# Лабораторная работа № 3

Выполнил: Лосев Данил

**Задание:**

1. Реализовать коды в виде функций в удобном для Вас редакторе кода на языке программирования С++:

* Построение стека
* Вывод стека на экран

1. На основе алгоритмов из лекции № 3, созданных с помощью схем Дональда Кнута, написать соответствующие коды в виде функций с комментариями и откомпилировать их:

* вставка звена в стек
* удаление звена из стека

3.  Самостоятельно реализовать функции:

* поиск адреса звена по заданному значению
* нахождения количества элементов в стеке
* нахождения максимального элемента в стеке
* очистка памяти

|  |
| --- |
| #include <cstddef>  #include <iostream>  // Определяю структуру Node, представляющую элемент стека.  // В каждом узле хранится значение и указатель на следующий узел.  struct Node  {  int value;  Node \*next;  };  // Прототипы функций для работы со стеком. Здесь я создаю функции для добавления, удаления и поиска элементов в стеке.  void fillUserStack(Node \*&fStack);  void deleteUserStack(Node \*&fStack);  void push(Node \*&fStack, int fValue);  void pop(Node \*&fStack);  void printStack(Node \*fStack);  int findValueOfStack(Node \*fStack, int fValue);  int countOfNodesInStack(Node \*fStack);  int maxValueOfStack(Node \*fStack);  int main()  {  int value;  Node \*stack = NULL; // Инициализирую пустой стек.  // Заполняю стек элементами, введенными пользователем.  fillUserStack(stack);  printStack(stack); // Вывожу текущее состояние стека.  std::cout << "\nAdding element to the stack: ";  std::cin >> value;  push(stack, value); // Добавляю элемент в стек.  printStack(stack); // Снова вывожу стек, чтобы показать обновление.  std::cout << "\nRemoving an element from a stack" << std::endl;  pop(stack); // Удаляю элемент из стека.  printStack(stack); // Вывожу стек после удаления.  // Определяю и вывожу количество элементов в стеке.  int countOfNodes = countOfNodesInStack(stack);  std::cout << "\nNumber of elements in the stack: " << countOfNodes << std::endl;  // Ищу элемент в стеке по значению, введенному пользователем.  std::cout << "\nWhat element do you want to find: ";  std::cin >> value;  int countOfFoundElement = findValueOfStack(stack, value);  std::cout << "Count of found element: " << countOfFoundElement << std::endl;  // Нахожу максимальный элемент в стеке и вывожу его.  int MaxValue = maxValueOfStack(stack);  std::cout << "\nMax element of stack: " << MaxValue << std::endl;  // Очищаю стек и освобождаю память.  deleteUserStack(stack);  printStack(stack);  delete stack;  return 0;  }  // Функция запрашивает у пользователя значения для заполнения стека.  // Введенные значения добавляются в стек до тех пор, пока пользователь не введет 0.  void fillUserStack(Node \*&fStack)  {  std::cout << "\*\*Building a stack: \*\*" << std::endl;  int value;  std::cout << "Enter the values of the information fields of the stack nodes (0 = end):" << '\n' << "> ";  std::cin >> value;  while (value != 0)  {  push(fStack, value); // Добавляю введенное значение в стек.  std::cout << "> ";  std::cin >> value;  }  std::cout << "Stack built" << std::endl;  }  // Очищаю весь стек, последовательно удаляя каждый элемент.  void deleteUserStack(Node \*&fStack)  {  std::cout << "\n\*\*Stack removal\*\*" << std::endl;  while (fStack != NULL)  {  pop(fStack); // Удаляю элементы стека до тех пор, пока он не станет пустым.  }  }  // Добавляю новый элемент в стек, который будет находиться на его вершине.  void push(Node \*&fStack, int fValue)  {  Node \*curNode = new (Node); // Выделяю память для нового элемента.  (\*curNode).value = fValue; // Присваиваю ему значение.  (\*curNode).next = fStack; // Указываю, что следующий элемент — это текущий верхний элемент стека.  fStack = curNode; // Новым верхним элементом становится добавленный.  }  // Удаляю элемент с вершины стека.  void pop(Node \*&fStack)  {  if (fStack != NULL)  {  Node \*curNode = fStack; // Сохраняю текущий верхний элемент.  fStack = (\*fStack).next; // Смещаю указатель на следующий элемент.  delete curNode; // Удаляю старый верхний элемент.  }  }  // Вывожу все элементы стека, начиная с вершины.  void printStack(Node \*fStack)  {  if (fStack == NULL)  {  std::cout << "Stack is empty" << std::endl; // Если стек пуст, сообщаю об этом.  }  else  {  Node \*curNode = fStack;  std::cout << "Stack: ";  while (curNode != NULL)  {  std::cout << '\t' << (\*curNode).value; // Вывожу значение текущего узла.  curNode = (\*curNode).next; // Перехожу к следующему узлу.  }  std::cout << std::endl;  }  }  // Ищу, сколько раз заданное значение встречается в стеке.  int findValueOfStack(Node \*fStack, int fValue)  {  int count = 0;  Node \*curNode = fStack;  while (curNode != NULL)  {  if ((\*curNode).value == fValue)  {  count++; // Если нахожу совпадение, увеличиваю счетчик.  }  curNode = (\*curNode).next;  }  return count;  }  // Подсчитываю количество узлов в стеке.  int countOfNodesInStack(Node \*fStack)  {  int count = 0;  Node \*curNode = fStack;  while (curNode != NULL)  {  count++; // Увеличиваю счетчик для каждого узла.  curNode = (\*curNode).next;  }  return count;  }  // Нахожу максимальный элемент стека.  int maxValueOfStack(Node \*fStack)  {  Node \*curNode = fStack;  int maxValue = (\*curNode).value; // Начинаю с значения вершины стека.  while (curNode != NULL)  {  if ((\*curNode).value > maxValue)  {  maxValue = (\*curNode).value; // Обновляю максимальное значение, если нахожу большее.  }  curNode = (\*curNode).next;  }  return maxValue;  } |

