestion

Use the function `getline(istream&we, string&wynik)` to read data from standard input stream line by line.

Hint

Some information about the classes `queue<>`, `stack<>`, `map<>`, and `pair<>` from STL can be found on the webpages:

queue : `http://en.wikipedia.org/wiki/Queue_(data_structure)`
stack : `http://en.wikipedia.org/wiki/Stack_(data_structure)`
map : `http://en.wikipedia.org/wiki/Map_(C%2B%2B_container)`
pair : `http://www.cplusplus.com/reference/std/utility/pair/`

Some information about RPN can be found on the webpage:

`http://en.wikipedia.org/wiki/Reverse_Polish_notation`