**Лабораторная работа № 2**

1. **Общая постановка задачи**

1. Реализуйте структуру данных «ограниченный» стек

1. Используйте шаблон для абстрактного типа данных класса стек:

*template class Stack { public: virtual ~Stack() {} virtual void push(const T& e) = 0; // Добавление элемента в стек*

*virtual T pop() = 0; // Удаление 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.

5. Выполните тестирование созданного класса.

2. Реализуйте функцию 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 В реализации используйте шаблон «ограниченный стек».

3. Реализуйте структуру данных «ограниченная» очередь

1. Используйте шаблон для абстрактного типа данных класса очередь:

*template class Queue { public: virtual ~Queue () {} virtual void enQueue(const T& e) = 0; // Добавление элемента в очередь*

*virtual T deQueue() = 0; // Удаление элемента из очереди*

*virtual bool isEmpty() = 0; // Проверка очереди на пустоту };*

2. На основе шаблона для очереди создайте шаблон для реализации структуры данных «ограниченная» очередь (через массив).

3. Создайте классы QueueOverflow и QueueUnderflow для работы с двумя исключительными ситуациями, которые могут возникнуть при работе с очередью.

4. Создайте класс WrongQueueSize для работы с исключительной ситуацией, которая может возникнуть, если в конструкторе очереди, реализуемой через массив, неправильно задан размер.

5. Выполните тестирование созданного класса

**Детальные требования, тест план**

1. **Таблица с детальными требованиями и тест планов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Требование** | **Детальные требования** | **Данные** | **Ожидаемый результат** |
| 1. Тест CheckBalanceBrackets() | | | |
| Если баланс скобок соблюден, выводится сообщение «1», иначе выводится сообщение «0» | bool checkBalanceBrackets(const char\* text, const int maxDeep)  {  char cur = text[0];  StackArray<char> stack;  int deep = 0;  int index = 0;  char popped = text[0];  while (cur != '\0' && maxDeep >= deep)  {    if (cur == '(' || cur == '{' || cur == '[')  {  stack.push(cur);  deep++;  }  if ((cur == ')' || cur == '}' || cur == ']') )  {  if (stack.isEmpty())  return false;  popped = stack.pop();  if (popped == '(' && cur == ')' || popped == '{' && cur == '}' || popped == '[' && cur == ']')  {  deep--;  }  else  return false;  }  index++;  cur = text[index];  }    return !deep;  }  void test\_CheckBalanceBrackets()  {  std::cout << "------------ТЕСТЫ CheckBalanceBrackets------------\n";  std::cout << "1:" << checkBalanceBrackets("((()))", 3) << std::endl;  std::cout << "2:" << checkBalanceBrackets("()))", 3) << std::endl;  std::cout << "3:" << checkBalanceBrackets("((()))", 2) << std::endl;  std::cout << "4:" << checkBalanceBrackets(" ", 3) << std::endl;  std::cout << "5:" << checkBalanceBrackets("g", 3) << std::endl;  std::cout << "6:" << checkBalanceBrackets("(", 3) << std::endl;  std::cout << "7:" << checkBalanceBrackets(")", 3) << std::endl;  std::cout << "8:" << checkBalanceBrackets(" ", 0) << std::endl;  std::cout << "9:" << checkBalanceBrackets("(", 0) << std::endl;  std::cout << "10:" << checkBalanceBrackets("((", 0) << std::endl;  std::cout << "11:" << checkBalanceBrackets("(3+1)\*2", 1) << std::endl;  std::cout << "12:" << checkBalanceBrackets("()()()()()()", 10) << std::endl;  std::cout << "13:" << checkBalanceBrackets("{}[]()", 3) << std::endl;  std::cout << "14:" << checkBalanceBrackets("({[]})", 3) << std::endl;  std::cout << "15:" << checkBalanceBrackets("{}[]()", 2) << std::endl;  std::cout << "16:" << checkBalanceBrackets("({[]})", 2) << std::endl;  std::cout << "17:" << checkBalanceBrackets("{[", 3) << std::endl;  std::cout << "18:" << checkBalanceBrackets("{(]", 3) << std::endl;  } |  | ------------ТЕСТЫ CheckBalanceBrackets------------  1:1  2:0  3:0  4:1  5:1  6:0  7:0  8:1  9:0  10:0  11:1  12:1  13:1  14:1  15:1  16:0  17:0  18:0 |
| 1. Тесты СТЕКА | | | |
| Если стек пустой, выводится сообщение «1», иначе выводится сообщение «0».  Если стек переполнен, выводится сообщение «Stack Overflow». Если стек пустой, выводится сообщение «Stack Overflow». При удалении элемента из стека выводится значение удаленного элемента. | void test\_Stack()  {  std::cout << "\n------------ТЕСТЫ Stack------------\n";  StackArray<int> stack1(4);  std::cout << stack1.isEmpty() << std::endl;  try  {  std::cout << stack1.pop() << std::endl;  std::cout << stack1.pop() << std::endl;  }  catch (StackUnderflow& e)  {  std::cout << e.what() << std::endl;  }  stack1.push(1);  stack1.push(2);  stack1.push(3);  stack1.push(4);  try  {  stack1.push(5);  }  catch (StackOverflow& e)  {  std::cout << e.what() << std::endl;  }  std::cout << stack1.isEmpty() << std::endl;  std::cout << stack1.pop() << std::endl;  std::cout << stack1.pop() << std::endl;  std::cout << stack1.pop() << std::endl;  std::cout << stack1.pop() << std::endl;  } |  | ------------ТЕСТЫ Stack------------  1  Stack Underflow  Stack Overflow  0  4  3  2  1 |
| 1. Тесты ОЧЕРЕДИ | | | |
| Если очередь пустая, выводится сообщение «1», иначе выводится сообщение «0».  Если очередь переполнена, выводится сообщение «Queue Overflow». Если очередь пустая, выводится сообщение «Queue Overflow». При удалении элемента из очереди выводится значение удаленного элемента. | void test\_Queue()  {  std::cout << "\n------------ТЕСТЫ Queue------------\n";  QueueArray<int> queue1(4);  std::cout << queue1.isEmpty() << std::endl;  try  {  queue1.deQueue();  }  catch (QueueUnderflow& e)  {  std::cout << e.what() << std::endl;  }  queue1.enQueue(1);  queue1.enQueue(2);  queue1.enQueue(3);  queue1.enQueue(4);  try  {  queue1.enQueue(5);  }  catch (QueueOverflow& e)  {  std::cout << e.what() << std::endl;  }  std::cout << queue1.isEmpty() << std::endl;  std::cout << queue1.deQueue() << std::endl;  std::cout << queue1.deQueue() << std::endl;  queue1.enQueue(5);  std::cout << queue1.deQueue() << std::endl;  std::cout << queue1.deQueue() << std::endl;  queue1.enQueue(6);  queue1.enQueue(7);  queue1.enQueue(8);  std::cout << queue1.deQueue() << std::endl;  std::cout << queue1.deQueue() << std::endl;  std::cout << queue1.deQueue() << std::endl;  std::cout << queue1.deQueue() << std::endl;  QueueArray<const char\*> queue2(2);  queue2.enQueue("sca");  queue2.enQueue("safahajsckb");  std::cout << queue2.deQueue() << std::endl;  std::cout << queue2.deQueue() << std::endl;  std::cout << queue2.isEmpty() << std::endl;  } |  | ------------ТЕСТЫ Queue------------  1  Queue Underflow  Queue Overflow  0  1  2  3  4  5  6  7  8  sca  safahajsckb  1 |