Стек

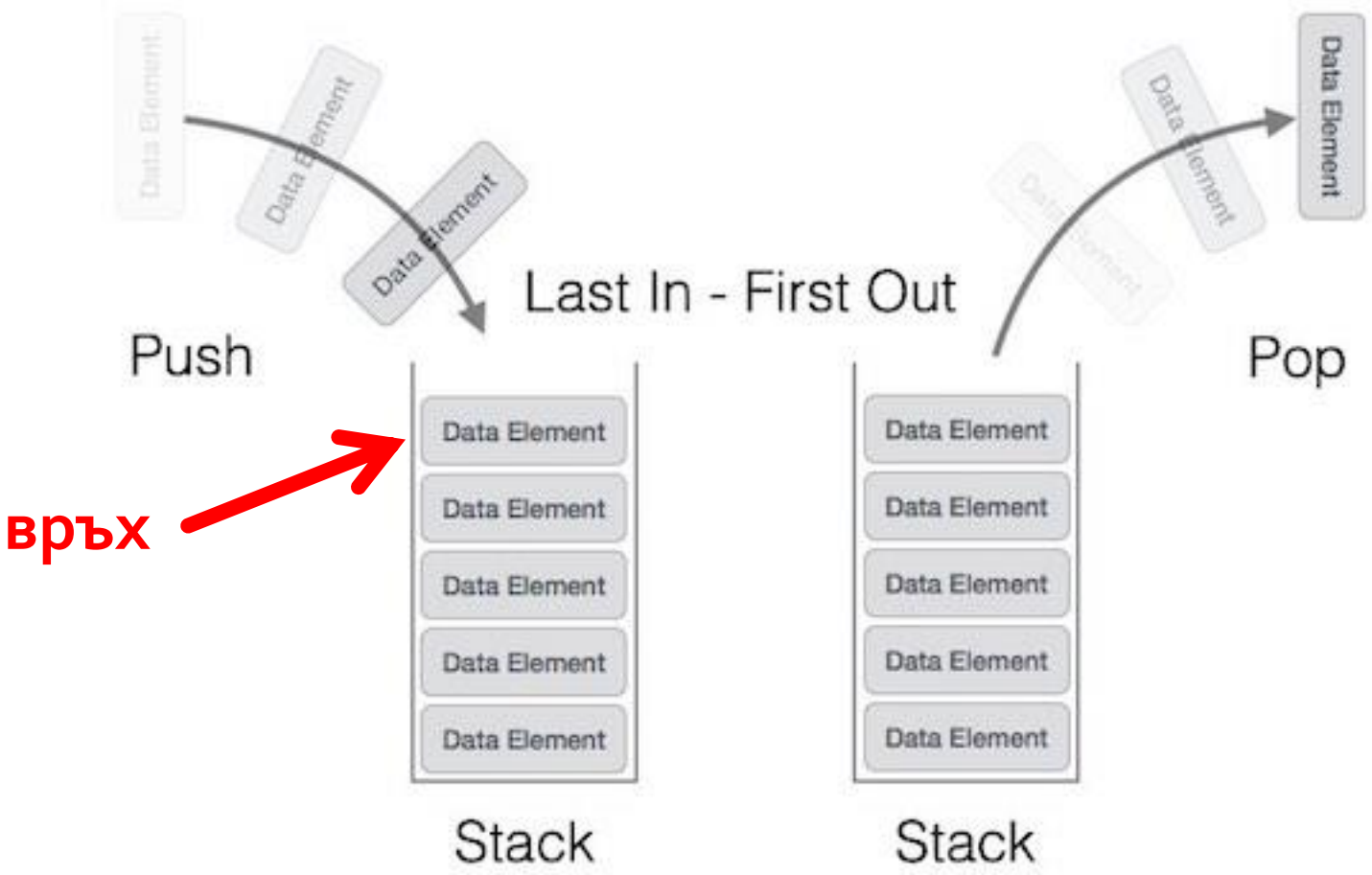
доц. д-р Нора Ангелова

Стек

- **Основни характеристики**
  - **съставна** структура от данни
  - **хомогенна** структура от данни
  - **линейна** структура от данни

Стек

- Поведение
  - „последен влязъл - пръв излязъл“ (LIFO)



Стек

Логическо представяне

- крайна редица от елементи от един и същ тип.
- операции – операциите включване и изключване са допустими само за върха на стека.
- пряк достъп – възможен е само до елемента, намиращ се на върха на стека.

Стек

Операции:

- empty() – проверка дали стекът е празен.
- push(x) – включване на елемент на стек.
- pop() – изключване на елемент от стек.
- top() – връщане на стойността на върха на стека.

Стек

Физическо представяне

- последователно
- свързано

Стек

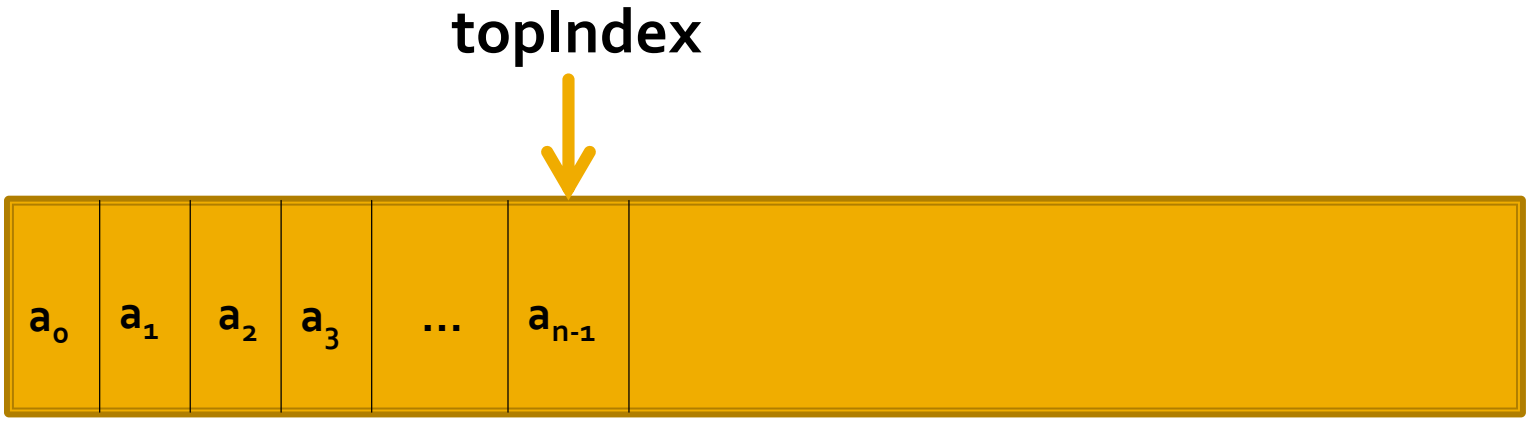
Последователно представяне

- запазва се блок от памет, в който стекът расте и се съкращава.

Как да заделим блок от последователни елементи в паметта?

Стек - последователно представяне

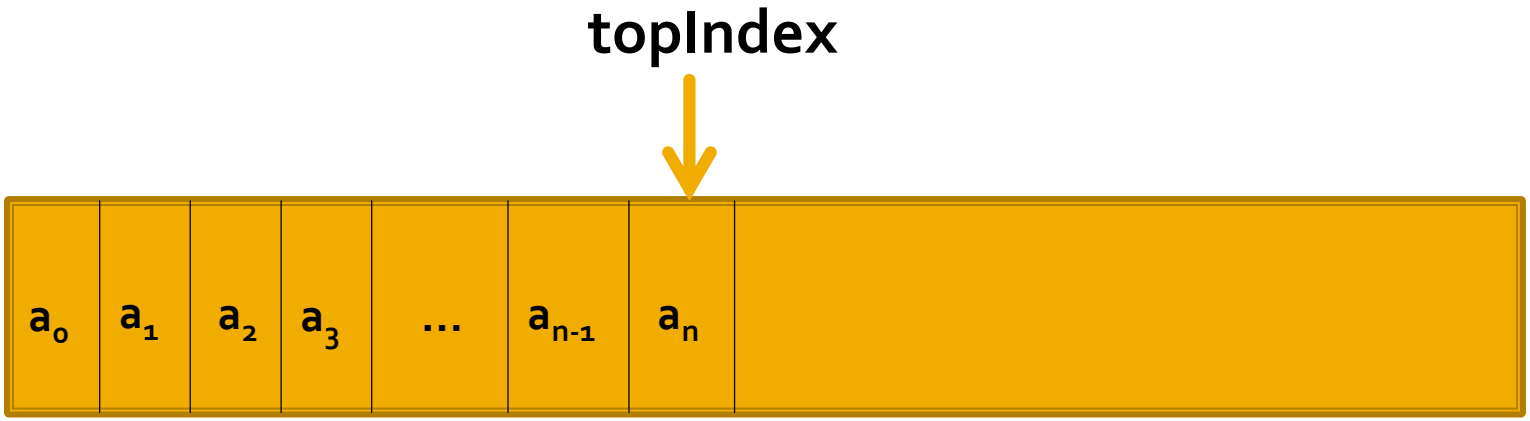
- Масив



Стек - последователно представяне

Реализация с масив

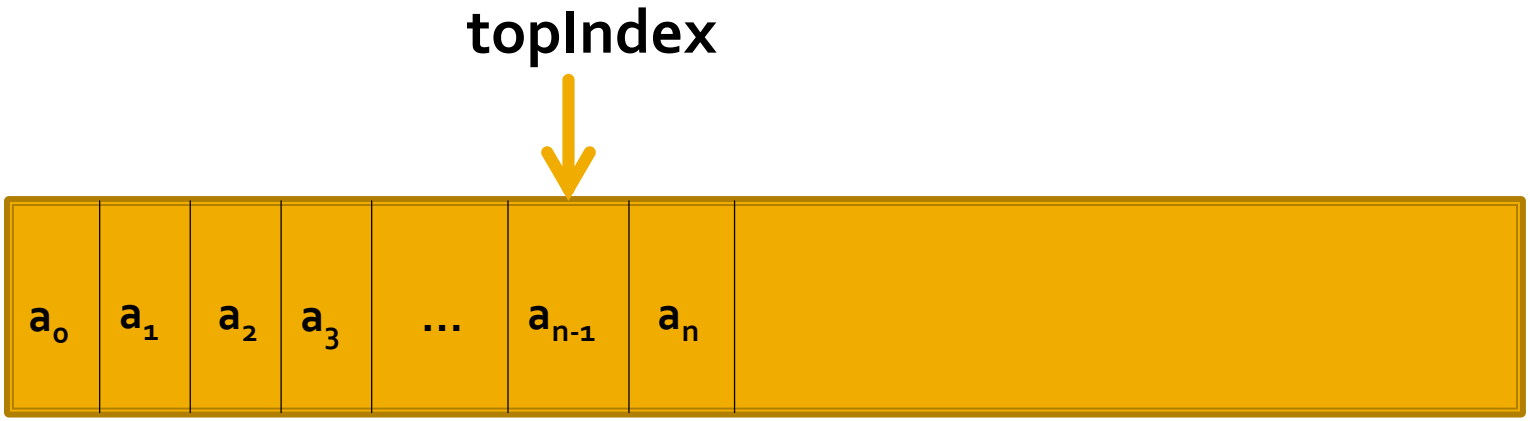
- push($\mathbf{a_n}$) – включва елемента $\mathbf{a_n}$



Стек - последователно представяне

Реализация с масив

- push($\mathbf{a_n}$) – включва елемента $\mathbf{a_n}$
- pop() – изключва елемент



Стек

- Анализ на решението
  - Липсват полета за брой на елементите и капацитет в класа – паметта за всеки обект ще бъде по-малка, но се използва външна константа и трябва да се правят проверки по индекс.
  - Методът full може да бъде част от интерфейсът на класа, ако той е със статичен капацитет.
  - Предполага се, че за всеки тип T са реализирани елементите от голямата петица.

Стек с преоразмеряване

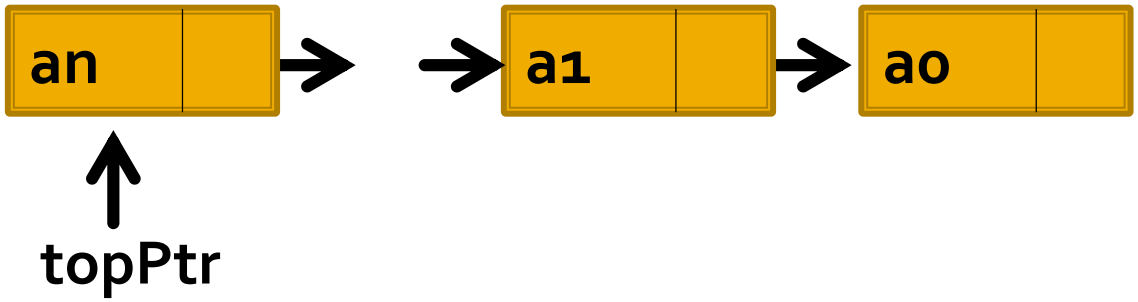
- Реализация на стек с преоразмеряване
  - добавя се поле за текущ капацитет
  - прави се проверка дали е достигната определен процент запълненост – може да е при запълване на 50%, 75% или при достигане на капацитета.
  - добавят се вътрешен метод за копиране на текущите елементи
  - добавя се вътрешен метод за преоразмеряване на стек

Стек

```
template <typename T>
class RStack {
private:
    T* elements;
    int topIndex;                   // Индекс на последния елемент в стека
    int capacity;                   // Капацитет на стека

    bool full() const;              // Проверка дали стек е пълен
    void resize();                  // Разширяване на стек
    void eraseStack();              // Изтриване на паметта
    void copyElements(T const*);    // Копиране на паметта на стек (до capacity)
    void copyStack(RStack const&); // Копиране на стек

public:
    RStack();
    RStack(RStack const&);
    RStack& operator=(RStack const&);
    ~RStack();

    // Move семантики
    RStack(RStack &&);
    RStack& operator=(RStack &&);

    bool empty() const;
    void push(T const& x);
    void pop();
    T top() const;
};
```

Стек

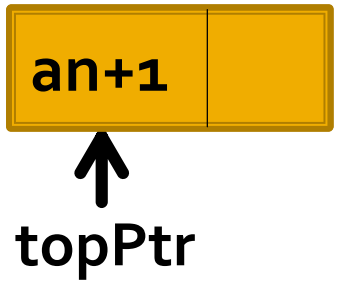
- Свързано представяне



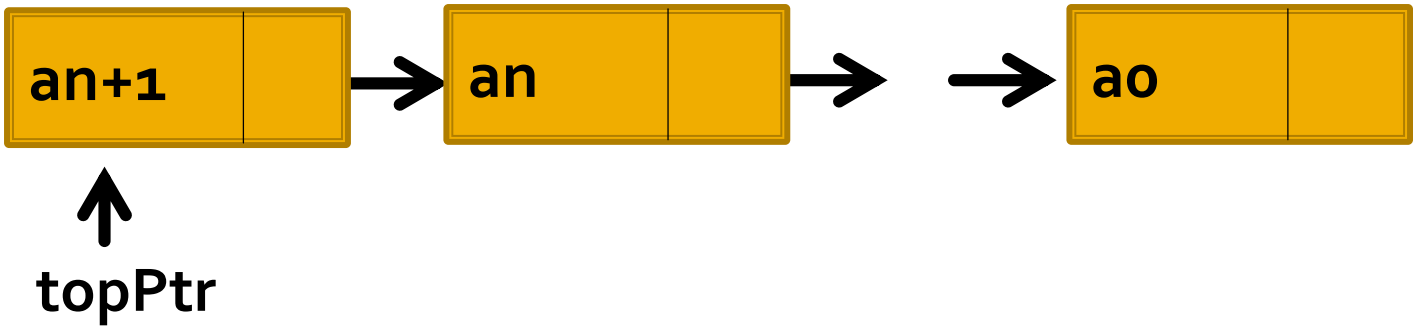
Стек

Свързано представяне

- добавяне на елемент
  - в празен стек



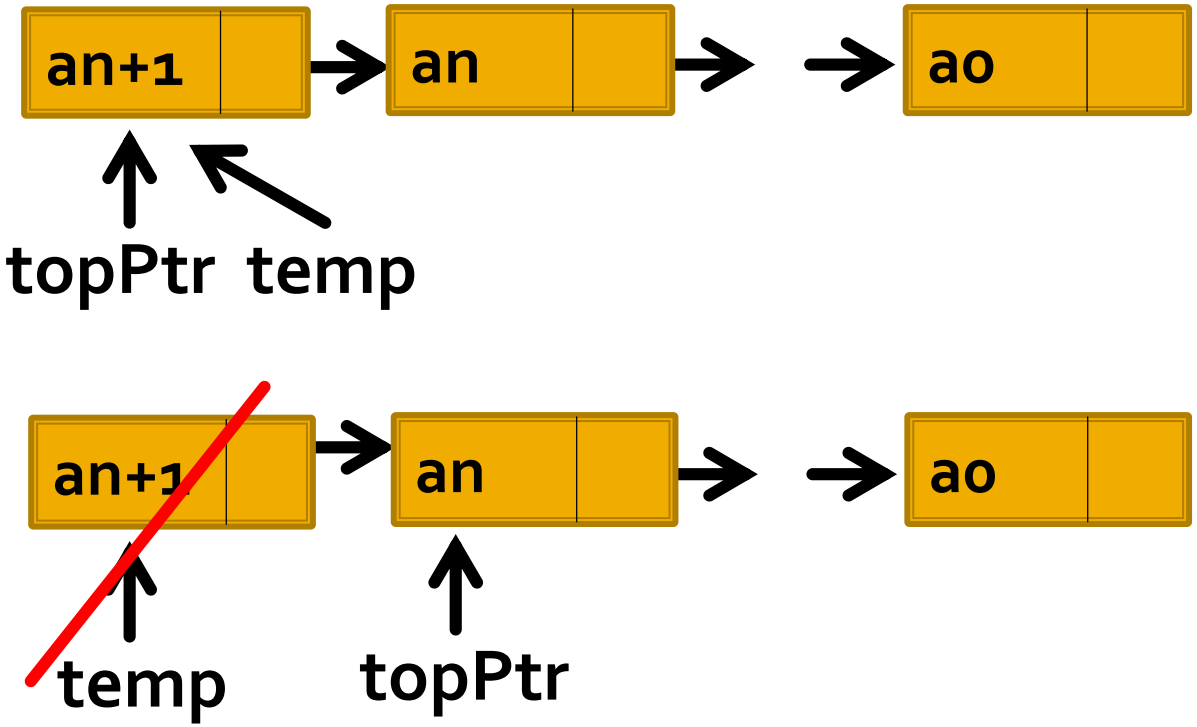
  - в стек с повече от един елемент



Стек

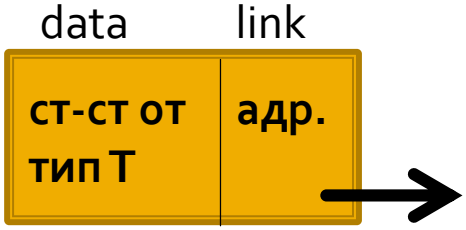
Свързано представяне

- добавяне на елемент
- премахване на елемент



Стек – свързано представяне

```
template <typename T>
struct StackElement {
  T data;
  StackElement<T>* link;
};
```



Стек

```
template <typename T>
class LinkedStack {
    // Вътрешно представяне
    StackElement<T>* topPtr; // указател към връх на стека
    // Помощни методи
    void copyStack(LinkedStack const& other);
    void erase();
public:
    LinkedStack();
    LinkedStack(LinkedStack const& other);
    LinkedStack& operator=(LinkedStack const& other);
    ~LinkedStack();

    // Move семантики
    LinkedStack(LinkedStack && other);
    LinkedStack& operator=(LinkedStack && other);
```

Стек

```
    // Проверка за празнота на стек
    bool empty() const;

    // Включване на елемент
    void push(T const& x);

    // Изключване на елемент
    void pop();

    // Извличане последно включения елемент
    T const& top() const;

    // Извличане последно включения елемент + възможност за неговата
промяната
    T& top();
};
```

Стек

```
// Вариант 1
template <typename T>
void LStack<T>::copyStack(const LStack<T>& stack) {
    topPtr = nullptr;

    if (stack.empty()) {
        return;
    }
    StackElement<T> * lastCopied, toCopy, copied;
    lastCopied = new StackElement<T>;
    lastCopied->data = stack.topPtr->data;

    topPtr = lastCopied;
    StackElement<T> * toCopy = stack.topPtr->link;
    while (toCopy) {
      copied = new StackElement<T>;
      copied->data = toCopy->data;
      lastCopied->link = copied;
      lastCopied = copied;
      toCopy = toCopy->link;
    }
    lastCopied->link = nullptr;
}
```

// Сложност: O(n)

Стек

```
// Вариант 2
template <typename T>
void LStack<T>::copy (StackElement<T>* toCopy) {

    if (toCopy == nullptr) {
        return;
    }

    copy(toCopy->link);
    push(toCopy->data);
}
```

```
template <typename T>
void LStack<T>::copyStack(const LStack<T>& stack) {
    topPtr = nullptr;
    copy(stack.topPtr);
}
```

// Сложност: O(n)

Стек

```
// Вариант 3
```

- Чрез използване на функции push and pop
  - Бележка: с използване на допълнителна структура

// Сложност: O(n)

STL (Standard Template Library)

Библиотека от шаблони, реализираща стандартни структури от данни и алгоритми.

- част от C++ Standard Library

Основни компоненти:

- алгоритми (<algorithm>)
- контейнери (<stack>, <queue>, <list>)
- функционални обекти (<functional>)

** Дава гаранции за сложност на алгоритми и операции над СД*

STL (Стек)

`std::stack<T>`

#include <stack>

Интерфейс:

- stack() — създаване на празен стек
- empty() — проверка за празнота на стек
- push(x) — включване на елемент на стек
- pop() — изключване на елемент от стек (void)
- top() — последен елемент на стека (reference || const_reference)
- size() — дължина на стека

- ==,!=,,<=,>= — лексикографско сравнение на два стека

Следва продължение...