

# Stek

## Slajdovi sa predavanja<sup>1</sup>

© Goodrich, Tamassia, Goldwasser

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2022.

---

<sup>1</sup>Po uzoru na materijale sa: <https://github.com/mbranko/asp-slajdovi>

# Apstraktni tipovi podataka

- ATP je apstrakcija strukture podataka
- ATP definiše
  - podatke koji se čuvaju
  - operacije nad podacima
  - uslovi kada dolazi do greške
- primer: ATP koji modeluje sistem za trgovinu akcijama
  - podaci: kupi/prodaj narudžbe
  - operacije:
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - greške:
    - nepostojeće akcije
    - otkazivanje narudžbe

# Stek ATP

- **stek** (stack) ATP čuva proizvoljne objekte
- ubacivanje i uklanjanje poštuje LIFO (last-in-first-out) princip
- primer: PEZ bombone :)
- glavne operacije:
  - **push**(object): dodaje element na vrh steka
  - object **pop**(): skida element sa vrha steka
- dodatne operacije:
  - object **top**(): vraća najviši element bez skidanja
  - integer **len**(): vraća broj elemenata na steku
  - boolean **is\_empty**(): vraća True ako je stek prazan



# Primer operacija nad stekom

operacija	rezultat	sadržaj steka
S.push(5)	–	[5]
S.push(3)	–	[5,3]
len(S)	2	[5,3]
S.pop()	3	[5]
S.is_empty()	False	[5]
S.pop()	5	[]
S.is_empty()	True	[]
S.pop()	greška	[]
S.push(7)	–	[7]
S.push(9)	–	[7,9]
S.top()	9	[7,9]
S.push(4)	–	[7,9,4]
len(S)	3	[7,9,4]
S.pop()	4	[7,9]
S.push(6)	–	[7,9,6]
S.push(8)	–	[7,9,6,8]
S.pop()	8	[7,9,6]

# Primene steka

- neposredne primene
  - istorija poseta stranicama u web čitaču
  - undo sekvenca u tekst editoru
  - lanac rekurzivnih poziva u programu
- indirektne primene
  - pomoćna struktura podataka za mnoge algoritme
  - komponenta u okviru drugih struktura podataka

# Stek pomoću niza

- implementiraćemo stek pomoću niza
- dodajemo elemente s leva u desno
- posebna promenljiva čuva indeks poslednjeg elementa



# Stek pomoću niza <sub>2</sub>

- ako se niz popuni...
- nova operacija push mora da proširi niz i iskopira sve elemente



# Performanse steka

- neka je  $n$  broj elemenata na steku
- prostor u memoriji je  $O(n)$
- svaka operacija je  $O(1)$  (amortizovano za push)



# Implementacija steka u Pythonu

```
class ArrayStack:
    def __init__(self):
        self._data = []

    def __len__(self):
        return len(self._data)

    def is_empty(self):
        return len(self._data) == 0

    def push(self, e):
        self._data.append(e)

    def top(self):
        if self.is_empty():
            raise Empty('stack is empty')
        return self._data[-1]

    def pop(self):
        if self.is_empty():
            raise Empty('stack is empty')
        return self._data.pop()
```

# Primer: uparivanje zagrada

- svako (, {, [ mora biti upareno sa ), }, ]
- ispravno: ( )(( )){([ ( ))}
- ispravno: ((( ))(( )){([ ( ))}
- neispravno: )(( )){([ ( ))}
- neispravno: ({ [ ]})
- neispravno: (

# Primer: uparivanje zagrada

**ParenMatch**( $X, n$ )

**Input:** niz  $X$  od  $n$  tokena, koji mogu biti zagrada, promenljiva, aritmetički operator ili broj

**Output:** True akko su zagrade ispravno napisane

Neka je  $S$  prazan stek

**for**  $i \leftarrow 0$  **to**  $n - 1$  **do**

**if**  $X[i]$  je otvorena zagrada **then**

$S.push(X[i])$

**else if**  $X[i]$  je zatvorena zagrada **then**

**if**  $S$  je prazan **then**

**return** False {nema odgovarajuće otvorene zagrade}

**if**  $S.pop()$  ne odgovara vrsti zagrade od  $X[i]$  **then**

**return** False {pogrešna vrsta zagrade}

**if**  $S$  je prazan **then**

**return** True {svi simboli su u korektnom redosledu}

**else**

**return** False {neki simboli nemaju svoj par}

# Primer: uparivanje zagrada

```
def is_matched(expr):
    """Vraća True ako se sve zagrade podudaraju."""
    lefty = '([{'          # otvorene zagrade
    righty = ')]}'         # zatvorene zagrade
    S = ArrayStack()
    for c in expr:
        if c in lefty:
            S.push(c)        # stavi zgradu na stek
        elif c in righty:
            if S.is_empty():
                return False  # nema sa čime da se upari
            if righty.index(c) != lefty.index(S.pop()):
                return False  # pogrešna zagrada
    return S.is_empty()     # da li su svi simboli upareni?
```

# Primer: uparivanje HTML tagova

- svako `<x>` mora biti upareno sa `</x>`
- i pravilno ugnježdjeno:
  - `<x><y>...</y></x>` je OK
  - `<x><y>...</x></y>` nije OK

```

<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine. The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>

```

## The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

# Primer: uparivanje HTML tagova

```
def is_matched_html(raw):
    """Vraća True ako se HTML tagovi podudaraju."""
    S = ArrayStack()
    j = raw.find('<') # nađi prvi '<'
    while j != -1:
        k = raw.find('>', j+1) # nađi sledeći '>'
        if k == -1:
            return False # neispravan tag
        tag = raw[j+1:k] # odbaci < >
        if not tag.startswith('/'): # ovo je otvarajući tag
            S.push(tag)
        else: # ovo je zatvarajući tag
            if S.is_empty():
                return False # nema svog para
            if tag[1:] != S.pop():
                return False # nije odgovarajući par
            j = raw.find('<', k+1) # nađi sledeći '<'
    return S.is_empty() # svi otvarajući tagovi upareni?
```