

Федеральное государственное бюджетное образовательное учреждение
высшего образования Нижегородский государственный технический
университет им. Р.Е. Алексеева

Лабораторная работа №2
«Строковой калькулятор»

Выполнил:
студент 2-го курса
кафедры «Прикладная математика»
Поляков Кирилл

2020г.

Оглавление

Введение.....	3
Документация.....	4
Файл main.cpp.....	4
Содержание main.cpp.....	5
Файл Stack.h.....	6
Содержание Stack.h.....	7
Шаблон класса Stack< T >.....	9
Структура Stack< T >::Node.....	13
Содержание Stack.cpp.....	14
Файл Value.h.....	16
Структура Data.....	17
Класс Value.....	18
Содержание Value.h.....	21
Содержание Value.cpp.....	22
Файл RPN.h.....	23
Класс RPN.....	24
Содержание RPN.h.....	27
Содержание RPN.cpp.....	28
Файл Operation.h.....	31
Пространство имен Operation.....	32
Содержание Operation.h.....	35
Содержание Operation.cpp.....	36
Тестирование.....	38

Введение

Цель работы - написать программу, анализирующую математическое выражение и вычисляющую его значение. Данная программа должна поддерживать базовые арифметические операции (+, -, *, /), поддерживать числа с плавающей запятой, поддерживать переменные в выражениях, поддерживать унарный минус. Для выполнения вычисления нам нужно будет переводить строку, содержащую математическое выражение, в обратную польскую запись и только после этого выполнять само вычисление.

Файл main.cpp

```
#include <iostream>
#include <fstream>
#include "RPN.h"
```

Функции

```
int main (int argc, char *argv[])
```

Функции

◆ main()

```
int main ( int    argc,
           char * argv[]
        )
```

Входная точка программы

Аргументы

argc Количество аргументов, переданных в консоль

argv Аргументы, переданные в консоль

Возвращает

int

См. определение в файле [main.cpp](#) строка **14**

main.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include "RPN.h"
4
5  using namespace std;
6
14 int main(int argc, char* argv[])
15 {
16     string expression = "";
17     for (int i = 0; i < argc; i++)
18     {
19         if ((string("--expression").find(argv[i]) != string::npos) && (i + 1 < argc))
20             expression = argv[i + 1];
21         else if ((string("--variable").find(argv[i]) != string::npos) && (i + 1 < argc))
22         {
23             ofstream fout("Data.txt");
24
25             int beg = i;
26             for (i = i + 1; i < argc; i++)
27             {
28                 if (string(argv[i]).find("=") != string::npos)
29                 {
30                     string variable;
31                     double value;
32
33                     bool fl = false;
34                     string tempValue;
35
36                     for (int b = 0; b < string(argv[i]).length(); b++)
37                     {
38                         if (argv[i][b] == '=')
39                             fl = true;
40                         if (!fl)
41                             variable += argv[i][b];
42                         else if (isdigit(argv[i][b]))
43                             tempValue += argv[i][b];
44                     }
45                     value = atof(tempValue.c_str());
46
47                     fout << variable << "=" << tempValue << endl;
48                     cout << "\"" + variable + "\" = " + tempValue << endl;
49                 }
50                 else
51                 {
52                     cout << "Expression number " << i - beg << " is written
incorrectly!" << endl;
53                 }
54             }
55         }
56     }
57     cout << expression << " = " << RPN::Calculation(expression) << endl;
58 }
59
60
```

Файл Stack.h

```
#include <iostream>
```

Классы

```
class Stack< T >
```

```
struct Stack< T >::Node
```

Stack.h

```
1  #pragma once
2  #include <iostream>
3
4  template <class T>
8  class Stack
9  {
13     struct Node
14     {
15         T element;
16         Node* prev;
17     }*top;
18
19 public:
23     Stack() :top(nullptr)
24     {}
25
29     ~Stack()
30     {
31         while (top)
32         {
33             Node* prev = top;
34             top = top->prev;
35             delete prev;
36         }
37     }
38
44     int Count()
45     {
46         if (IsEmpty())
47             return 0;
48
49         int count = 1;
50         Node* temp = top;
51         while (temp->prev)
52         {
53             count++;
54             temp = temp->prev;
55         }
56         return count;
57     }
58
64     void Push(T element)
65     {
66         Node* newNode = new Node;
67         newNode->element = element;
68         newNode->prev = top;
69         top = newNode;
70     }
71
76     T Pop()
77     {
78         //if (IsEmpty())
79         //    return nullptr;
80
81         T temp = top->element;
82         Node* del = top;
83         top = top->prev;
84         delete del;
85         return temp;
86     }
87
93     T Peek()
94     {
95         if (IsEmpty())
96             return -1;
97         return top->element;
98     }
99
105     bool IsEmpty()
106     {
107         return top ? false : true;
108     }
109 }
```

```
114 void PrintStack()
115 {
116     while (top)
117     {
118         cout << Pop() << ' ';
119     }
120     cout << endl;
121 }
122
123
124 };
125
```


Шаблон класса Stack< T >

```
#include <Stack.h>
```

Классы

```
struct Node
```

Открытые члены

```
Stack ()
```

```
~Stack ()
```

```
int Count ()
```

```
void Push (T element)
```

```
T Pop ()
```

```
T Peek ()
```

```
bool IsEmpty ()
```

```
void PrintStack ()
```

Закрытые данные

```
struct Stack::Node * top
```

Подробное описание

```
template<class T>
class Stack< T >
```

Класс, реализующий стек

См. определение в файле [Stack.h](#) строка 8

Конструктор(ы)

◆ Stack()

```
template<class T >
```

```
Stack< T >::Stack
```

inline

Конструктор класса

См. определение в файле [Stack.h](#) строка 23

◆ ~Stack()

template<class T >

Stack< T >::~~Stack

inline

Деконструктор класса

См. определение в файле **Stack.h** строка **29**

Методы

◆ Count()

template<class T >

int **Stack**< T >::Count

inline

Функция для подсчёта элементов в стеке

Возвращает

Количество элементов в стеке

См. определение в файле **Stack.h** строка **44**

◆ IsEmpty()

template<class T >

bool **Stack**< T >::IsEmpty

inline

Функция, проверяющая стек на пустоту

Возвращает

true-список пустой, false-список не пустой

См. определение в файле **Stack.h** строка **105**

◆ Peek()

```
template<class T >
```

```
T Stack< T >::Peek
```

inline

Функция, возвращающая последний элемент из стека

Возвращает

Последний элемент в стеке

См. определение в файле [Stack.h](#) строка **93**

◆ Pop()

```
template<class T >
```

```
T Stack< T >::Pop
```

inline

Функция, возвращающая последний элемент из стека и удаляющая его из самого стека

Возвращает

Последний элемент в стеке

См. определение в файле [Stack.h](#) строка **76**

◆ PrintStack()

```
template<class T >
```

```
void Stack< T >::PrintStack
```

inline

Функция для вывода стека в консоль

См. определение в файле [Stack.h](#) строка **114**

◆ Push()

```
template<class T >
```

```
void Stack< T >::Push ( T element )
```

inline

Фнкция для добавления элемента в конец стека

Аргументы

element Элемент, который нужно добавить в стек

См. определение в файле [Stack.h](#) строка **64**

Данные класса

◆ top

```
template<class T >
```

```
struct Stack::Node* Stack< T >::top
```

private

Объявления и описания членов классов находятся в файлах:

- **Stack.h**
- **Stack.cpp**

Структура `Stack< T >::Node`

Открытые атрибуты

`T element`

`Node * prev`

Подробное описание

template<class T>
struct Stack< T >::Node

Структура, реализующая элементы стека

См. определение в файле [Stack.h](#) строка **13**

Данные класса

◆ element

template<class T >

`T Stack< T >::Node::element`

См. определение в файле [Stack.h](#) строка **15**

◆ prev

template<class T >

`Node* Stack< T >::Node::prev`

См. определение в файле [Stack.h](#) строка **16**

Объявления и описания членов структуры находятся в файле:

- [Stack.h](#)

Stack.cpp

```
1  #include "Stack.h"
2  #include <iostream>
3
4  using namespace std;
5
6  template <class T>
7  Stack<T>::Stack() :top(nullptr)
8  {
9
10 }
11
12 template <class T>
13 Stack<T>::~~Stack()
14 {
15     while (top)
16     {
17         Node* prev = top;
18         top = top->prev;
19         delete prev;
20     }
21 }
22
23 template <class T>
24 int Stack<T>::Count()
25 {
26     if (IsEmpty())
27         return 0;
28
29     int count = 1;
30     Node *temp = top;
31     while (temp->prev)
32     {
33         count++;
34         temp = temp->prev;
35     }
36     return count;
37 }
38
39 template <class T>
40 void Stack<T>::Push(T element)
41 {
42     Node* newNode = new Node;
43     newNode->element = element;
44     newNode->prev = top;
45     top = newNode;
46 }
47
48 template <class T>
49 bool Stack<T>::IsEmpty()
50 {
51     return top ? false : true;
52 }
53
54 template <class T>
55 T Stack<T>::Pop()
56 {
57     if (IsEmpty())
58         return -1;
59
60     T temp = top->element;
61     Node* del = top;
62     top = top->prev;
63     delete del;
64     return temp;
65 }
66
67 template <class T>
68 T Stack<T>::Peek()
69 {
70     if (IsEmpty())
71         return -1;
72     return top->element;
73 }
```

```
74
75 template <class T>
76 void Stack<T>::PrintStack()
77 {
78     while (top)
79     {
80         cout << Pop() << ' ';
81     }
82     cout << endl;
83 }
84
```

Файл Value.h

```
#include <string>
```

См. исходные тексты.

Классы

struct	Data
class	Value

Структура Data

```
#include <Value.h>
```

Открытые атрибуты

```
double DoubleValue = NULL
```

```
string StringValue = ""
```

Подробное описание

Структура, содержащая данные о типе данных числа

См. определение в файле [Value.h](#) строка **9**

Данные класса

◆ DoubleValue

```
double Data::DoubleValue = NULL
```

См. определение в файле [Value.h](#) строка **11**

◆ StringValue

```
string Data::StringValue = ""
```

См. определение в файле [Value.h](#) строка **12**

Объявления и описания членов структуры находятся в файле:

- [Value.h](#)

Класс Value

```
#include <Value.h>
```

Открытые члены

	Value (string value)
bool	IsString ()
bool	IsDouble ()
int	TypeInfo ()

Открытые атрибуты

Data	data
-------------	-------------

Подробное описание

Класс, реализующий входное значение

См. определение в файле **Value.h** строка **19**

Конструктор(ы)

◆ Value()

Value::Value (string **value**)

Конструктор класса

Аргументы

value Переданное значение

См. определение в файле **Value.cpp** строка **3**

Методы

◆ IsDouble()

```
bool Value::IsDouble ( )
```

Функция для проверки числа на принадлежность к типу double

Возвращает

true, если тип значения является double и false в любом другом случае

См. определение в файле **Value.cpp** строка **16**

◆ IsString()

```
bool Value::IsString ( )
```

Функция для проверки числа на принадлежность к строке

Возвращает

true, если тип значения является строкой и false в любом другом случае

См. определение в файле **Value.cpp** строка **26**

◆ TypeInfo()

```
int Value::TypeInfo ( )
```

Функция, возвращающая тип значения

Возвращает

1-double, 2-в другом случае

См. определение в файле **Value.cpp** строка **36**

Данные класса

◆ data

```
Data Value::data
```

См. определение в файле **Value.h** строка **22**

Объявления и описания членов классов находятся в файлах:

- **Value.h**

- [Value.cpp](#)

Value.h

```
1 #pragma once
2 #include <string>
3
4 using namespace std;
5
9 struct Data
10 {
11     double DoubleValue = NULL;
12     string StringValue = "";
13 };
14
15
19 class Value
20 {
21 public:
22     Data data;
28     Value(string value);
34     bool IsString();
40     bool IsDouble();
46     int TypeInfo();
47 };
```

Value.cpp

```
1  #include "Value.h"
2
3  Value::Value(string value)
4  {
5      char* end;
6      double res = strtod(value.c_str(), &end);
7
8      if (end == value.c_str() || *end != '\0')
9          data.StringValue = value;
10     //data->StringValue = value;
11     else
12         data.DoubleValue = res;
13     //data->DoubleValue = res;
14 }
15
16 bool Value::IsDouble()
17 {
18     /*if (sizeof(data) == sizeof(double))
19         return true;
20     return false;*/
21     if (data.DoubleValue == NULL)
22         return false;
23     return true;
24 }
25
26 bool Value::IsString()
27 {
28     //if (sizeof(data) == sizeof(string))
29     //    return true;
30     //return false;
31     if (data.StringValue == "")
32         return false;
33     return true;
34 }
35
36 int Value::TypeInfo()
37 {
38     if (IsDouble())
39         return 1;
40     return 2;
41 }
42
```

Файл RPN.h

```
#include <string>
```

Классы

```
class RPN
```

Класс RPN

```
#include <RPN.h>
```

Открытые статические члены

static string **Calculation** (string input)

Закрытые статические члены

static bool **IsDelimiter** (char el)

static int **GetPriority** (char el)

static bool **IsOperator** (char el)

static string **Reverse** (string input)

Подробное описание

Класс для выполнения основных операций для вычисления значения

См. определение в файле **RPN.h** строка **9**

Методы

◆ GetPriority()

```
int RPN::GetPriority ( char el )
```

static private

Функция для получения приоритета операции

Аргументы

el символ, содержащий операцию

Возвращает

приоритет операции

См. определение в файле **RPN.cpp** строка **24**

◆ IsDelimiter()


```
bool RPN::IsDelimiter ( char el )
```

static private

Функция, проверяющая символ на принадлежность к разделителям (пробел, =)

Аргументы

el символ, который нужно проверить

Возвращает

true-является разделителем, false-не является

См. определение в файле [RPN.cpp](#) строка **10**

◆ IsOperator()

```
bool RPN::IsOperator ( char el )
```

static private

Функция, проверяющая символ на принадлежность к операторам (пробел, =)

Аргументы

el символ, который нужно проверить

Возвращает

true-является оператор, false-не является

См. определение в файле [RPN.cpp](#) строка **17**

◆ Reverse()

```
string RPN::Reverse ( string input )
```

static private

Функция, переводящая выражение в обратную польскую запись

Аргументы

input строка, содержащая математическое выражение

Возвращает

математическое выражение в обратной польской записи

См. определение в файле [RPN.cpp](#) строка **39**

◆ Calculation()

string RPN::Calculation (string **input**)

static

Функция, вычисляющая значение математического выражения в обратной польской записи

Аргументы

input математическое выражение в обратной польской записи

Возвращает

вычисленное значение

См. определение в файле **RPN.cpp** строка **154**

Объявления и описания членов классов находятся в файлах:

- **RPN.h**
- **RPN.cpp**

RPN.h

```
1 #pragma once
2 #include <string>
3
4 using namespace std;
5
6 static class RPN
7 {
8     static bool IsDelimiter(char el);
9     static int GetPriority(char el);
10    static bool IsOperator(char el);
11    static string Reverse(string input);
12 public:
13    static string Calculation(string input);
14 };
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
```

RPN.cpp

```
1  #include "RPN.h"
2  #include "Stack.h"
3  #include "Operation.h"
4  #include <iostream>
5  #include <fstream>
6  #include "Value.h"
7
8  using namespace Operation;
9
10 bool RPN::IsDelimiter(char el)
11 {
12     if (string(" =").find(el) != string::npos)
13         return true;
14     return false;
15 }
16
17 bool RPN::IsOperator(char el)
18 {
19     if (string("+ - / * ^").find(el) != string::npos)
20         return true;
21     return false;
22 }
23
24 int RPN::GetPriority(char el)
25 {
26     switch (el)
27     {
28     case '(': return 0;
29     case ')': return 1;
30     case '+': return 2;
31     case '-': return 3;
32     case '*': return 4;
33     case '/': return 4;
34     case '^': return 5;
35     default: return 6;
36     }
37 }
38
39 string RPN::Reverse(string input)
40 {
41     if (input.empty())
42         return "Error. The string is empty!";
43
44     string output;
45     Stack<char> operStack;
46
47     string temp = "";
48     for (int i = 0; i < input.length(); i++)
49         if (input[i] != ' ')
50             temp += input[i];
51
52     input = temp;
53
54     for (int i = 0; i < input.length(); i++)
55     {
56         if (IsDelimiter(input[i]))
57             continue;
58
59         if (isdigit(input[i]))
60         {
61             while (!IsDelimiter(input[i]) && !IsOperator(input[i]) && i < input.length())
62             {
63                 output += input[i];
64                 i++;
65             }
66             output += " ";
67             i--;
68         }
69         else if (isalpha(input[i]))
70         {
71             string temp;
72             while (!IsDelimiter(input[i]) && !IsOperator(input[i]) && i < input.length())
```

```

74     {
75         temp += input[i];
76         i++;
77     }
78     ifstream fin("Data.txt");
79     if (fin.is_open())
80     {
81         string str;
82         while (getline(fin, str))
83         {
84             string tempStr;
85             tempStr = str.substr(0, str.find_first_of('='));
86             if (temp == tempStr)
87             {
88                 output += str.substr(str.find_first_of('=')+1, str.length()-1) +
89 " ";
89                 temp = "";
90                 break;
91             }
92         }
93         output += temp;
94     }
95     else
96         cout << "Failed to open the file" << endl;
97     fin.close();
98 }
99
100 if (IsOperator(input[i]))
101 {
102     if (input[i] == '(')
103     {
104         operStack.Push(input[i]);
105     }
106     else if (input[i] == ')')
107     {
108         char s = operStack.Pop();
109         try
110         {
111             while (s != '(')
112             {
113                 output += s;
114                 s = operStack.Pop();
115             }
116         }
117         catch (exception ex)
118         {
119             cout << "Perhaps there is no '('" << endl;
120         }
121     }
122     else
123     {
124         if (!operStack.IsEmpty())
125         {
126             if (GetPriority(input[i]) <= GetPriority(operStack.Peek()))
127             {
128                 output += operStack.Pop();
129                 operStack.Push(input[i]);
130             }
131             else
132                 operStack.Push(input[i]);
133         }
134         else
135             operStack.Push(input[i]);
136     }
137 }
138 }
139 }
140 }
141 while (!operStack.IsEmpty())
142 {
143     char el = operStack.Pop();
144     if (el == '(')
145         cout << "Missing ')" << endl;
146     else
147         output += el;
148 }
149 }
150
151 return output;
152 }

```

```

153
154 string RPN::Calculation(string input)
155 {
156     string result;
157     Stack<string> temp;
158     string RevInput = Reverse(input);
159
160     for (int i = 0; i < RevInput.length(); i++)
161     {
162         if (isdigit(RevInput[i]))
163         {
164             string str;
165
166             while (!IsDelimiter(RevInput[i]) && !IsOperator(RevInput[i]))
167             {
168                 str += RevInput[i];
169                 i++;
170                 if (i == RevInput.length())
171                     break;
172             }
173             temp.Push(str);
174             i--;
175         }
176         else if (IsOperator(RevInput[i]))
177         {
178             if (temp.Count() < 2)
179                 return temp.Pop();
180
181             Value first(temp.Pop());
182             Value second(temp.Pop());
183
184             switch (RevInput[i])
185             {
186
187             case '+':
188                 result = Addition(first, second);
189                 break;
190             case '-':
191                 result = Subtraction(first, second);
192                 break;
193             case '*':
194                 result = Multiply(first, second);
195                 break;
196             case '/':
197                 result = Divide(first, second);
198                 break;
199             case '^':
200                 result = Pow(first, second);
201                 break;
202             }
203             temp.Push(result);
204         }
205     }
206     string str;
207     while (!temp.IsEmpty())
208         str += temp.Pop() + " ";
209     return str;
210 }

```

Файл Operation.h

```
#include <string>
#include "Value.h"
```

Пространства имен

Operation

Функции

```
string Operation::Addition (Value one, Value two)
string Operation::Subtraction (Value one, Value two)
string Operation::Multiply (Value one, Value two)
string Operation::Divide (Value one, Value two)
string Operation::Pow (Value one, Value two)
```

Пространство имен Operation

Функции

string	Addition (Value one, Value two)
string	Subtraction (Value one, Value two)
string	Multiply (Value one, Value two)
string	Divide (Value one, Value two)
string	Pow (Value one, Value two)

Подробное описание

Пространство имён, содержащее базовые математические операции

Функции

◆ Addition()

```
string Operation::Addition ( Value one,
                             Value two
                             )
```

Функция, реализующая операцию сложения

Аргументы

one Аргумент, содержащий первое значение для сложения

two Аргумент, содержащий второе значение для сложения

Возвращает

Строка, содержащее значение после вычисления

См. определение в файле **Operation.cpp** строка **7**

◆ Divide()


```
string Operation::Divide ( Value one,
                          Value two
                          )
```

Функция, реализующая операцию деления

Аргументы

one Аргумент, содержащий первое значение для деления

two Аргумент, содержащий второе значение для деления

Возвращает

Строка, содержащее значение после вычисления

См. определение в файле **Operation.cpp** строка **61**

◆ Multiply()

```
string Operation::Multiply ( Value one,
                             Value two
                             )
```

Функция, реализующая операцию умножения

Аргументы

one Аргумент, содержащий первое значение для умножения

two Аргумент, содержащий второе значение для умножения

Возвращает

Строка, содержащее значение после вычисления

См. определение в файле **Operation.cpp** строка **43**

◆ Pow()

```
string Operation::Pow ( Value one,  
                       Value two  
                       )
```

Функция, реализующая операцию возведения в степень

Аргументы

one Аргумент, содержащий первое значение для возведения в степень

two Аргумент, содержащий второе значение для возведения в степень

Возвращает

Строка, содержащее значение после вычисления

См. определение в файле **Operation.cpp** строка **79**

◆ Subtraction()

```
string Operation::Subtraction ( Value one,  
                               Value two  
                               )
```

Функция, реализующая операцию вычитания

Аргументы

one Аргумент, содержащий первое значение для вычитания

two Аргумент, содержащий второе значение для вычитания

Возвращает

Строка, содержащее значение после вычисления

См. определение в файле **Operation.cpp** строка **25**

Operation.h

```
1 #pragma once
2 #include <string>
3 #include "Value.h"
4
5 using namespace std;
6
10 namespace Operation
11 {
19     string Addition(Value one, Value two);
27     string Subtraction(Value one, Value two);
35     string Multiply(Value one, Value two);
43     string Divide(Value one, Value two);
51     string Pow(Value one, Value two);
52 };
53
54
```

Operation.cpp

```
1  #include "Operation.h"
2
3  #include <iostream>
4
5  using namespace std;
6
7  string Operation::Addition(Value one, Value two)
8  {
9      if (one.TypeInfo() == two.TypeInfo())
10     {
11         if (one.IsDouble())
12         {
13             string temp = to_string(two.data.DoubleValue + one.data.DoubleValue);
14             return temp.substr(0, temp.find_last_not_of('0') + 1);
15         }
16         if (one.IsString())
17             return two.data.StringValue + " + " + one.data.StringValue;
18     }
19     if (one.IsDouble() && two.IsString())
20         return two.data.StringValue + " + " + to_string(one.data.DoubleValue).substr(0,
21 to_string(one.data.DoubleValue).find_last_not_of('0') + 1);
22     return to_string(two.data.DoubleValue).substr(0,
23 to_string(two.data.DoubleValue).find_last_not_of('0') + 1)
24 + " + " + one.data.StringValue;
25 }
26
27 string Operation::Subtraction(Value one, Value two)
28 {
29     if (one.TypeInfo() == two.TypeInfo())
30     {
31         if (one.IsDouble())
32         {
33             string temp = to_string(two.data.DoubleValue - one.data.DoubleValue);
34             return temp.substr(0, temp.find_last_not_of('0') + 1);
35         }
36         if (one.IsString())
37             return two.data.StringValue + " - " + one.data.StringValue;
38     }
39     if (one.IsDouble() && two.IsString())
40         return two.data.StringValue + " - " + to_string(one.data.DoubleValue).substr(0,
41 to_string(one.data.DoubleValue).find_last_not_of('0') + 1);
42     return to_string(two.data.DoubleValue).substr(0,
43 to_string(two.data.DoubleValue).find_last_not_of('0') + 1)
44 + " - " + one.data.StringValue;
45 }
46
47 string Operation::Multiply(Value one, Value two)
48 {
49     if (one.TypeInfo() == two.TypeInfo())
50     {
51         if (one.IsDouble())
52         {
53             string temp = to_string(two.data.DoubleValue * one.data.DoubleValue);
54             return temp.substr(0, temp.find_last_not_of('0') + 1);
55         }
56         if (one.IsString())
57             return two.data.StringValue + " * " + one.data.StringValue;
58     }
59     if (one.IsDouble() && two.IsString())
60         return two.data.StringValue + " * " + to_string(one.data.DoubleValue).substr(0,
61 to_string(one.data.DoubleValue).find_last_not_of('0') + 1);
62     return to_string(two.data.DoubleValue).substr(0,
63 to_string(two.data.DoubleValue).find_last_not_of('0') + 1)
64 + " * " + one.data.StringValue;
65 }
66
67 string Operation::Divide(Value one, Value two)
68 {
69     if (one.TypeInfo() == two.TypeInfo())
70     {
71         if (one.IsDouble())
72         {
73             string temp = to_string(two.data.DoubleValue / one.data.DoubleValue);
```

```

68         return temp.substr(0, temp.find_last_not_of('0') + 1);
69     }
70     if (one.IsString())
71         return two.data.StringValue + " / " + one.data.StringValue;
72 }
73 if (one.IsDouble() && two.IsString())
74     return two.data.StringValue + " / " + to_string(one.data.DoubleValue).substr(0,
to_string(one.data.DoubleValue).find_last_not_of('0') + 1);
75     return to_string(two.data.DoubleValue).substr(0,
to_string(two.data.DoubleValue).find_last_not_of('0') + 1)
76         + " / " + one.data.StringValue;
77 }
78
79 string Operation::Pow(Value one, Value two)
80 {
81     if (one.TypeInfo() == two.TypeInfo())
82     {
83         if (one.IsDouble())
84         {
85             string temp = to_string(pow(two.data.DoubleValue, one.data.DoubleValue));
86             return temp.substr(0, temp.find_last_not_of('0') + 1);
87         }
88         if (one.IsString())
89             return two.data.StringValue + "^" + one.data.StringValue;
90     }
91     if (one.IsDouble() && two.IsString())
92         return two.data.StringValue + "^" + to_string(one.data.DoubleValue).substr(0,
to_string(one.data.DoubleValue).find_last_not_of('0') + 1);
93
94     return to_string(two.data.DoubleValue).substr(0,
to_string(two.data.DoubleValue).find_last_not_of('0') + 1)
95         + "^" + one.data.StringValue;
96 }

```

```
'a' = 4  
'b' = 226  
10^a+b/2 = 10113.
```

```
'a' = 153  
'b' = 3  
-153+20*b = -93.
```