**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 2.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Бабкин И.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

## Постановка задачи и описание реализуемого класса и методов.

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. Программа должна выполнить проверку корректности введенного выражения. В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода. В противном случае выводится польская нотация введенного выражения, а также отображается результат вычисления.

## Обоснование выбора используемых структур данных.

Был использован список для реализации стэка.

Был использован string для простоты считывания выражения.

## Описание алгоритма программы

Пользователь вводит выражение в инфиксной форме.

Оно просматривается справа налево.

Если просматриваемый символ закрывающая скобка, то помещаем его в стек.

Если просматриваемый символ является оператором или функцией, то удаляем из стека все верхние операторы с приоритетом, большим или равным текущему и помещаем их в выходной потом. Затем помещаем его в стек.

Если просматриваемый символ является открывающей скобкой, то извлекаем из стека все операторы до открывающей скобки и добавляем их в выходной поток.

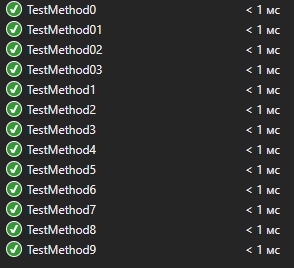
При вычислении выражение просматривается в префиксной форме справа налево

Если текущий элемент является операндом, то помещаем его в стек

Если текущий элемент является оператором, то достаем из стека одно число, если унарная функция, и два числа, если бинарная.

## Описание реализованных unit-тестов.

Для проверки реализованных методов были написаны unit-тесты. В таблице ниже представлены их названия. Названия совпадают с названиями проверяемых методов.



## Пример работы









## Листинг

|  |
| --- |
| LinkedList.h |
| #pragma once  #include <string>  using namespace std;  class LinkedList  {  private:  void add\_first(string newElem);  void reset\_list();  class Node  {  public:  Node(string data, Node\* next = nullptr, Node\* prev = nullptr) {  this->data = data;  this->next = next;  this->prev = prev;  }  ~Node() {}  string data;  Node\* next;  Node\* prev;  };  Node\* head;  Node\* tail;  size\_t size;  public:  LinkedList();  ~LinkedList();  void push\_back(string newElem);  string pop\_back();  string getTop();  size\_t get\_size();  void clear();  bool isEmpty();  }; |

|  |
| --- |
| LinkedList.cpp |
| #include <stdexcept>  #include "iostream"  #include "LinkedList.h"  using namespace std;  void LinkedList::reset\_list()  {  head = nullptr;  tail = nullptr;  }  LinkedList::LinkedList()  {  reset\_list();  size = 0;  }  LinkedList::~LinkedList()  {  }  void LinkedList::add\_first(string newElem)  {  head = new Node(newElem);  tail = head;  }  void LinkedList::push\_back(string newElem)  {  string temp;  temp = newElem;  if (size == 0) {  add\_first(temp);  }  else {  tail->next = new Node(temp, nullptr, tail);  tail = tail->next;  }  size++;  }  string LinkedList::pop\_back()  {  string temp;  if (size == 0) return " ";  if (size == 1) {  temp = head->data;  delete head;  reset\_list();  }  else {  temp = tail->data;  Node\* current = tail->prev;  current->next = nullptr;  delete tail;  tail = current;  }  size--;  return temp;  }  string LinkedList::getTop()  {  if (size == 0) return " ";  return tail->data;  }  size\_t LinkedList::get\_size()  {  return size;  }  void LinkedList::clear()  {  Node\* current = head->next;  while (current->next != nullptr) {  delete head;  head = current;  current = head->next;  }  delete head;  delete tail;  size = 0;  }  bool LinkedList::isEmpty() // if empty - true, else false  {  return !size;  } |

|  |
| --- |
| PolishNotation.h |
| #pragma once  #include <string>  #include "LinkedList.h"  using namespace std;  class PolishNotation {  private:  LinkedList stack;  string space = " ";  public:  string infixString;  string prefixString;  PolishNotation();  ~PolishNotation();  void input();  void infixToPrefix();  int getPriority(string c);  void printToConsole();  double calculation();  }; |

|  |
| --- |
| PolishNotation.cpp |
| #include "PolishNotation.h"  #include <iostream>  #include <cmath>  const double PI = 3.1415926535;  const double E = 2.7182818284;  PolishNotation::PolishNotation()  {  }  PolishNotation::~PolishNotation()  {  }  void PolishNotation::input()  {  getline(cin, infixString);  }  void PolishNotation::infixToPrefix()  {  string temp;  for (int i = infixString.length() - 1; i >= 0; i--) {  if (infixString[i] == '0' || infixString[i] == '1' || infixString[i] == '2' || infixString[i] == '3' || infixString[i] == '4' || infixString[i] == '5' || infixString[i] == '6' || infixString[i] == '7' || infixString[i] == '8'  || infixString[i] == '9' || infixString[i] == '.') {  prefixString = infixString[i] + prefixString;  }  else if (infixString[i] == '+' || infixString[i] == '-' || infixString[i] == '\*' || infixString[i] == '/' || infixString[i] == '^') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back(temp);  prefixString = space + prefixString;  }  else if (infixString[i] == 'n') {  if (infixString[i - 1] == 'i' && infixString[i - 2] == 's') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("sin");  prefixString = space + prefixString;  i -= 2;  }  else if (infixString[i - 1] == 'l') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("ln");  prefixString = space + prefixString;  i--;  }  }  else if (infixString[i] == 's') {  if (infixString[i - 1] == 'o' && infixString[i - 2] == 'c') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("cos");  prefixString = space + prefixString;  i -= 2;  }  }  else if (infixString[i] == 'g') {  if (infixString[i - 1] == 'o' && infixString[i - 2] == 'l') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("log");  prefixString = space + prefixString;  i -= 2;  }  else if (i > 1 && infixString[i - 1] == 't' && infixString[i - 2] == 'c') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("ctg");  prefixString = space + prefixString;  i -= 2;  }  else if (infixString[i - 1] == 't') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("tg");  prefixString = space + prefixString;  i--;  }  }  else if (infixString[i] == 't') {  if (infixString[i - 1] == 'r' && infixString[i - 2] == 'q' && infixString[i - 3] == 's') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("sqrt");  prefixString = space + prefixString;  i -= 3;  }  }  else if (infixString[i] == 'h') {  if (infixString[i - 1] == 's') {  temp = infixString[i];  while (getPriority(temp) <= getPriority(stack.getTop())) {  prefixString = stack.pop\_back() + space + prefixString;  }  stack.push\_back("sh");  prefixString = space + prefixString;  i--;  }  }  else if (infixString[i] == 'e') {  prefixString = infixString[i] + prefixString;  }  else if (infixString[i] == 'i') {  if (infixString[i - 1] == 'p') {  prefixString = "pi" + prefixString;  }  i--;  }  else if (infixString[i] == ')') {  temp = infixString[i];  stack.push\_back(temp);  }  else if (infixString[i] == '(') {  temp = stack.pop\_back();  while (temp != ")") {  prefixString = temp + space + prefixString;  temp = stack.pop\_back();  }  }  else {  cout << "Incorrect input in " << i + 1 << " position";  return;  }  }  while (!stack.isEmpty()) {  prefixString = stack.pop\_back() + space + prefixString;  }  }  int PolishNotation::getPriority(string c)  {  if (c == "cos" || c == "sin" || c == "tg" || c == "ctg" || c == "log" || c == "ln" || c == "sqrt" || c == "sh") return 4;  if (c == "n" || c == "s" || c == "g" || c == "t" || c == "h") return 4;  if (c == "^") return 3;  if (c == "\*" || c == "/") return 2;  if (c == "+" || c == "-") return 1;  if (c == "(" || c == ")") return 0;  return -1;  }  void PolishNotation::printToConsole()  {  cout << prefixString;  }  double PolishNotation::calculation()  {  string temp;  double operand1;  double operand2;  for (int i = prefixString.length() - 1; i >= 0; i--)  {  if (prefixString[i] == '0' || prefixString[i] == '1' || prefixString[i] == '2' || prefixString[i] == '3' || prefixString[i] == '4' || prefixString[i] == '5' || prefixString[i] == '6' || prefixString[i] == '7' || prefixString[i] == '8'  || prefixString[i] == '9' || prefixString[i] == '.') {  string number;  while (prefixString[i] == '0' || prefixString[i] == '1' || prefixString[i] == '2' || prefixString[i] == '3' || prefixString[i] == '4' || prefixString[i] == '5' || prefixString[i] == '6' || prefixString[i] == '7' || prefixString[i] == '8'  || prefixString[i] == '9' || prefixString[i] == '.') {  number = prefixString[i] + number;  i--;  }  i++;  stack.push\_back(number);  }  if (prefixString[i] == '+') {  operand1 = atof(stack.pop\_back().c\_str());  operand2 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(operand1 + operand2));  }  if (prefixString[i] == '-') {  operand1 = atof(stack.pop\_back().c\_str());  operand2 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(operand1 - operand2));  }  if (prefixString[i] == '\*') {  operand1 = atof(stack.pop\_back().c\_str());  operand2 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(operand1 \* operand2));  }  if (prefixString[i] == '/') {  operand1 = atof(stack.pop\_back().c\_str());  operand2 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(operand1 / operand2));  }  if (prefixString[i] == '^') {  operand1 = atof(stack.pop\_back().c\_str());  operand2 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(pow(operand1, operand2)));  }  if (prefixString[i] == 's') {  if (prefixString[i - 1] == 'o' && prefixString[i - 2] == 'c') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(cos(operand1)));  i -= 2;  }  }  if (prefixString[i] == 'n' && i > 1) {  if (prefixString[i - 1] == 'i' && prefixString[i - 2] == 's') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(sin(operand1)));  i -= 2;  }  }  if (prefixString[i] == 'n' && i > 0) {  if (prefixString[i - 1] == 'l') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(log(operand1)));  i--;  }  }  if (prefixString[i] == 'g' && i > 1) {  if (prefixString[i - 1] == 'o' && prefixString[i - 2] == 'l') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(log10(operand1)));  i -= 2;  }  }  if (prefixString[i] == 'g' && i > 0) {  if (prefixString[i - 1] == 't') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(tan(operand1)));  i--;  }  }  if (prefixString[i] == 'g' && i > 1) {  if (prefixString[i - 1] == 't' && prefixString[i - 2] == 'c') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(1 / tan(operand1)));  i -= 2;  }  }  if (prefixString[i] == 't' && i > 2) {  if (prefixString[i - 1] == 'r' && prefixString[i - 2] == 'q' && prefixString[i - 3] == 's') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string(sqrt(operand1)));  i -= 3;  }  }  if (prefixString[i] == 'h' && i > 0) {  if (prefixString[i - 1] == 's') {  operand1 = atof(stack.pop\_back().c\_str());  stack.push\_back(to\_string((pow(E, operand1) - pow(E, -operand1)) / 2));  i--;  }  }  if (prefixString[i] == 'i' && i > 0) {  if (prefixString[i - 1] == 'p') {  temp = to\_string(PI);  stack.push\_back(temp);  i--;  }  }  if (prefixString[i] == 'e') {  temp = to\_string(E);  stack.push\_back(temp);  }  }  double result = atof(stack.pop\_back().c\_str());  cout << endl << result;  return result;  } |