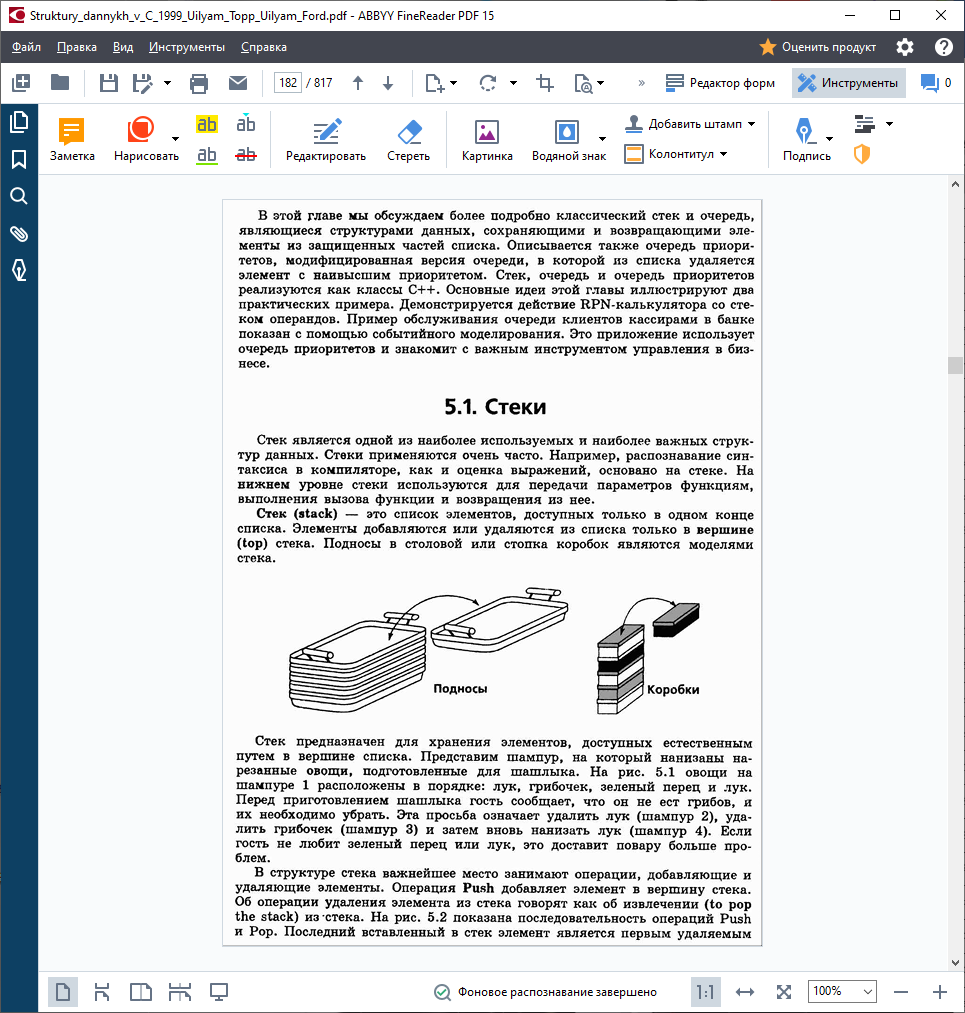
Стек является одной из наиболее используемых и наиболее важных струк­ тур данных. Стеки применяются очень часто. Например, распознавание син­ таксиса в компиляторе, как и оценка выражений, основано на стеке. На нижнем уровне стеки используются для передачи параметров функциям, выполнения вызова функции и возвращения из нее.

Стек (stack) — это список элементов, доступных только в одном конце списка. Элементы добавляются или удаляются из списка только в вершине (top) стека. Подносы в столовой или стопка коробок являются моделями стека.

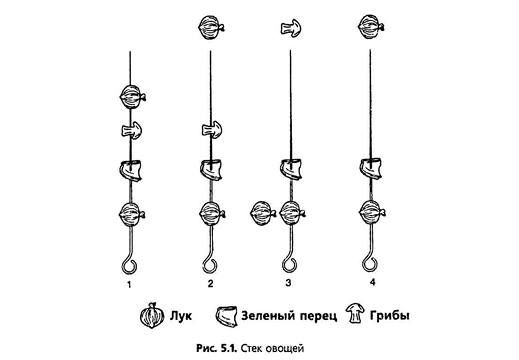


Стек предназначен для хранения элементов, доступных естественным путем в вершине списка. Представим шампур, на который нанизаны на­ резанные овощи, подготовленные для шашлыка. На рис. 5.1 овощи на шампуре 1 расположены в порядке: лук, грибочек, зеленый перец и лук.

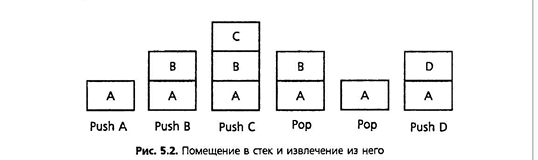
Перед приготовлением шашлыка гость сообщает, что он не ест грибов, и их необходимо убрать. Эта просьба означает удалить лук (шампур 2), уда­ лить грибочек (шампур 3) и затем вновь нанизать лук (шампур 4). Если гость не любит зеленый перец или лук, это доставит повару больше про­ блем.

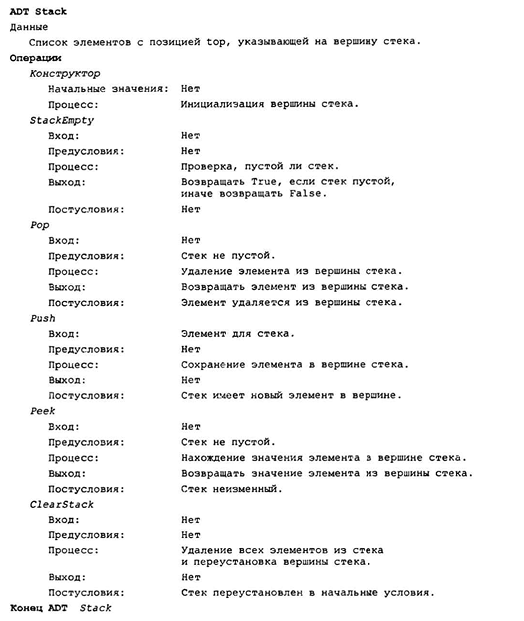
В структуре стека важнейшее место занимают операции, добавляющие и удаляющие элементы. Операция Push добавляет элемент в вершину стека.

Об операции удаления элемента из стека говорят как об извлечении (to pop the stack) из стека. На рис. 5.2 показана последовательность операций Push и Pop. Последний вставленный в стек элемент является первым удаляемым

элементом. По этой причине о стеке говорят, что он имеет порядок LIFO (last-in/first-out) (последний пришел/первый ушел). 

Абстрактное понятие стека допускает неопределенно большой список. Ло­ гически подносы в столовой могут складываться бесконечно. В действитель­ ности подносы находятся на полке, а овощи нанизаны на коротких шампурах.

Когда полка или шампур переполнены, мы не можем добавить (Push) еще один элемент в стек. Стек достигает максимального количества элементов, которыми он может управлять. Эта ситуация поясняет значение условия полного стека (stack full). Другая крайность — вы не можете взять поднос с пустой полки. Условие пустого стека (stack empty) подразумевает, что вы не можете удалить (Pop) элемент. Описание ADT Stack включает только условие пустого стека. Условие полного стека является уместным в том слу­ чае, если реализация содержит верхнюю границу размера списка. 



Мой класс стэка на c#:

public class Stack<T> // Объявляем класс стэка с использованием обобщений

{

private T[] items; // Массив для хранения элементов стэка

private int top; // Индекс верхнего элемента стэка

private int count; // Количество элементов в стеке

public Stack(int size=10) // Конструктор, который принимает размер стэка

{

items = new T[size]; // Инициализируем массив заданным размером

top = -1; // Устанавливаем индекс верхнего элемента в -1, чтобы указать на пустой стэк

}

public void Push(T item) // Метод для добавления элемента в стэк

{

if (top == items.Length - 1) // Проверяем, полон ли стэк

{

throw new Exception("Stack overflow"); // Если стэк полон, выбрасываем исключение

}

top++; // Увеличиваем индекс верхнего элемента на 1

items[top] = item; // Добавляем элемент на вершину стэка

count++; // Увеличиваем количество элементов на 1

}

public T Pop() // Метод для удаления и возврата верхнего элемента стэка

{

if (top == -1) // Проверяем, пуст ли стэк

{

throw new Exception("Stack underflow"); // Если стэк пуст, выбрасываем исключение

}

T item = items[top]; // Сохраняем верхний элемент стэка в переменную

top--; // Уменьшаем индекс верхнего элемента на 1

count--; // Уменьшаем количество элементов на 1

return item; // Возвращаем удаленный элемент

}

public T Peek() // Метод для просмотра верхнего элемента стэка без удаления

{

if (top == -1) // Проверяем, пуст ли стэк

{

throw new Exception("Stack is empty"); // Если стэк пуст, выбрасываем исключение

}

return items[top]; // Возвращаем верхний элемент стэка

}

public bool IsEmpty() // Метод для проверки, пуст ли стэк

{

return (top == -1); // Возвращаем true, если индекс верхнего элемента равен -1 (то есть, стэк пуст)

}

public bool IsFull() // Метод для проверки, полон ли стэк

{

return (top == items.Length - 1); // Возвращаем true, если индекс верхнего элемента равен длине массива - 1 (то есть, стэк полон)

}

public int Count() // Метод для получения количества элементов в стеке

{

return count; // Возвращаем текущее количество элементов

}

}

Постфиксный калькулятор на основе класса Стэк:

using System;

using System.Collections.Generic;

public class PostfixCalculator

{

public static double Calculate(string expression)

{

Stack<double> stack = new Stack<double>(); // Создаем экземпляр класса Stack для хранения чисел

foreach (string token in expression.Split(' ')) // Разбиваем выражение на токены и перебираем их

{

if (double.TryParse(token, out double number)) // Если токен является числом

{

stack.Push(number); // Добавляем число на вершину стэка

}

else // Иначе, токен является оператором

{

double operand2 = stack.Pop(); // Извлекаем из стэка второй операнд

double operand1 = stack.Pop(); // Извлекаем из стэка первый операнд

switch (token) // Выполняем операцию в зависимости от оператора

{

case "+":

stack.Push(operand1 + operand2); // Добавляем результат операции на вершину стэка

break;

case "-":

stack.Push(operand1 - operand2); // Добавляем результат операции на вершину стэка

break;

case "\*":

stack.Push(operand1 \* operand2); // Добавляем результат операции на вершину стэка

break;

case "/":

stack.Push(operand1 / operand2); // Добавляем результат операции на вершину стэка

break;

default:

throw new ArgumentException($"Invalid operator: {token}"); // Если оператор недопустим, выбрасываем исключение

}

}

}

if (stack.Count != 1) // Если в стэке осталось больше одного элемента

{

throw new ArgumentException("Invalid expression"); // Выражение неверно

}

return stack.Pop(); // Возвращаем результат вычислений

}

}

using System;

public class Program

{

public static void Main(string[] args)

{

string expression = "5 3 + 8 2 - \*"; // Задаем постфиксное выражение

double result = PostfixCalculator.Calculate(expression); // Вычисляем значение выражения

Console.WriteLine($"Result: {result}"); // Выводим результат

}

}

source: https://metanit.com/sharp/tutorial/4.8.php