Interpreter in C++

Back to **Interpreter** description

Using Interpreter pattern with Template Method

Discussion. Uses a class hierarchy to represent the grammar given below. When a roman numeral is provided, the class hierarchy validates and interprets the string. RNInterpreter "has" 4 sub-interpreters. Each sub-interpreter receives the "context" (remaining unparsed string and cumulative parsed value) and contributes its share to the processing. Sub-interpreters simply define the Template Methods declared in the base class RNInterpreter.

```
 romanNumeral ::= {thousands} {hundreds} {tens} {ones}
thousands, hundreds, tens, ones ::= nine | four | {five} {one} {one} {one}
nine ::= "CM" | "XC" | "IX"
four ::= "CD" | "XL" | "IV"
five ::= 'D' | 'L' | 'V'
one  ::= 'M' | 'C' | 'X' | 'I'
```

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

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(){}
    virtual char *four(){}
    virtual char five(){}
    virtual char *nine(){}
    virtual int multiplier(){}
  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: ";
  }
}
```

Output

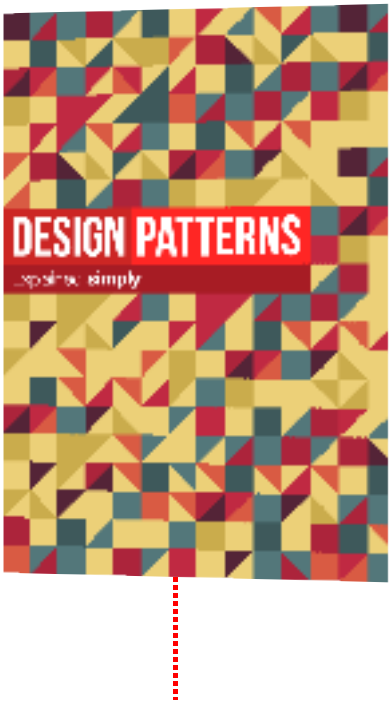
```
Enter Roman Numeral: MCMXCVI
   interpretation is 1996
Enter Roman Numeral: MMMCMXCIX
   interpretation is 3999
Enter Roman Numeral: MMMM
   interpretation is 0
Enter Roman Numeral: MDCLXVIIII
   interpretation is 0
Enter Roman Numeral: CXCX
   interpretation is 0
Enter Roman Numeral: MDCLXVI
   interpretation is 1666
Enter Roman Numeral: DCCCLXXXVIII
   interpretation is 888
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



Learn more

Code examples

Java	Interpreter in Java: Before and after	Interpreter in Java
C++	Interpreter in C++	
PHP	Interpreter in PHP	
Delphi	Interpreter in Delphi	

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us