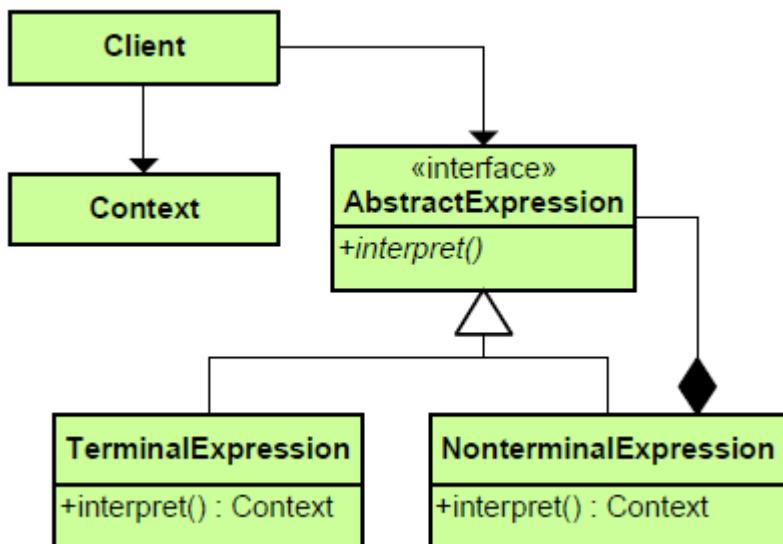


# Interpreter

Behavioral Pattern



[interpreter1.cpp](#)

```
#include <iostream>
#include <string.h>

using namespace std;

class Thousand;
class Hundred;
class Ten;
class One;

class RNInterpreter
```

```
{  
public:  
    RNInterpreter(); // ctor for client  
    RNInterpreter(int){}  
    // ctor for subclasses, avoids infinite loop  
    int interpret(char*); // interpret() for client  
    virtual void interpret(char *input, int &total)  
    {  
        // for internal use  
        int index;  
        index = 0;  
        if (!strncmp(input, nine(), 2))  
        {  
            total += 9 * multiplier();  
            index += 2;  
        }  
        else if (!strncmp(input, four(), 2))  
        {  
            total += 4 * multiplier();  
            index += 2;  
        }  
        else  
        {  
            if (input[0] == five())  
            {  
                total += 5 * multiplier();  
                index = 1;  
            }  
            else  
                index = 0;  
            for (int end = index + 3; index < end; index++)  
                if (input[index] == one())  
                    total += 1 * multiplier();  
                else  
                    break;  
            }  
            strcpy(input, &(input[index]));  
        } // remove leading chars processed  
protected:  
    // cannot be pure virtual because client asks for instance  
    virtual char one(){ return '\0'; }  
    virtual char *four(){ return ""; }  
    virtual char five(){ return '\0'; }  
    virtual char *nine(){ return ""; }  
    virtual int multiplier(){ return 0; }  
private:  
    RNInterpreter *thousands;  
    RNInterpreter *hundreds;  
    RNInterpreter *tens;  
    RNInterpreter *ones;
```

```
};

class Thousand: public RNInterpreter
{
public:
    // provide 1-arg ctor to avoid infinite loop in base class ctor
    Thousand(int): RNInterpreter(1){}
protected:
    char one()
    {
        return 'M';
    }
    char *four()
    {
        return "";
    }
    char five()
    {
        return '\0';
    }
    char *nine()
    {
        return "";
    }
    int multiplier()
    {
        return 1000;
    }
};

class Hundred: public RNInterpreter
{
public:
    Hundred(int): RNInterpreter(1){}
protected:
    char one()
    {
        return 'C';
    }
    char *four()
    {
        return "CD";
    }
    char five()
    {
        return 'D';
    }
    char *nine()
    {
        return "CM";
    }
};
```

```
int multiplier()
{
    return 100;
}

class Ten: public RNInterpreter
{
public:
    Ten(int): RNInterpreter(1){}
protected:
    char one()
    {
        return 'X';
    }
    char *four()
    {
        return "XL";
    }
    char five()
    {
        return 'L';
    }
    char *nine()
    {
        return "XC";
    }
    int multiplier()
    {
        return 10;
    }
};

class One: public RNInterpreter
{
public:
    One(int): RNInterpreter(1){}
protected:
    char one()
    {
        return 'I';
    }
    char *four()
    {
        return "IV";
    }
    char five()
    {
        return 'V';
    }
}
```

```
char *nine()
{
    return "IX";
}
int multiplier()
{
    return 1;
}
};

RNInterpreter::RNInterpreter()
{
    // use 1-arg ctor to avoid infinite loop
    thousands = new Thousand(1);
    hundreds = new Hundred(1);
    tens = new Ten(1);
    ones = new One(1);
}

int RNInterpreter::interpret(char *input)
{
    int total;
    total = 0;
    thousands->interpret(input, total);
    hundreds->interpret(input, total);
    tens->interpret(input, total);
    ones->interpret(input, total);
    if (strcmp(input, ""))
        // if input was invalid, return 0
        return 0;
    return total;
}

int main()
{
    RNInterpreter interpreter;
    char input[20];
    cout << "Enter Roman Numeral: ";
    while (cin >> input)
    {
        cout << "    interpretation is " << interpreter.interpret(input)
<< endl;
        cout << "Enter Roman Numeral: ";
    }
}
```

## interpreter2.cpp

```
#include <iostream>
#include <string>
```

```
#include <map>
#include <list>

namespace wikibooks_design_patterns
{

    // based on the Java sample around here

    typedef std::string String;
    struct Expression;
    typedef std::map<String, Expression*> Map;
    typedef std::list<Expression*> Stack;

    struct Expression {
        virtual int interpret(Map variables) = 0;
        virtual ~Expression() {}
    };

    class Number : public Expression {
    private:
        int number;
    public:
        Number(int number) { this->number = number; }
        int interpret(Map variables) { return number; }
    };

    class Plus : public Expression {
        Expression* leftOperand;
        Expression* rightOperand;
    public:

        Plus(Expression* left, Expression* right) {
            leftOperand = left;
            rightOperand = right;
        }
        ~Plus(){
            delete leftOperand;
            delete rightOperand;
        }

        int interpret(Map variables) {
            return leftOperand->interpret(variables) +
rightOperand->interpret(variables);
        }
    };

    class Minus : public Expression {
        Expression* leftOperand;
        Expression* rightOperand;
    public:

```

```

        Minus(Expression* left, Expression* right) {
            leftOperand = left;
            rightOperand = right;
        }
        ~Minus(){
            delete leftOperand;
            delete rightOperand;
        }

        int interpret(Map variables) {
            return leftOperand->interpret(variables) -
rightOperand->interpret(variables);
        }
    };

class Variable : public Expression {
    String name;
public:
    Variable(String name) { this->name = name; }
    int interpret(Map variables) {
        if(variables.end() == variables.find(name)) return 0;
        return variables[name]->interpret(variables);
    }
};

// While the interpreter pattern does not address parsing, a
parser is provided for completeness.

class Evaluator : public Expression {
    Expression* syntaxTree;

public:
    Evaluator(String expression){
        Stack expressionStack;

        size_t last = 0;
        for (size_t next = 0; String::npos != last; last =
(String::npos == next) ? next : (1+next)) {
            next = expression.find(' ', last);
            String token( expression.substr(last, (String::npos ==
next) ? (expression.length()-last) : (next-last)));

            if (token == "+") {
                Expression* right = expressionStack.back();
                expressionStack.pop_back();
                Expression* left = expressionStack.back();
                expressionStack.pop_back();
                Expression* subExpression = new Plus(right, left);
                expressionStack.push_back( subExpression );
            }
            else if (token == "-") {

```

```
// it's necessary remove first the right operand
from the stack
    Expression* right = expressionStack.back();
expressionStack.pop_back();
    // ..and after the left one
    Expression* left = expressionStack.back();
expressionStack.pop_back();
    Expression* subExpression = new Minus(left, right);
    expressionStack.push_back( subExpression );
}
else
    expressionStack.push_back( new Variable(token) );
}

syntaxTree = expressionStack.back();
expressionStack.pop_back();
}

~Evaluator() {
    delete syntaxTree;
}

int interpret(Map context) {
    return syntaxTree->interpret(context);
}
};

}

void main()
{
    using namespace wikibooks_design_patterns;

    Evaluator sentence("w x z - +");

    static
        const int sequences[][][3] = {
            {5, 10, 42}, {1, 3, 2}, {7, 9, -5},
        };
    for (size_t i = 0; sizeof(sequences)/sizeof(sequences[0]) > i; ++i)
    {
        Map variables;
        variables["w"] = new Number(sequences[i][0]);
        variables["x"] = new Number(sequences[i][1]);
        variables["z"] = new Number(sequences[i][2]);
        int result = sentence.interpret(variables);
        for (Map::iterator it = variables.begin(); variables.end() != it; ++it) delete it->second;

        std::cout<<"Interpreter result: "<<result<<std::endl;
    }
}
```

```
    }  
}
```

[http://en.wikibooks.org/wiki/C%2B%2B\\_Programming/Code/Design\\_Patterns#Interpreter](http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Interpreter)

From:  
<http://obg.co.kr/doku/> - **OBG WiKi**



Permanent link:  
[http://obg.co.kr/doku/doku.php?id=programming:design\\_pattern:interpreter](http://obg.co.kr/doku/doku.php?id=programming:design_pattern:interpreter)

Last update: **2020/11/29 14:09**