

Какво беше това?



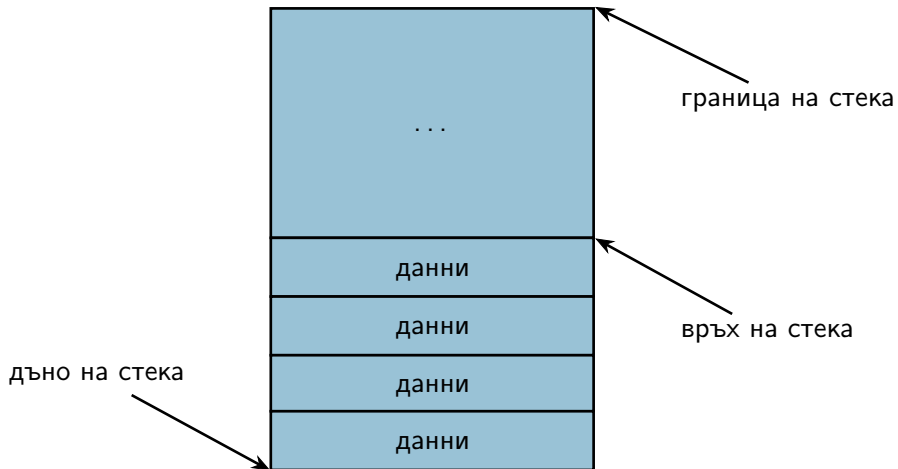
Стек

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

6–13 март 2019 г.

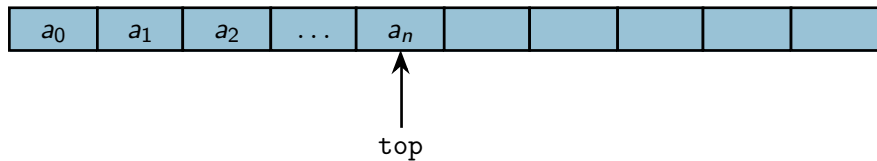
Програмен стек



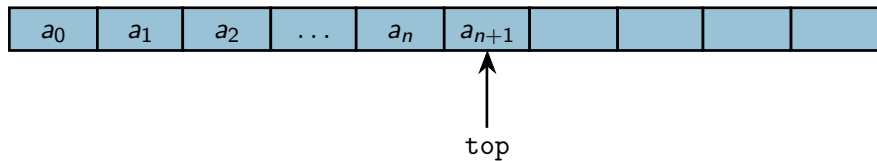
Структура от данни стек

- Организация на данни от тип Last In First Out (LIFO)
- Операции:
 - създаване на празен стек (create)
 - проверка за празнота (empty)
 - включване на елемент (push)
 - намиране на последния включен елемент (peek)
 - изключване на последния включен елемент (pop)

Последователно представяне на стек

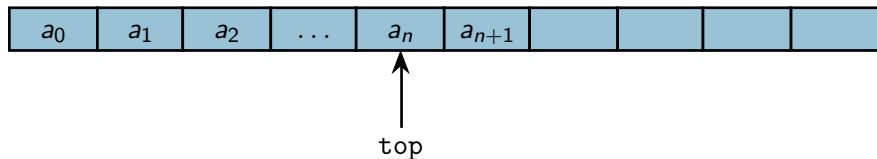


Последователно представяне на стек



- включване на елемент (push)

Последователно представяне на стек



- включване на елемент (push)
- изключване на елемент (pop)

Примерни приложения на стек

- 1 Намиране на записа на дадено число в k-ична бройна система

$$12 = 6k + 0$$

$$6 = 3k + 0$$

$$3 = 1 \cdot k + 1$$

$$1 = 0 \cdot k + 1$$

$$\underline{\underline{0}}$$

$$1100_{(2)} = 12_{(10)}$$

2
2
2
2

Примерни приложения на стек

$$\begin{array}{ccccccc}
 & & -6 & & -11 & & 5 \\
 9 & & 2 & * & & & 4 & + \\
 3 & - & & & -12 & & &
 \end{array}$$

- 1 Намиране на записа на дадено 1 $+$ число в k -ична бройна система
- 2 Пресмятане на аритметичен израз

$$\left((2+3) * 4 - 5 \right) \quad \left((2 + (3 * 4)) - 5 \right)$$

$$\left(4 + \left(2 * \left(3 - \left(4 + 5 \right) \right) \right) \right)$$

$\underbrace{\hspace{10em}}_9$
 $\underbrace{\hspace{10em}}_{-6}$
 $\underbrace{\hspace{10em}}_{-12}$

Примерни приложения на стек

$$\begin{array}{ccc}
 () [] \{ \} & \times & \{ 2 + [(3 - 4) + 5] \} \\
) 2 (& \times & \\
 & \times &
 \end{array}$$

- 1 Намиране на записа на дадено число в k-ична бройна система
- 2 Пресмятане на аритметичен израз
- 3 Проверка за коректност на вложени скоби

$$(2 + 3))$$

$$((2 + 3))$$

$$(2 + (3 + 4))$$

$$[2 + 3]$$

)

Ограничения на последователния стек

- Нашата реализация изисква предварително да зададем горна граница на броя на елементите в стека!

Ограничения на последователния стек

- Нашата реализация изисква предварително да зададем горна граница на броя на елементите в стека!
- Ако стекът се препълни, програмата няма да може да продължи да работи... въпреки че компютърът има много налична свободна памет

Ограничения на последователния стек

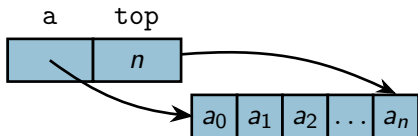
- Нашата реализация изисква предварително да зададем горна граница на броя на елементите в стека!
- Ако стекът се препълни, програмата няма да може да продължи да работи... въпреки че компютърът има много налична свободна памет
- Дали е възможно стекът да се “разширява” при нужда?

Разширяващ се стек

- Обектът няма да съдържа целия масив
- Ще се пази указател към масив в динамичната памет

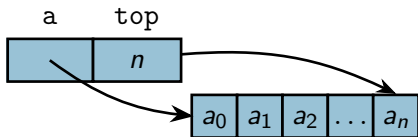
Разширяващ се стек

- Обектът няма да съдържа целия масив
- Ще се пази указател към масив в динамичната памет



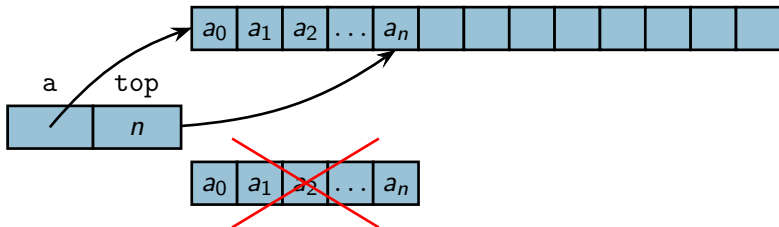
Разширяващ се стек

- Обектът няма да съдържа целия масив
- Ще се пази указател към масив в динамичната памет
- При нужда стекът ще се разширява



Разширяващ се стек

- Обектът няма да съдържа целия масив
- Ще се пази указател към масив в динамичната памет
- При нужда стекът ще се разширява



Ограничения на разширяващия се стек

- При разширяване трябва да се копират всички съществуващи данни!

Ограничения на разширяващия се стек

- При разширяване трябва да се копират всички съществуващи данни!
- Операцията push обикновено е бърза, но ако се случи да правим разширение може да са доста по-бавни

Ограничения на разширяващия се стек

- При разширяване трябва да се копират всички съществуващи данни!
- Операцията `push` обикновено е бърза, но ако се случи да правим разширение може да са доста по-бавни
- Ако стекът се пълни рядко, то в по-голямата част от живота му паметта няма да се използва

Ограничения на разширяващия се стек

- При разширяване трябва да се копират всички съществуващи данни!
- Операцията push обикновено е бърза, но ако се случи да правим разширение може да са доста по-бавни
- Ако стекът се пълни рядко, то в по-голямата част от живота му паметта няма да се използва
- Дали може да се направи стек, при който:

Ограничения на разширяващия се стек

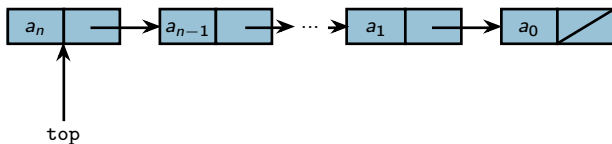
- При разширяване трябва да се копират всички съществуващи данни!
- Операцията `push` обикновено е бърза, но ако се случи да правим разширение може да са доста по-бавни
- Ако стекът се пълни рядко, то в по-голямата част от живота му паметта няма да се използва
- Дали може да се направи стек, при който:
 - не се налага копиране на памет

Ограничения на разширяващия се стек

- При разширяване трябва да се копират всички съществуващи данни!
- Операцията push обикновено е бърза, но ако се случи да правим разширение може да са доста по-бавни
- Ако стекът се пълни рядко, то в по-голямата част от живота му паметта няма да се използва
- Дали може да се направи стек, при който:
 - не се налага копиране на памет
 - не се държи излишна памет

Свързано представяне на стек

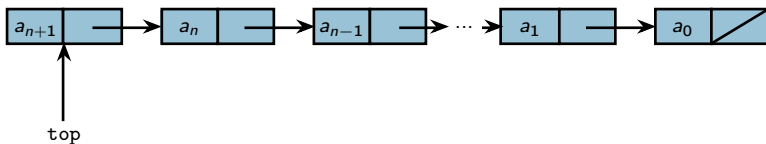
Представяме стека като “верига” от двойни кутии



```
struct StackElement {  
    int data;  
    StackElement* next;  
};
```


Свързано представяне на стек

Представяме стека като “верига” от двойни кутии

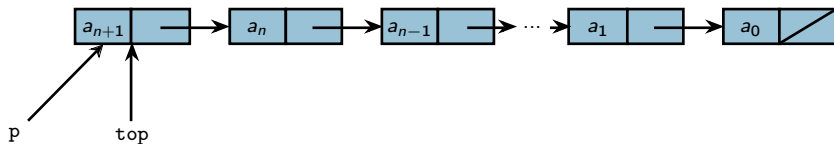


```
struct StackElement {
    int data;
    StackElement* next;
};
```

- включване на елемент (push)

Свързано представяне на стек

Представяме стека като “верига” от двойни кутии

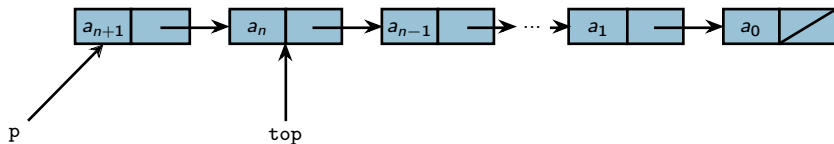


```
struct StackElement {
    int data;
    StackElement* next;
};
```

- включване на елемент (push)
- изключване на елемент (pop)

Свързано представяне на стек

Представяме стека като “верига” от двойни кутии

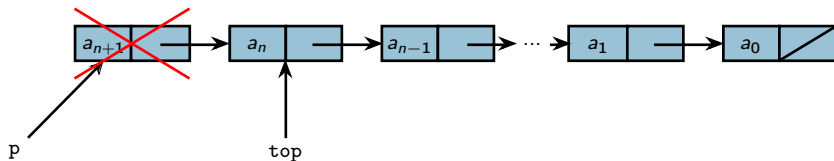


```
struct StackElement {
    int data;
    StackElement* next;
};
```

- включване на елемент (push)
- изключване на елемент (pop)

Свързано представяне на стек

Представяме стека като “верига” от двойни кутии



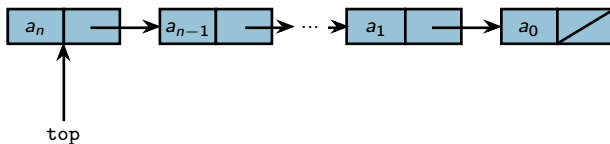
```
struct StackElement {
    int data;
    StackElement* next;
};
```

- включване на елемент (push)
- изключване на елемент (pop)



Свързано представяне на стек

Представяме стека като “верига” от двойни кутии



```
struct StackElement {
    int data;
    StackElement* next;
};
```

- включване на елемент (push)
- изключване на елемент (pop)

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет
- Често се заделят и освобождават малки парчета памет

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет
- Често се заделят и освобождават малки парчета памет
- Какво се случва при копиране на стек?

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет
- Често се заделят и освобождават малки парчета памет
- Какво се случва при копиране на стек?
 - `LinkedList s2 = s1; s1.pop(); s2.pop(); s2.push(10);`

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет
- Често се заделят и освобождават малки парчета памет
- Какво се случва при копиране на стек?
 - `LinkedList s2 = s1; s1.pop(); s2.pop(); s2.push(10);`
- Какво се случва при унищожаване на стек?

```
for(int i = 0; i < 1E8; i++) { LinkedList s; .... }
```

Ограничения на свързания стек

- За всеки елемент се изразходва 2-3 пъти повече памет
- Често се заделят и освобождават малки парчета памет
- Какво се случва при копиране на стек?
 - `LinkedList s2 = s1; s1.pop(); s2.pop(); s2.push(10);`
- Какво се случва при унищожаване на стек?

```
for(int i = 0; i < 1E8; i++) { LinkedList s; .... }
```

```
for(int i = 0; i < 1E8; i++) {  
    LinkedList* s = new LinkedList;  
    ....  
    delete s;  
}
```