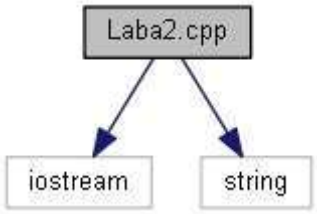
Laba2.cpp File Reference

Файл реализующий вторую лабораторную работу. More...

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

Include dependency graph for Laba2.cpp:



Classes

struct	**Sign** Структура знака More...

Functions

int	**Count** (**Sign** *stack) Функция подсчитывающая количество элементов More...
string	**Calculation** (string FirstNumber, string SecondNumber, string sign) Функция выполняющая математические действия More...
void	**PushBack** (string *arr, const string &number, int &i) Функция кладущая строку в конец массива More...
void	**PushBack** (**Sign** *arr, **Sign** &sign, int &i) Функция кладущая знак в конец массива More...
Sign	**PopBack** (**Sign** *arr, int &i) Функция выдающая знак из конца массива и удаляет его More...
string	**PopBack** (string *arr, int &i) Функция выдающая строку из конца массива и удаляет ее More...
string	**RPN** (string *tokens, int countOfTokens, string *variables, int countOfVariables) Функция выдающая строку из конца массива и удаляет ее More...
string	**Fragmentation** (string expression, string *variables, int countOfVariables) Функция разбивающая строку на токены More...
string	**Parsing** (int argc, char *argv[]) Функция парсящая входные строки More...
int	**main** (int argc, char *argv[]) Основная функция программы More...

Detailed Description

Файл реализующий вторую лабораторную работу.

Задача: Написать программу, анализирующую математическое выражение и вычисляющую его значение.

Function Documentation

- Calculation()

```
string Calculation ( string  FirstNumber,
                     string  SecondNumber,
                     string  sign
                   )
```

Функция выполняющая математические действия

Parameters

FirstNumber	– строка с первым числом
SecondNumber	– строка со вторым числом
sign	– знак операции

Returns

string - результат операции

Код функции выглядит следующим образом:

```
string Calculation(string FirstNumber, string SecondNumber, string sign)
{
    double first = stod(FirstNumber);
    double second = stod(SecondNumber);
    double result;
    if (sign == "+")
    {
        result = first + second;
    }
    else if (sign == "-")
    {
        result = first - second;
    }

    else if (sign == "/")
    {
        result = first / second;
    }

    else if (sign == "*")
    {
        result = first * second;
    }

    return to_string(result);
}
```

Authors

Tereshin D.D

Here is the caller graph for this function:

◆ Count()

int Count (Sign * stack)

Функция подсчитывающая количество элементов

Parameters

stack – указатель на элемент типа Sign

Returns

int - количество элементов

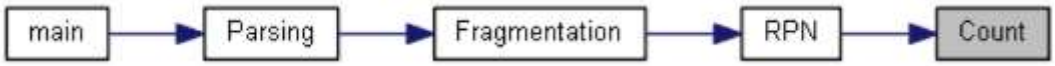
Код функции выглядит следующим образом:

```
int Count(Sign* stack)
{
    int i = 0;
    while (true)
    {
        if (stack[i].value == "")
            return i;
        i++;
    }
}
```

Authors

Tereshin D.D

Here is the caller graph for this function:



◆ Fragmentation()

```
string Fragmentation ( string   expression,
                       string * variables,
                       int      countOfVariables
                     )
```

Функция разбивающая строку на токены

Parameters

expression – строка выражения

variables – массив строк переменных

countOfVariables – количество строк переменных

Returns

string - возращаемое значение функции RPN

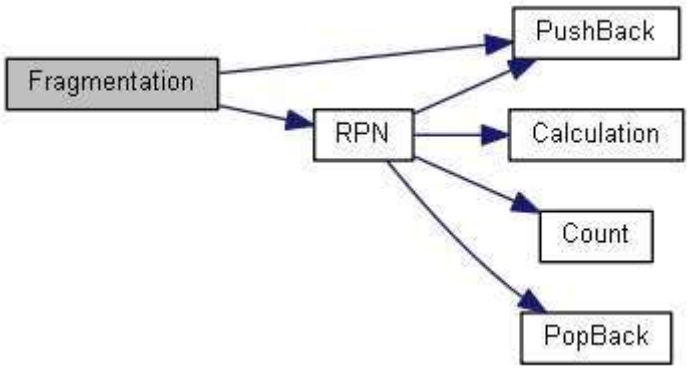
Код функции выглядит следующим образом:

```
string Fragmentation(string expression, string* variables, int countOfVariables)
{
    string *tokens = new string[expression.length()];
    string number;
    int j = 0;
    for (int i = 0; i < expression.length(); i++)
    {
        switch (expression[i])
        {
        case '+':
            PushBack(tokens, number, j);
            PushBack(tokens, "+", j);
            number.clear();
            break;
        case '-':
            PushBack(tokens, number, j);
            PushBack(tokens, "-", j);
            number.clear();
            break;
        case '/':
            PushBack(tokens, number, j);
            PushBack(tokens, "/", j);
            number.clear();
            break;
        case '*':
            PushBack(tokens,number, j);
            PushBack(tokens, "*", j);
            number.clear();
            break;
        case '(':
            PushBack(tokens, number, j);
            PushBack(tokens, "(", j);
            number.clear();
            break;
        case ')':
            PushBack(tokens, number, j);
            PushBack(tokens, ")", j);
            number.clear();
            break;
        default:
            number += expression[i];
            break;
        }
    }
    PushBack(tokens, number, j);
    return RPN(tokens, j, variables, countOfVariables);
}
```

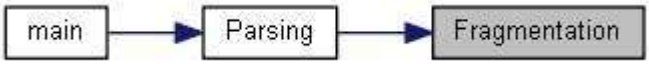
Authors

Tereshin D.D

Here is the call graph for this function:



Here is the caller graph for this function:



◆ main()

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

Основная функция программы

Parameters

argc – количество передающихся параметров

argv[] – входные строки

Returns

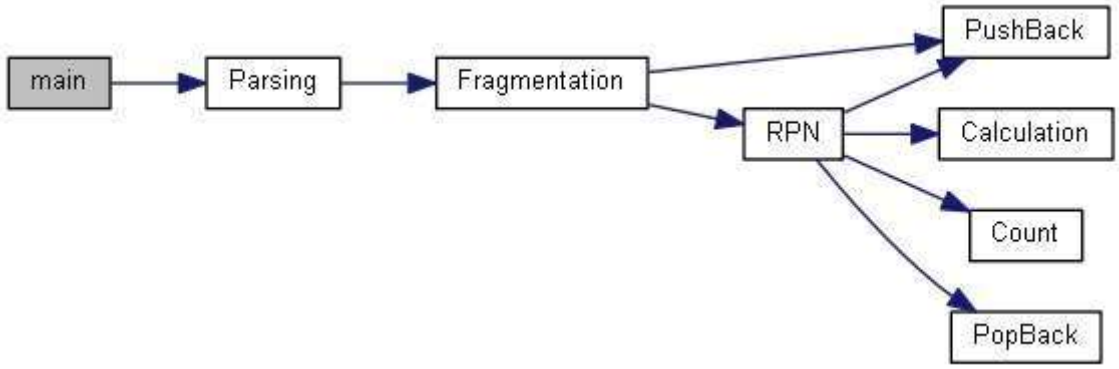
int

```
int main(int argc, char* argv[])
{
    cout << "result: " << Parsing(argc, argv) << endl;

    return 0;
}
```

Authors

Tereshin D.D

Here is the call graph for this function:



◆ Parsing()

string Parsing (int argc,
char * argv[]
)

Функция парсящая входные строки

Parameters

argc – количество передающихся параметров

argv[] – входные строки

Returns

string - возращаемое значение функции Fragmentation

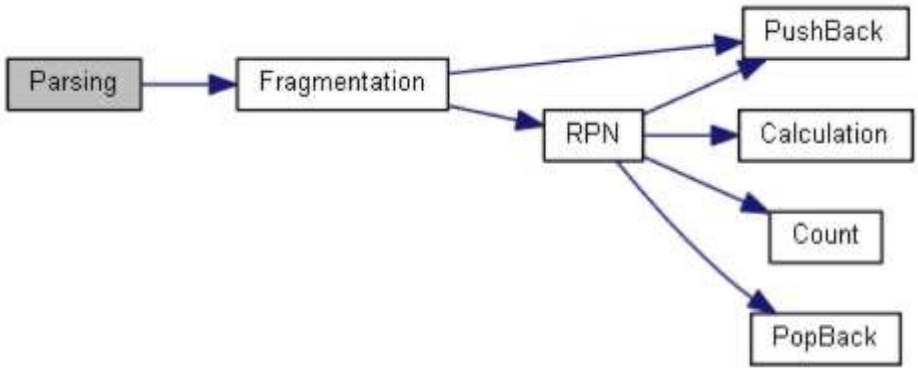
Код функции выглядит следующим образом:

```
string Parsing(int argc, char* argv[])
{
    string expression = "";
    string *variables = new string[argc];
    int countOfVariables = 0;

    for (int i = 0; i < argc; i++)
    {
        string str = argv[i];
        if (str.find("--expression") != string::npos)
        {
            if (i + 1 < argc)
            {
                string str = argv[i + 1];
                if (str != "--variable")
                    expression = argv[i + 1];

            }
        }

        if (str.find("--variable") != string::npos)
        {
            if (i + 1 < argc)
            {
                for (int j = i + 1; j < argc; j++)
                {
                    string variable = argv[j];
                    if (variable == "--expression")
                        break;
                    variables[countOfVariables] = variable;
                    countOfVariables++;
                }

            }
        }
    }
    if (expression.length() == 0)
        return "Error expression";

    return Fragmentation(expression, variables, countOfVariables);
}
```

Authors

Tereshin D.D

Here is the call graph for this function:



Here is the caller graph for this function:

◆ PopBack() [1/2]

```
Sign PopBack ( Sign *  arr,
               int &   i
             )
```

Функция выдающая знак из конца массива и удаляет его

Parameters

arr – массив, из которого нужно взять элемент

i – индекс места, из которого нужно взять элемент

Returns

Sign - последний элемент в массиве

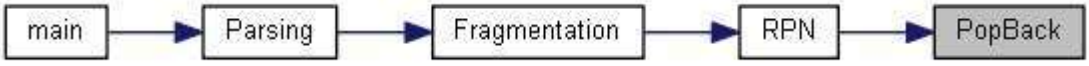
Код функции выглядит следующим образом:

```
Sign PopBack(Sign* arr, int& i)
{
    Sign sign = arr[i - 1];
    arr[i - 1].value = "";
    arr[i - 1].weight = 0;
    i--;
    return sign;
}
```

Authors

Tereshin D.D

Here is the caller graph for this function:



◆ PopBack() [2/2]

```
string PopBack ( string *  arr,
                 int &     i
               )
```

Функция выдающая строку из конца массива и удаляет ее

Parameters

arr – массив, из которого нужно взять элемент

i – индекс места, из которого нужно взять элемент

Returns

string - последний элемент в массиве

Код функции выглядит следующим образом:

```
string PopBack(string* arr, int& i)
{
    string str = arr[i - 1];
    arr[i - 1] = "";
    i--;
    return str;
}
```

Authors

Tereshin D.D

◆ PushBack() [1/2]

```
void PushBack ( Sign *  arr,
                Sign &  sign,
                int &   i
              )
```

Функция кладущая знак в конец массива

Parameters

arr – массив, в который нужно положить элемент

number – знак для операции

i – индекс места, в которое нужно положить элемент

Код функции выглядит следующим образом:

```
void PushBack(Sign* arr, Sign& sign, int& i)
{
    arr[i] = sign;
    i++;
}
```

Authors

Tereshin D.D

◆ PushBack() [2/2]

```
void PushBack ( string *        arr,
                const string &  number,
                int &           i
              )
```

Функция кладущая строку в конец массива

Parameters

arr	– массив, в который нужно положить элемент
number	– строка для операции
i	– индекс места, в которое нужно положить элемент

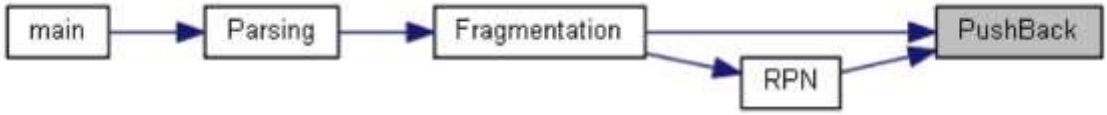
Код функции выглядит следующим образом:

```
void PushBack(string* arr, const string& number, int& i)
{
    if (number.length() != 0)
    {
        arr[i] = number;
        i++;
    }
}
```

Authors

Tereshin D.D

Here is the caller graph for this function:



◆ RPN()

```
string RPN ( string *  tokens,
             int       countOfTokens,
             string *  variables,
             int       countOfVariables
           )
```

Функция выдающая строку из конца массива и удаляет ее

Parameters

tokens – массив токенов

countOfTokens – количество токенов

variables – массив строк переменных

countOfVariables – количество строк переменных

Returns

string - результат всего выражения

Код функции выглядит следующим образом:

```
string RPN(string* tokens, int countOfTokens, string* variables, int countOfVariables)
{
    for (int i = 0; i < countOfVariables; i++)
    {
        int equal = variables[i].find("=");
        string name = variables[i].substr(0, equal);
        string value = variables[i].substr(equal + 1);

        for (int i = 0; i < countOfTokens; i++)
        {
            if (tokens[i] == name)
                tokens[i] = value;
        }
    }

    string* queue = new string[countOfTokens];
    Sign* stackOfSigns = new Sign[countOfTokens];
    int queueCounter = 0, stackCounter = 0;
    bool UnaryMinus = false;
    bool OpenBracket = false;
    for (int i = 0; i < countOfTokens; i++)
    {
        if (tokens[i] == "+")
        {
            Sign plus("+", 2);
            if (Count(stackOfSigns) != 0)
            {
                while (Count(stackOfSigns) != 0 && stackOfSigns[Count(stackOfSigns) -
    1].weight > plus.weight)
                {
                    Sign sign = PopBack(stackOfSigns, stackCounter);
                    PushBack(queue, sign.value, queueCounter);
                }
            }
            PushBack(stackOfSigns, plus, stackCounter);
        }
        else if (tokens[i] == "-")
        {
            Sign minus("-", 2);
            if (Count(stackOfSigns) != 0)
            {
                while (Count(stackOfSigns) != 0 && stackOfSigns[Count(stackOfSigns) -
    1].weight > minus.weight)
                {
                    Sign sign = PopBack(stackOfSigns, stackCounter);
                    PushBack(queue, sign.value, queueCounter);
```

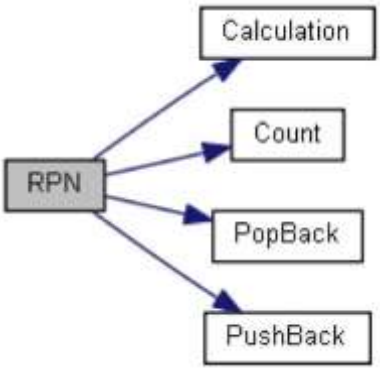
```
                }
            }
            PushBack(stackOfSigns, minus, stackCounter);
            if (OpenBracket == true)
                UnaryMinus = true;
        }

        else if (tokens[i] == "/")
        {
            Sign slash("/", 3);
            if (Count(stackOfSigns) != 0)
            {
                while (Count(stackOfSigns) != 0 && stackOfSigns[Count(stackOfSigns) -
    1].weight > slash.weight)
                {
                    Sign sign = PopBack(stackOfSigns, stackCounter);
                    PushBack(queue, sign.value, queueCounter);
                }
            }
            PushBack(stackOfSigns, slash, stackCounter);
        }

        else if (tokens[i] == "*")
        {
            Sign star("*", 3);
            if (Count(stackOfSigns) != 0)
            {
                while (Count(stackOfSigns) != 0 && stackOfSigns[Count(stackOfSigns) -
    1].weight > star.weight)
                {
                    Sign sign = PopBack(stackOfSigns, stackCounter);
                    PushBack(queue, sign.value, queueCounter);
                }
            }
            PushBack(stackOfSigns, star, stackCounter);
        }

        else if (tokens[i] == "(")
        {
            Sign bracket("(", 1);
            PushBack(stackOfSigns, bracket, stackCounter);
            OpenBracket = true;
        }

        else if (tokens[i] == ")")
        {
            while (stackOfSigns[Count(stackOfSigns) - 1].value != "(")
            {
                if (Count(stackOfSigns) == 0)
                    return "Input Error";
                Sign sign = PopBack(stackOfSigns, stackCounter);
                PushBack(queue, sign.value, queueCounter);
            }
            PopBack(stackOfSigns, stackCounter);
        }

        else
        {
            OpenBracket = false;
            if (UnaryMinus == true)
            {
                tokens[i] = "-" + tokens[i];
                PopBack(stackOfSigns, stackCounter);
            }
            PushBack(queue, tokens[i], queueCounter);
            UnaryMinus = false;
        }
    }
    while (Count(stackOfSigns) != 0)
    {
        Sign sign = PopBack(stackOfSigns, stackCounter);
        if (sign.value == "(")
            return "Input Error";
        PushBack(queue, sign.value, queueCounter);
    }

    string* stackOfnumbers = new string[queueCounter];
    int finalCounter = 0;
    for (int i = 0; i < queueCounter; i++)
    {
```

```
        try
        {
            if (queue[i] == "+")
            {
                string SecondNumber = PopBack(stackOfnumbers, finalCounter);
                string FirstNumber = PopBack(stackOfnumbers, finalCounter);
                PushBack(stackOfnumbers, Calculation(FirstNumber, SecondNumber, "+"), finalCo
            }
            else if (queue[i] == "-")
            {
                string SecondNumber = PopBack(stackOfnumbers, finalCounter);
                string FirstNumber = PopBack(stackOfnumbers, finalCounter);
                PushBack(stackOfnumbers, Calculation(FirstNumber, SecondNumber, "-"), finalCo
            }

            else if (queue[i] == "/")
            {
                string SecondNumber = PopBack(stackOfnumbers, finalCounter);
                string FirstNumber = PopBack(stackOfnumbers, finalCounter);
                PushBack(stackOfnumbers, Calculation(FirstNumber, SecondNumber, "/"), finalCo
            }

            else if (queue[i] == "*")
            {
                string SecondNumber = PopBack(stackOfnumbers, finalCounter);
                string FirstNumber = PopBack(stackOfnumbers, finalCounter);
                PushBack(stackOfnumbers, Calculation(FirstNumber, SecondNumber, "*"), finalCo
            }

            else
            {
                PushBack(stackOfnumbers, queue[i], finalCounter);
            }
        }
        catch (const std::exception&)
        {
            return "Input Error";
        }
    }

    return *stackOfnumbers;
}
```

Authors

Tereshin D.D

Here is the call graph for this function:



Here is the caller graph for this function:

Результаты работы программы.