МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**ЛАБОРАТОРНАЯ РАБОТА №11**

**«Арифметика с использованием обратной польской записи»**

**Выполнил:** студент группы 381903-3

Алилуев Артём Олегович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Проверил:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2020

Содержание

[1.Введение 3](#_Toc2955132)

[2. Цели и задачи 6](#_Toc2955133)

[2.1. Используемые инструменты 6](#_Toc2955134)

[3. Руководство пользователя 7](#_Toc2955135)

[4. Руководство программиста 8](#_Toc2955136)

[4.1. Описание структуры программы 8](#_Toc2955137)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc2955138)

[5. Эксперименты 11](#_Toc2955139)

[6. Заключение 12](#_Toc2955140)

[7. Литература 13](#_Toc2955141)

[8. Приложения 14](#_Toc2955142)

[8.1. Приложение 1: Класс TExсeption 14](#_Toc2955143)

[8.2. Приложение 2: Реализация задачи 14](#_Toc2955144)

[8.3. Приложение 3: Класс TStack 16](#_Toc2955145)

[8.4. Приложение 4: Класс TQueue 19](#_Toc2955146)

[8.5. Приложение 5: Код программы тестирования и экспериментов 21](#_Toc2955147)

[8.6. Приложение 6: Тесты 22](#_Toc2955148)

# 1.Введение

Как правило арифметические выражения удобно преобразовывать в обратную польскую запись (ОПЗ), чтобы избавиться от скобок, содержащихся в выражении. Выражения, преобразованные в ОПЗ, можно вычислять последовательно, слева направо.

Преобразование выражения в ОПЗ с использованием стека:

Нам понадобится стек для переменных типа char, т.к. исходное выражение мы получаем в виде строки.  
Рассматриваем поочередно каждый символ:  
1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.  
2. Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.  
Получив один из этих символов, мы должны проверить стек:  
а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.  
б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).  
3. Если текущий символ - открывающая скобка, то помещаем ее в стек.  
4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:  
Дано выражение:  
a + ( b - c ) \* d

Рассмотрим поочередно все символы:

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Действие | Состояние выходной строки после совершенного действия | Состояние стека после совершенного действия |
| a | 'a' - переменная. Помещаем ее в выходную строку | a | пуст |
| + | '+' - знак операции. Помещаем его в стек (поскольку стек пуст, приоритеты можно не проверять) | a | + |
| ( | '(' - открывающая скобка. Помещаем в стек. | a | + ( |
| b | 'b' - переменная. Помещаем ее в выходную строку | a b | + ( |
| - | '-' - знак операции, который имеет приоритет 2. Проверяем стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в стек. | a b | + ( - |
| c | 'c' - переменная. Помещаем ее в выходную строку | a b c | + ( - |
| ) | ')' - закрывающая скобка. Извлекаем из стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки. | a b c - | + |
| \* | '\*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '\*'. Следовательно мы должны просто поместить текущий символ '\*' в стек. | a b c - | + \* |
| d | 'd' - переменная. Помещаем ее в выходную строку | a b c - d | + \* |

Теперь вся входная строка разобрана, но в стеке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку. Поскольку стек - это структура, организованная по принципу LIFO, сначала извлекается символ '\*', затем символ '+'.  
Итак, мы получили конечный результат: a b c - d \* +.

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации вычислений с помощью преобразования арифметических выражений в обратную польскую запись.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация вычислений с помощью преобразования арифметических выражений в обратную польскую запись, с использованием классов TStack и TQueue.
2. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Модификация примера использования в тестовое приложение, позволяющее задавать арифметические выражения с использованием основных операций и получать верный ответ.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

Запускаем программу файла sample\_polish.cpp (Рис.1):

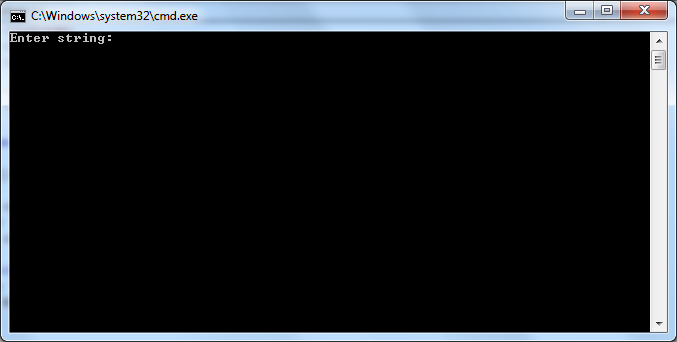


Рис.1.Ввод арифметического выражения.

Пользователю предлагается ввести арифметическое выражение с использованием операций “+”, “-”, “\*”, “/” и скобочек “(”, “)”.

Программа выведет результат (Рис.2).

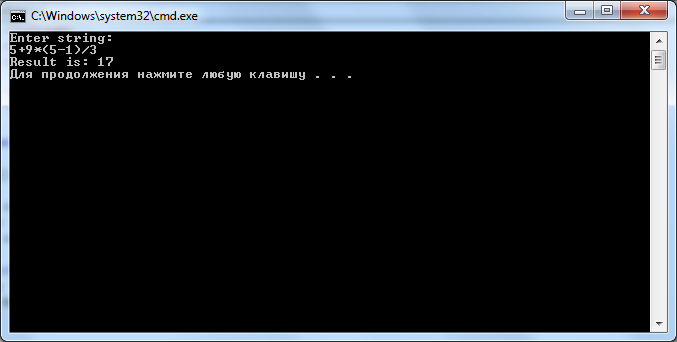


Рис.2.Вывод результата.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано 4 класса:

* Класс «Стек» (TStack), реализованный с использованием массива.
* Класс «Очередь» (TQueue), построенного на основе класса TStack.
* Класс исключения (TExсeption).
* Базовый класс Строки.

А также:

* пространство имен Polish.

## 4.1. Описание структуры программы

Модульная структура программы:

* tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
* exсeption.h – модуль с классом исключения TExсeption.
* tqueue.h– модуль с классом TQueue, в котором определен интерфейс шаблонного класса Очередь и реализация его методов.
* polish.h – модуль с пространством имен Polish.
* polish\_sample.cpp , sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
* test\_main.cpp, test\_polish.cpp – модуль с функциями тестирования для созданной программы. Содержит 9 тестов для функций пространства имен Polish.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TStack:

template <typename StackType>

class TStack

1. TStack(int n = 0) - конструктор класса, принимающий размер стека. По умолчанию создается стек размера 0 с позицией вершины стека 0.

2. TStack(TStack<StackType> &S) - конструктор копирования.

Принимает ссылку на объект класса TStack.

3. ~TStack() - деструктор. Освобождает выделенную под вектор память.

4. TStack& operator=(const TStack<StackType>& stack) – перегрузка оператора присваивания одного стека другому.

Присваивает полям первого объекта класса поля второго объекта класса.

5. int GetSize() - возвращает размер стека.

6. StackType Get() – метод изъятия элемента из вершины стека с удалением.

7. StackType ShowLast() – метод изъятия элемента из вершины стека без удаления.

8. void Put(StackType A) - метод, позволяющий добавить новый элемент в стек.

9. bool IsFull() – метод проверки стека на полноту.

10. bool IsEmpty() – метод проверки стека на пустоту.

11. void PrintStack() - метод отображения текущих элементов стека.

12. bool operator==(const TStack<StackType>& stack) const – перегрузка оператора сравнения.

Выполняется проверка стеков на равенство. Возвращает true, если равенство выполняется, false в противном случае.

13. bool operator!=(const TStack<StackType>& stack) const - перегрузка оператора сравнения.

Выполняется проверка стеков на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию методов шаблонного класса TQueue:

template <typename QueueType>

class TQueue : public TStack <QueueType>

1. TQueue(int n = 0) - конструктор класса с одним параметром.

2. TQueue(TQueue <QueueType> &Q) - конструктор копирования.

Принимает ссылку на объект класса TQueue.

3. TQueue& operator=(const TQueue<QueueType>& queue) - перегрузка оператора присваивания одной очереди другому.

Присваивает полям первого объекта класса поля второго объекта класса.

4. QueueType Get() - метод изъятия элемента из начала очереди с удалением.

5. void Put(QueueType Q) - метод, позволяющий добавить новый элемент в конец очереди.

6. bool IsFull() - метод проверки очереди на полноту.

7. bool IsEmpty() - метод проверки очереди на пустоту.

8. void PrintQueue() - метод отображения текущих элементов очереди.

9. bool operator!=(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на равенство. Возвращает true, если равенство выполняется, false в противном случае.

10. bool operator==(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию функций пространства имен Polish:

namespace Polish

1. inline int GetPriority( const char operation ) – функция, определяющая приоритет записи различных символов.

2. inline int Convert( const char symbol ) – функция, которая переводит символ в число.

3. TQueue<char> ToPolish( string str ) – функция, принимающая строку. Преобразует арифметическое выражение в вид обратной польской записи.

4. double Calculate( TQueue<char> polish ) – функция, принимающая очередь. Производит вычисления и возвращает конечный результат арифметического выражения.

# 5. Эксперименты

В качестве примера рассмотрим функции Calculate( TQueue<char> polish ) и ToPolish( string str ) совместно, для пространства имен (Polish).

Теоретическая сложность выполнения алгоритма O(n).

Мы провели измерение, задавая строку выражения с разным кол-вом символов: 10, 100, …, 1000000 символов. Ниже вы можете увидеть график зависимости времени выполнения вычислений от количества символов в строке. (Грифик.1) По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

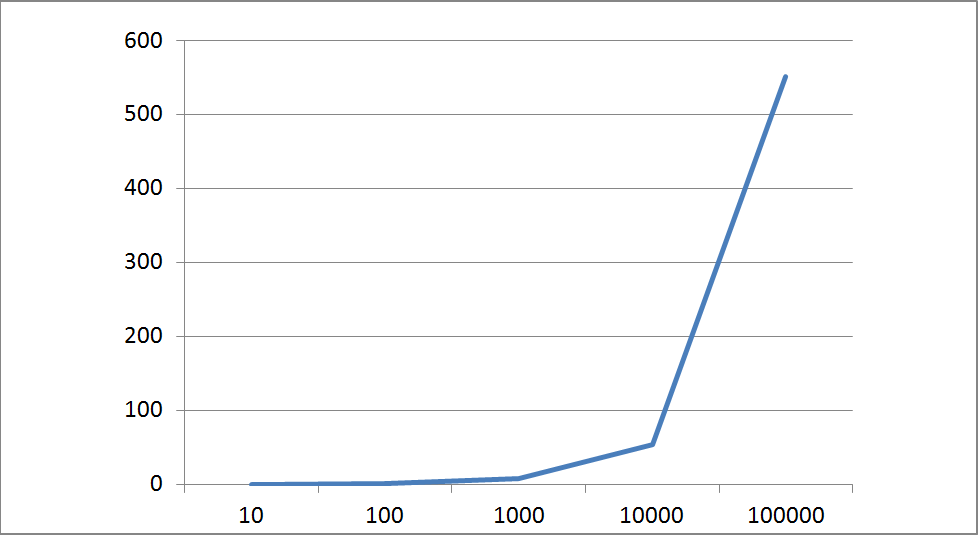


График.1. Зависимость времени выполнения вычислений от кол-ва символов в строке.

По горизонтали – количество символов в строке.

По вертикали - время выполнения программы.

# 6. Заключение

В результате лабораторной работы была разработана реализация преобразования арифметических выражений в обратную польскую запись, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Созданная программа была протестирована с использованием Google Tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на функции пространства имен.

# 7. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Тестирование с использованием Google Test

(<http://neerc.ifmo.ru/wiki/index.php?title=Тестирование_с_использованием_Google_Test#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D1.8F_main.28.29> )

1. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
2. <http://www.interface.ru/home.asp?artid=1492>

# 8. Приложения

## 8.1. Приложение 1: Класс TExсeption

|  |
| --- |
| **exception.h** |
| #pragma once  #include <iostream>  #include <string>  class TException  {  private:  std::string str;  public:  TException(std::string \_str);  void Show();  };  TException::TException(std::string \_str) : str(\_str) {}  void TException::Show()  {  std::cout << "\nWarning! \nMessage: " << str << std::endl;  } |

## 8.2. Приложение 2: Реализация задачи

|  |
| --- |
| **polish.h** |
| #pragma once  #include <iostream>  #include <stdlib.h>  #include <string>  #include <cctype>  #include "tqueue.h"  #include "exception.h"  using namespace std;  namespace Polish {  namespace  {  inline int GetPriority( const char operation )  {  if ( operation == '+' || operation == '-' )  return 3;  if ( operation == '\*' || operation == '/' )  return 4;  if ( operation == ')' )  return 2;  if ( operation == '(' )  return 1;  if ( isdigit(operation) )  return 0;  return -1;  }  inline int Convert( const char symbol )  {  return isdigit(symbol) ? symbol - '0' : -1;  }  }  TQueue<char> ToPolish( string str )  {  TStack<char> operations\_stack(str.size());  TQueue<char> result(str.size());  int priority = 0;  for( string::iterator it = str.begin(); it != str.end(); it++ )  {  priority = GetPriority( \*it );  if ( priority == -1 )  {  throw TException( "Wrong symbol" );  }  else if ( priority == 0 )  {  result.Put(\*it);  }  else if ( priority == 1 )  {  operations\_stack.Put(\*it);  }  else if ( priority == 2 )  {  for(;;)  {  if ( GetPriority(operations\_stack.ShowLast()) == 1 )  {  operations\_stack.Get();  break;  }  else  {  result.Put( operations\_stack.Get() );  }  }  }  else  {  while ( !operations\_stack.IsEmpty() && GetPriority(operations\_stack.ShowLast()) >= priority )  {  result.Put(operations\_stack.Get());  }  operations\_stack.Put(\*it);  }  }  while ( !operations\_stack.IsEmpty() )  {  result.Put( operations\_stack.Get() );  }  return result;  }  double Calculate( TQueue<char> polish )  {  TStack<double> digits\_stack( polish.GetSize() );  int priority = 0;  double var1, var2;  while ( !polish.IsEmpty() )  {  char const element = polish.Get();  priority = GetPriority(element);  if ( priority == -1 || priority == 1 || priority == 2 )  {  throw TException( "Wrong string format" );  }  if ( priority == 0 )  {  digits\_stack.Put( Convert(element) );  }  else  {  var1 = digits\_stack.Get();  var2 = digits\_stack.Get();  switch ( element )  {  case '+':  digits\_stack.Put( var2 + var1 );  break;  case '-':  digits\_stack.Put( var2 - var1 );  break;  case '\*':  digits\_stack.Put( var2 \* var1 );  break;  case '/':  digits\_stack.Put( var2 / var1 );  break;  }  }  }  return digits\_stack.Get();  }  }; |

## 8.3. Приложение 3: Класс TStack

|  |
| --- |
| **tstack.h** |
| #pragma once  #include <iostream>  #include "exception.h"  using namespace std;  template <typename StackType>  class TStack  {  protected:  int size;  int top;  StackType\* memory;  public:  int GetSize() { return size; }  TStack(int n = 0);  TStack(TStack<StackType> &S);  TStack& operator=(const TStack<StackType>& stack);  ~TStack() { delete[] memory; }  StackType Get();  StackType ShowLast();  void Put(StackType A);  bool IsFull();  bool IsEmpty();  void PrintStack();  bool operator!=(const TStack<StackType>& stack) const;  bool operator==(const TStack<StackType>& stack) const;  };  //----------------------------------------------------------------------  template <typename StackType>  TStack <StackType> ::TStack(int n)  {  if ( n < 0 )  throw TException(" Negative size.");  else if ( n == 0 )  {  size = 0;  top = 0;  memory = NULL;  }  else  {  size = n;  top = 0;  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = 0;  }  }  //----------------------------------------------------------------------  template <typename StackType>  TStack <StackType> ::TStack(TStack <StackType> &S)  {  size = S.size;  top = S.top;  if ( size == 0 )  memory = NULL;  else  {  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = S.memory[i];  }  }  //----------------------------------------------------------------------  template <typename StackType>  TStack<StackType>& TStack<StackType> ::operator=(const TStack<StackType>& stack)  {  if ( this != &stack )  {  top = stack.top;  if ( size != stack.size )  {  size = stack.size;  delete[] memory;  memory = new StackType[size];  }  for ( int i = 0; i < size; i++ )  {  memory[i] = stack.memory[i];  }  }  return \*this;  }  //----------------------------------------------------------------------  template <typename StackType>  void TStack<StackType> ::Put(StackType A)  {  if ( IsFull() )  {  throw TException("Stack is full");  }  else  {  memory[top] = A;  top++;  }  }  //----------------------------------------------------------------------  template <typename StackType>  StackType TStack<StackType> ::Get()  {  if ( IsEmpty() )  throw TException("Stack is empty");  else  {  top--;  return memory[top];  }  }  //----------------------------------------------------------------------  template <typename StackType>  StackType TStack<StackType> ::ShowLast()  {  if ( IsEmpty() )  throw TException( "Stack is empty" );  else  {  return memory[top - 1];  }  }  //----------------------------------------------------------------------  template <typename StackType>  bool TStack<StackType> ::IsFull()  {  //ternary operator  return top >= size ? true : false;  }  //----------------------------------------------------------------------  template <typename StackType>  bool TStack<StackType> ::IsEmpty()  {  return !top;  }  //----------------------------------------------------------------------  template <typename StackType>  bool TStack<StackType> ::operator==(const TStack<StackType>& stack) const  {  if ( top != stack.top || size != stack.size )  return false;  for ( int i = 0; i < top; i++ )  {  if ( memory[i] != stack.memory[i] )  return false;  }  return true;  }  //----------------------------------------------------------------------  template <typename StackType>  bool TStack<StackType> ::operator!=(const TStack<StackType>& stack) const  {  return !( \*this == stack );  }  //----------------------------------------------------------------------  template <typename StackType>  void TStack<StackType>:: PrintStack()  {  for ( int i = top-1; i >= 0; i-- )  cout << "\t|" << memory[i] << "|" << endl;  } |

## 8.4. Приложение 4: Класс TQueue

|  |
| --- |
| **tqueue.h** |
| #pragma once  #include "tstack.h"  template <typename QueueType>  class TQueue : public TStack <QueueType>  {  protected:  int start;  int count;  public:  TQueue(int n = 0);  TQueue(TQueue <QueueType> &Q);  TQueue& operator=(const TQueue<QueueType>& queue);  QueueType Get();  void Put(QueueType Q);  bool IsFull();  bool IsEmpty();  void PrintQueue();  bool operator!=(const TQueue<QueueType>& queue) const;  bool operator==(const TQueue<QueueType>& queue) const;  };  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>::TQueue(int n) : TStack<QueueType>(n)  {  start = 0;  count = 0;  }  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>::TQueue(TQueue<QueueType> &Q) : TStack<QueueType>(Q)  {  start = Q.start;  count = Q.count;  }  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>& TQueue<QueueType> ::operator=(const TQueue<QueueType>& queue)  {  if ( this == &queue ) return \*this;  start = queue.start;  count = queue.count;  TStack<QueueType>::top = queue.top;  if ( TStack<QueueType>::size != queue.size )  {  TStack<QueueType>::size = queue.size;  delete[] TStack<QueueType>::memory;  TStack<QueueType>::memory = new QueueType[TStack<QueueType>::size];  }  for ( int i = start; i <count+start; i++ )  {  TStack<QueueType>::memory[i % TStack<QueueType>::size] = queue.memory[i % TStack<QueueType>::size];  }  return \*this;  }  //-----------------------------------------------------------------  template <typename QueueType>  void TQueue<QueueType>::Put(QueueType el)  {  if (IsFull())  throw TException("Queue is full.");  else  {  TStack<QueueType>::memory[TStack<QueueType>::top] = el;  TStack<QueueType>::top = ++TStack<QueueType>::top % TStack<QueueType>::size;  count++;  }  }  //-----------------------------------------------------------------  template <typename QueueType>  QueueType TQueue<QueueType>::Get()  {  if (IsEmpty())  throw TException("Queue is empty.");  else  {  QueueType buf = TStack<QueueType>::memory[start];  start = ++start % TStack<QueueType>::size;  count--;  return buf;  }  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType>::IsFull()  {  return count == TStack<QueueType>::size ? true: false;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType>::IsEmpty()  {  return count == 0 ? true: false;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType> ::operator==(const TQueue<QueueType>& queue) const  {  if ( TStack<QueueType>::top != queue.top || TStack<QueueType>::size != queue.size ||  count != queue.count || start !=queue.start )  return false;  for ( int i = start; i < count + start; i++ )  {  if ( TStack<QueueType>::memory[i % TStack<QueueType>::size] != queue.memory[i % TStack<QueueType>::size] )  return false;  }  return true;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType> ::operator!=(const TQueue<QueueType>& queue) const  {  return !( \*this == queue );  }  template <typename QueueType>  void TQueue<QueueType>:: PrintQueue()  {  for ( int i = start; i <count+start; i++ )  {  cout<<TStack<QueueType>::memory[i % TStack<QueueType>::size]<<" | ";  }  cout << endl;  } |

## 8.5. Приложение 5: Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_polish.cpp** |
| #include <iostream>  #include <locale.h>  #include <string>  #include "polish.h"  #include "tqueue.h"  using namespace std;  int main()  {  string calc\_exp;  cout << "Enter string: " << endl;  cin >> calc\_exp;  TQueue<char> polish = Polish::ToPolish(calc\_exp);  cout << "Result is: " << Polish::Calculate(polish) << endl;  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include <string>  #include "polish.h"  #include "time.h"  using namespace std;  int main()  {  int max\_count = 0;  cout << "Enter number of checks - ";  cin >> max\_count;  clock\_t time;  clock\_t average\_time;  for ( unsigned size = 10; size < 1000000; size \*= 10 )  {  string expr = "1";  for ( int i = 1; i < size; i++ )  {  expr.push\_back('+');  expr.push\_back('1');  }  average\_time = 0;  for ( unsigned count = 0; count < max\_count; count++ )  {  time = clock();  Polish::Calculate(Polish::ToPolish(expr));  average\_time += clock() - time;  }  average\_time /= max\_count;  cout<< "Time is: " <<average\_time<<endl;  }  return 0;  } |

## 8.6. Приложение 6: Тесты

|  |
| --- |
| **test\_main.cpp** |
| #include <gtest.h>  int main(int argc, char \*\*argv)  {  ::testing::InitGoogleTest(&argc, argv);  return RUN\_ALL\_TESTS();  } |

|  |
| --- |
| **test\_polish.cpp** |
| #include "polish.h"  #include <string>  #include <gtest.h>  TEST(Polish, can\_get\_priority)  {  EXPECT\_EQ(Polish::GetPriority('+'), 3);  EXPECT\_EQ(Polish::GetPriority('-'), 3);  EXPECT\_EQ(Polish::GetPriority('\*'), 4);  EXPECT\_EQ(Polish::GetPriority('/'), 4);  EXPECT\_EQ(Polish::GetPriority(')'), 2);  EXPECT\_EQ(Polish::GetPriority('('), 1);  EXPECT\_EQ(Polish::GetPriority('y'), -1);  EXPECT\_EQ(Polish::GetPriority('8'), 0);  }  TEST(Polish, can\_convert)  {  EXPECT\_EQ(Polish::Convert('5'), 5);  EXPECT\_EQ(Polish::GetPriority('u'), -1);  }  TEST(Polish, can\_use\_plus)  {  double rez;  string exp = "1+1";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, 2);  }  TEST(Polish, can\_use\_minus)  {  double rez;  string exp = "1-5";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, -4);  }  TEST(Polish, can\_use\_division)  {  double rez;  string exp = "8/4";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, 2);  }  TEST(Polish, can\_get\_double\_number)  {  double rez;  string exp = "9/2";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, 4.5);  }  TEST(Polish, can\_use\_multiplication)  {  double rez;  string exp = "5\*4";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, 20);  }  TEST(Polish, can\_use\_brackets)  {  double rez;  string exp = "5\*(1+5)-9";  TQueue<char> polish = Polish::ToPolish(exp);  rez = Polish::Calculate(polish);  EXPECT\_EQ(rez, 21);  }  TEST(Polish, thow\_when\_use\_wrong\_symbol)  {  string exp = "1y+1";  ASSERT\_ANY\_THROW(TQueue<char> polish = Polish::ToPolish(exp));  } |