Практика #3.

АТД - Стек.

Вычисление арифметических выражений

Рассмотрено на лекции:

АТД стек – Stack – абстрактный класс

СД стек «ограниченный» - StackArray – реализуется через массив

СД стек «неограниченный» - StackList – реализуется через список (односвязный)

1. Реализуйте структуру данных «ограниченный» стек:

1. Используйте шаблон для абстрактного типа данных класса стек:

```
template <class T>
class Stack
{
public:
    virtual ~Stack() {}
    virtual void push(const T& e) = 0; // Добавление элемента в стек
    virtual T  pop() = 0;       // Удаление и возвращение верхнего элемента
    virtual bool isEmpty() = 0; // Проверка стека на пустоту
};
```

2. На основе шаблона для стека создайте шаблон для реализации структуры данных «ограниченный» стек (через массив).
3. Создайте классы StackOverflow и StackUnderflow для работы с двумя исключительными ситуациями, которые могут возникнуть при работе со стеком.
4. Создайте класс WrongStackSize для работы с исключительной ситуацией, которая может возникнуть, если в конструкторе стека, реализуемого через массив, неправильно задан размер.

2. Реализуйте перечисленные ниже функции, используйте шаблон «ограниченный» стек.

1. Функция анализа правильности расстановки скобок

   Функция должна возвращать True, если количество открывающих и закрывающих скобок одного типа совпадает, и они имеют правильную вложенность. Допускаются три вида скобок: круглые, квадратные и фигурные.
   Прототип функции:
   `bool checkBalanceBrackets (const char* text, const int maxDeep);`
   или
   `bool checkBalanceBrackets (const string& text, const int maxDeep);`
   `text` - анализируемый текст, содержащий скобки
   `maxDeep` - максимально возможный уровень вложенности скобок

2. Функция перевода арифметического выражения из инфиксной формы в постфиксную.

   Функция должна формировать выражение в постфиксной форме и генерировать исключение, если возникает ошибка (например, при переполнении стека…)
   Считать, что выражение состоит из операндов (одноразрядных чисел), знаков бинарных операций (+, -, *, /) и круглых скобок.
   Перевод выполнить за один проход по выражению, в процессе выполнения которого проверить и правильность скобок.
   Прототип функции:

```
bool getPostfixFromInfix (const char* infix, char* postfix,
                          const size_t stackSize);
```

   или

```
bool getPostfixFromInfix (const string& infix, string& postfix,
                          const size_t stackSize);
```

   `infix` - выражение в инфиксной форме
   `postfix` - выражение в постфиксной форме
   `stackSize` - максимальный размер стека операций

3. Функция вычисления значения арифметического выражения в постфиксной форме

   Функция должна вычислять значение выражения в постфиксной форме или генерировать исключение, если возникают ошибки, например, в инфиксном выражении, при переполнении стека…
   Считать, что выражение состоит из операндов (одноразрядных чисел), знаков бинарных операций (+, -, *, /). Деление – целочисленное.
   Прототип функции:

```
int evaluatePostfix  (const char* infix, const size_t stackSize);
```

   или

```
int evaluatePostfix (const string& infix, const size_t stackSize);
```

   `postfix` - выражение в постфиксной форме
   `stackSize` - максимальный размер стека операций