RI301
Strukture podataka

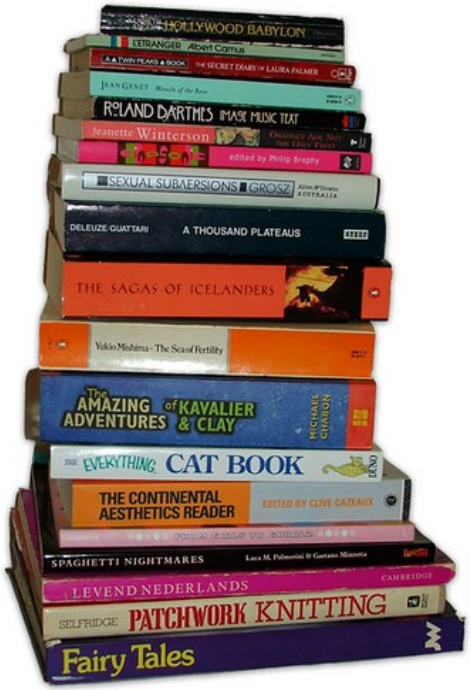
dr.sc. Edin Pjanić

Pregled predavanja

- Stog (stack)
  - osnovne operacije
  - neki načini implementacije

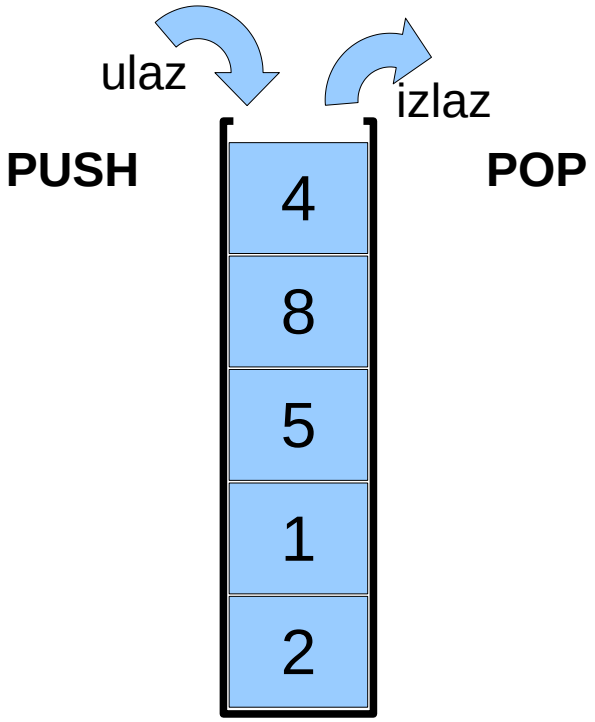
Stog (stack)

- Primjeri iz života za stog (stack)



ATP: stog (stack)

- ... je kolekcija elemenata u linearnom redu (linearna struktura podataka, tj. **lista**) sa osobinama:
  - pristup elementima samo sa jedne strane
- Za stog važi da su sve operacije $O(1)$



Stack:
LIFO - Last In First Out

Stog (stack) – osnovne operacije

Stog (stack)	
`void push(T x)` `void ubaci(T x)`	dodavanje
`void pop() ili T pop()` `void izbaci() ili T izbaci()`	čitanje i uklanjanje
`T& top()` `T& gornji()`	čitanje bez uklanjanja
`bool empty()` `bool je_prazan()`	provjera da li je struktura prazna

Moramo obratiti pažnju da li je struktura prazna pri operacijama uklanjanja ili čitanja elementa.

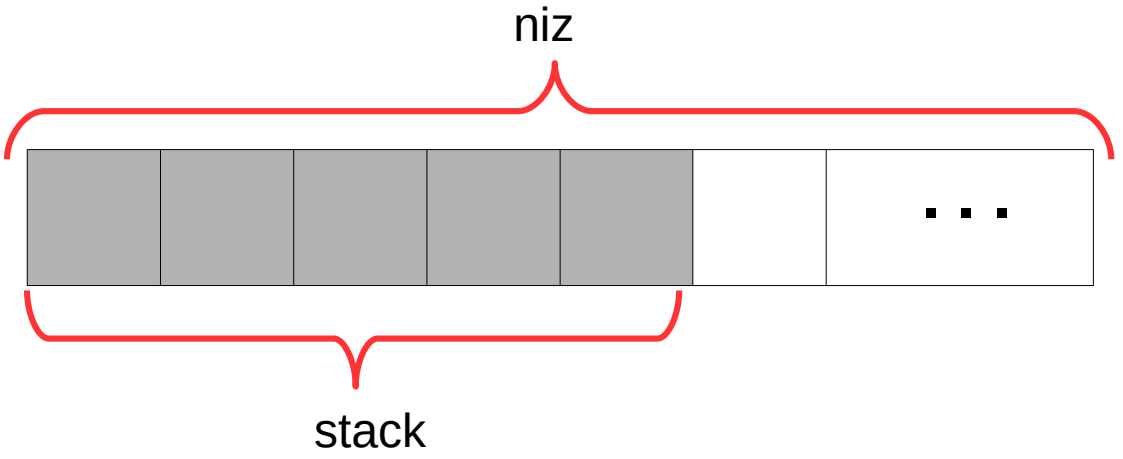
- Moguće je definisati i ostale operacije (metode, operatore), prema potrebi.

Stack – načini implementacije

- Stack (stog) je lista.
- Implementacija je moguća na razne načine, kao i svaka lista.
- Moramo voditi računa o specifičnostima ove strukture.

Stack – implementacija pomoću niza

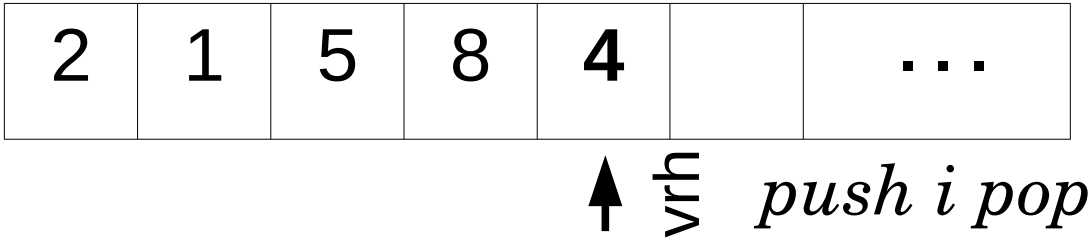
- S obzirom na to da je stack lista, pomoću niza stack možemo realizovati na način da se elementi ove strukture nalaze od početka niza pa nadalje, kao kod standardne liste.



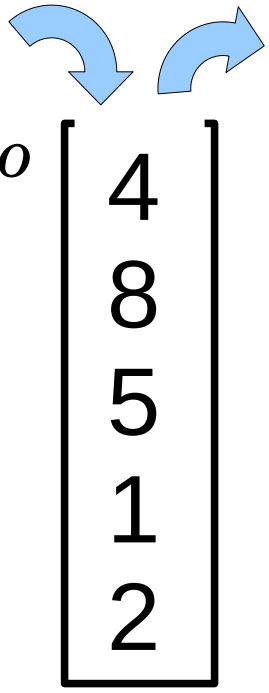
- S obzirom na to da se ovoj strukturi pristupa samo s jedne strane (sa tzv. vrha), trebamo odlučiti kako poredati elemente stoga, odnosno gdje će se nalaziti vrh.

Stack – implementacija pomoću niza

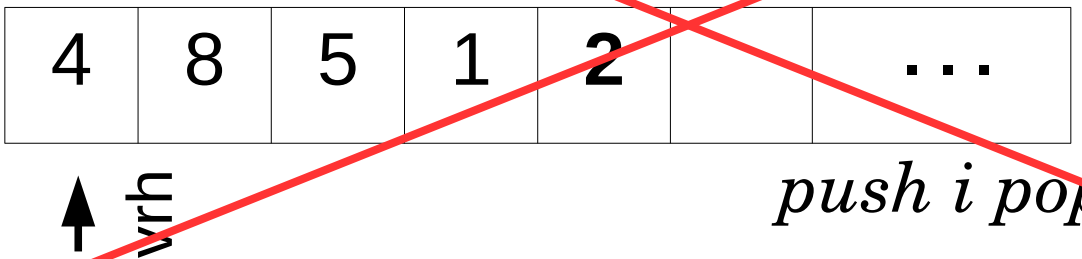
- Stack pomoću niza možemo realizovati korištenjem jedne od dvije strategije (koja je bolja?):
- 1. vrh stacka je na kraju liste
  - dodajemo na kraj i uklanjamo sa kraja liste



push i pop: O(1) => dobro



2. vrh stacka je na početku liste
  - dodajemo na početak i uklanjamo sa početka liste



push i pop: O(n) => loše

Stack – implementacija pomoću niza

```
template <typename T>
class StackNiz
{
  public:
    StackNiz(int k);                  // k - inicijalni kapacitet
    StackNiz(StackNiz const &);       // copy constructor
    StackNiz(StackNiz &&);            // move constructor
    ~StackNiz();
    template <typename U>
     StackNiz& push(U &&);
    T pop(); // void pop();
    T & top();
    const T & top() const;
    int size() const;
    bool empty() const;
    void realocate(int k); // k - nova velicina niza
  private:
    int capacity; // kapacitet (velicina alociranog niza)
    int top_idx;  // indeks vrha stacka
    T * elem;
};
```

Umjesto:

```
StackNiz& push(T const &);
StackNiz& push(T &&);
```

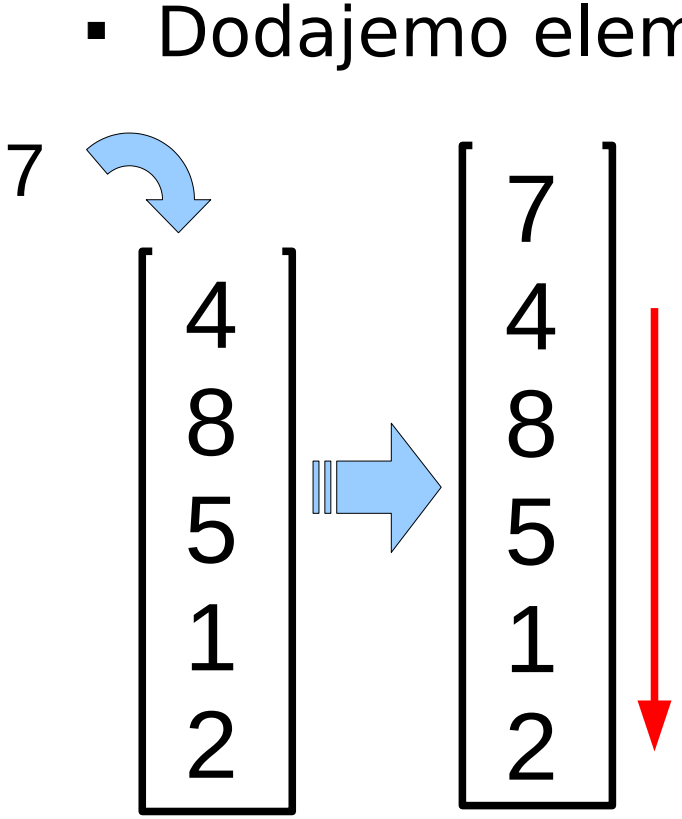
Stack – realociranje

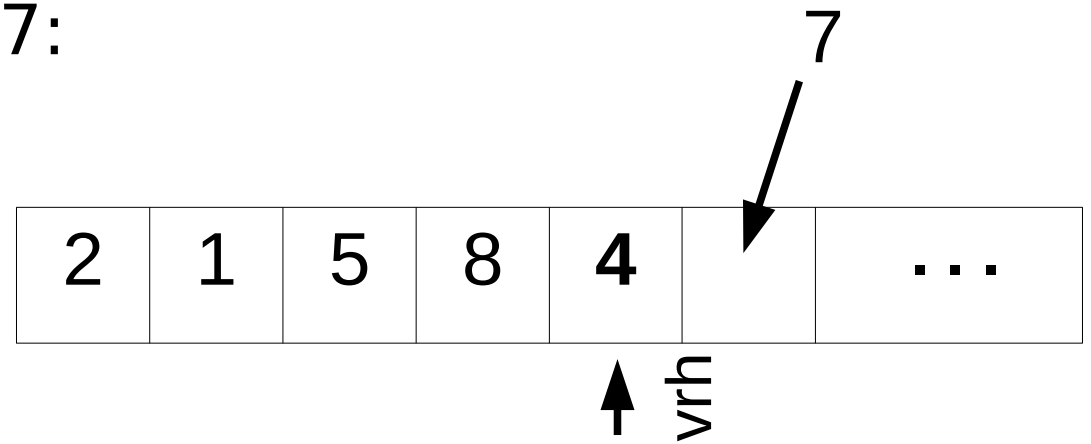
```
template <typename T>
void StackNiz<T>::realocate(int nova_vel)
{
  T * novi_niz = new T[nova_vel];

  int prenos = top_idx;
  if(nova_vel < prenos) // Gubitak podataka. Razmisliti!
    prenos = nova_vel;

  for(int i=0; i<prenos; i++)
    novi_niz[i] = std::move(elem[i]); // Zašto?

  delete[] elem;
  elem = novi_niz;
  capacity = nova_vel;
  top_idx = prenos;
}
```

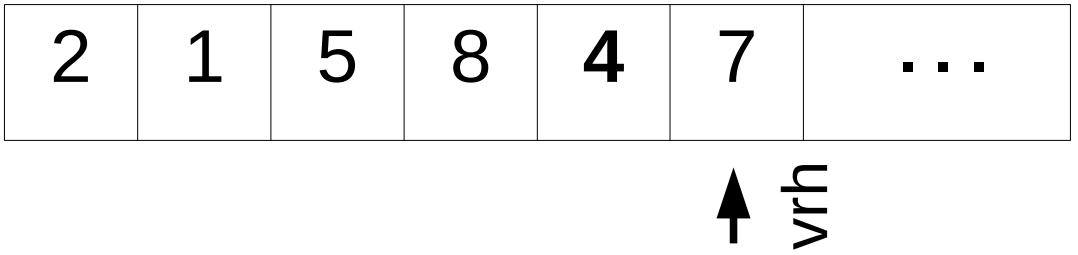
$O(n)$

push – dodavanje novog elementa

- Dodajemo element 7:



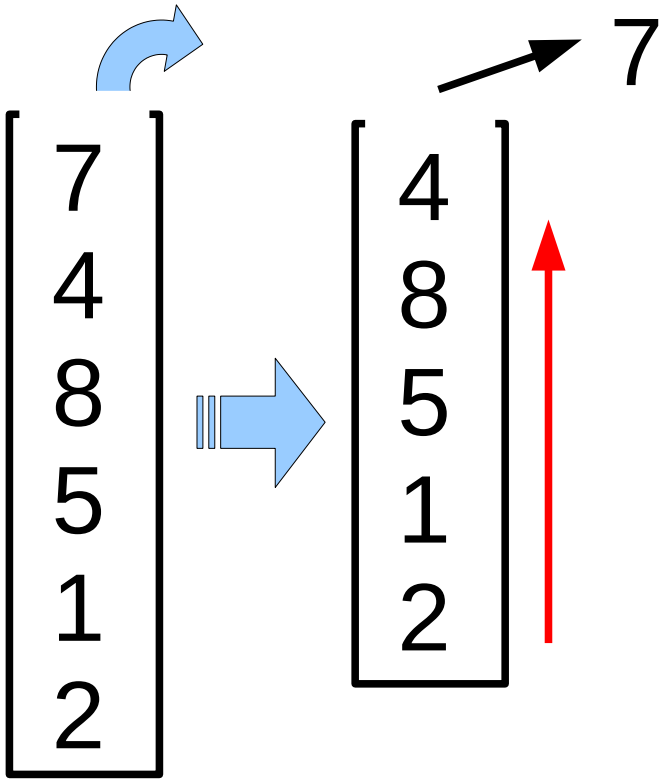


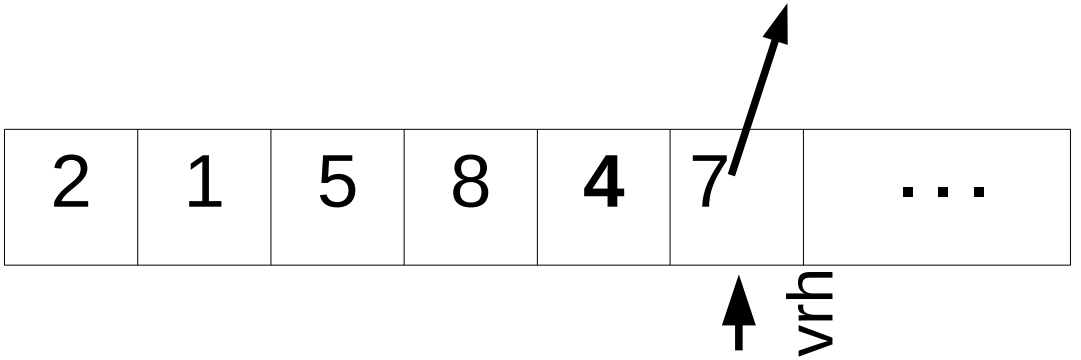


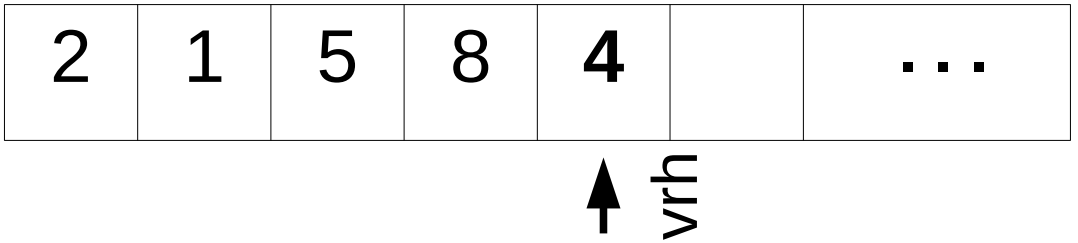
Pomjeriti vrh za jedno mjesto naprijed i postaviti vrijednost tog elementa niza na novu vrijednost (7).
Obratiti pažnju na stanje kad je stack pun:
- ili ćemo generisati grešku
- ili "povećati" niz.

pop – uklanjanje elementa

- Uklanjamo gornji element sa stacka:







“Uklonimo” vrijednost na indeksu na koji pokazuje vrh i vrh smanjimo za jedan.
Obratiti pažnju na stanje kad je stack prazan!

Stack – ubacivanje i izbacivanje elemenata

- Ubacivanje elementa u stack (*push*)

```
template <typename T>
template <typename U>
StackNiz<T>& StackNiz<T>::push(U && x)
{
  top_idx++;
  if(top_idx == capacity)
    realocate(capacity + capacity/2);   // ili drugacije?
  elem[top_idx] = std::forward<U>(x);   // zašto?
  return *this;
}
```

O(n) – ponekad

O(1)

- Izbacivanje elementa iz stacka (*pop*): *O(1)*

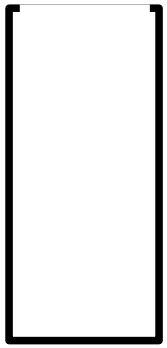
```
template <typename T>
T StackNiz<T>::pop()
{
  top_idx--;
  return std::move(elem[top_idx+1]);//niz je jos uvijek tu
}
```

Složenost push operacije

- Složenost push operacije je O(1) sve dok se ne napuni niz pomoću kojeg je stack implementiran.
- Tada je potrebno izvršiti realokaciju niza čime proširujemo niz. Ova operacije je O(n).
- Dakle, većina push operacija su O(1) a samo neke su O(n).

- Amortizovana analiza - dugoročna analiza sekvence operacija ili aplikacije:
  - dugoročno gledano, imaćemo veliki broj push koji su O(1) i mali broj onih koji su O(n).
  - u tom kontekstu možemo reći da će očekivano prosječno vrijeme za push biti više od vremena jednog upisa u niz (npr. dva upisa) ali konstantno, odnosno O(1)

Stack – konstrukcija, destrukcija



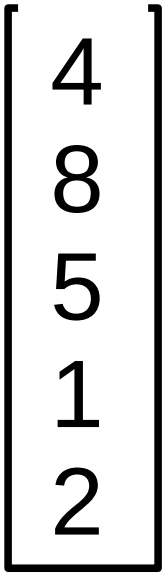
Prazan stack

```
template <typename T>
StackNiz<T>::StackNiz(int vel = DEFAULT_SIZE)
{
  T * elem = new T[vel];
  capacity = vel;
  top_idx = -1;
}
```

$O(n)$

```
template <typename T>
StackNiz<T>::~StackNiz()
{
  delete [] elem;
}
```

$O(n)$

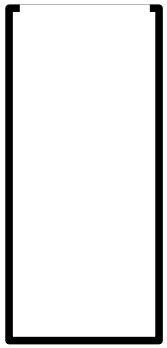


Nije prazan stack

```
template <typename T>
StackNiz<T>::StackNiz(StackNiz&& s)
{
  elem = s.elem;
  capacity = s.capacity;
  op_idx = s.op_idx;
  s.elem = nullptr
}
```

$O(1)$

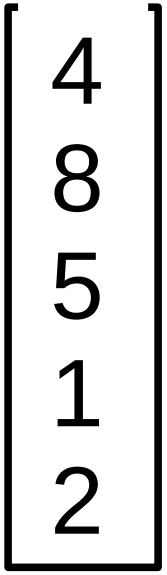
Stack – ostale operacije



Prazan stack

```
template <typename T>
StackNiz<T>& StackNiz<T>::operator=(StackNiz&& s)
{
  std::swap(elem,     s.elem);
  std::swap(capacity, s.capacity);
  std::swap(op_idx,   s.op_idx);
  return *this;
}
```

$O(1)$



Nije prazan stack

```
template <typename T>
int StackNiz<T>::size() const
{
  return top_idx + 1;
}
```

$O(1)$

Stack – ostale operacije

- Čitanje vrijednosti elementa na vrhu (vrijednost elementa sa indeksom na koji pokazuje vrh):

```
template <typename T>
T const & StackNiz<T>::top() const
{
  if( empty() )
    throw std::underflow_error("Prazan stack.");
  return elem[top_idx];
}
```

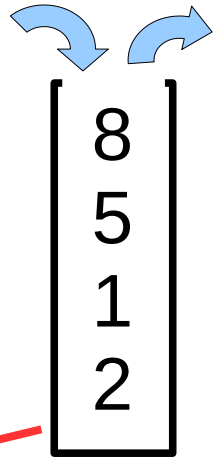
$O(1)$

- Provjera da li je stack prazan (ako je vrh na indeksu -1 => stack je prazan):

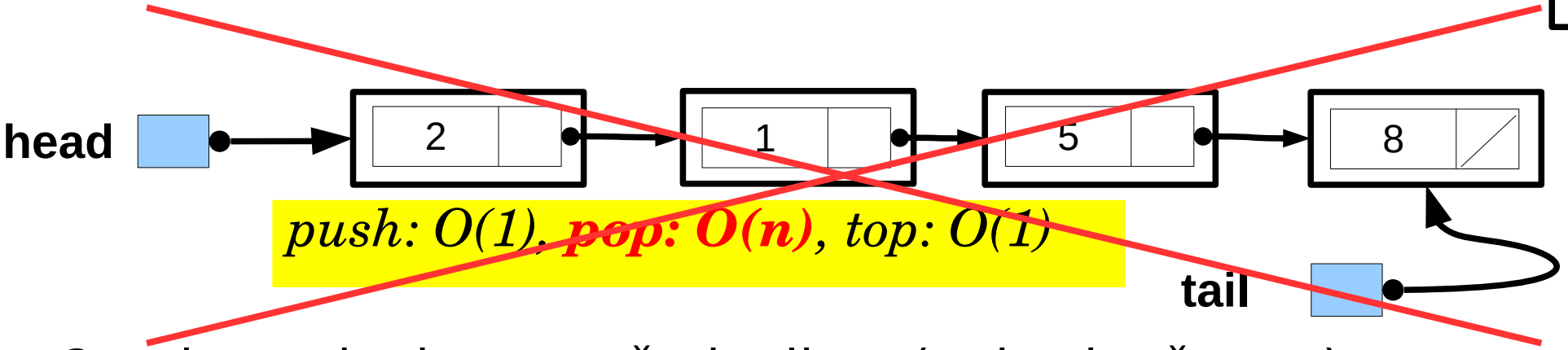
```
template <typename T>
bool StackNiz<T>::empty() const
{
  return (top_idx == -1);
}
```

$O(1)$

Stack – implementacija povezanom listom

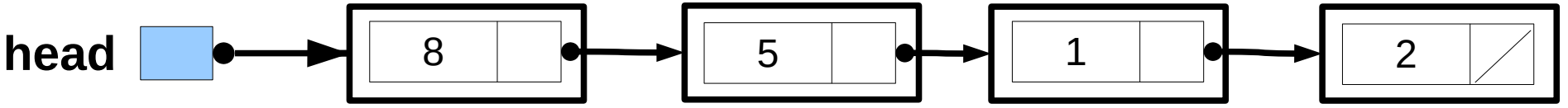


- Stack pomoću povezane liste možemo realizovati korištenjem jedne od dvije strategije (koja je bolja?):
- 1. vrh stacka je na kraju liste (pokazivač `tail`)
  - dodajemo na kraj i uklanjamo sa kraja liste



*push: O(1), **pop: O(n)**, top: O(1)*

2. vrh stacka je na početku liste (pokazivač `head`)
   - dodajemo na početak i uklanjamo sa početka liste

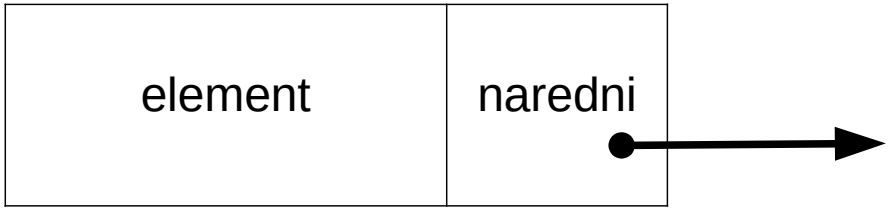


push: O(1), pop: O(1), top: O(1)

BOLJE RJEŠENJE

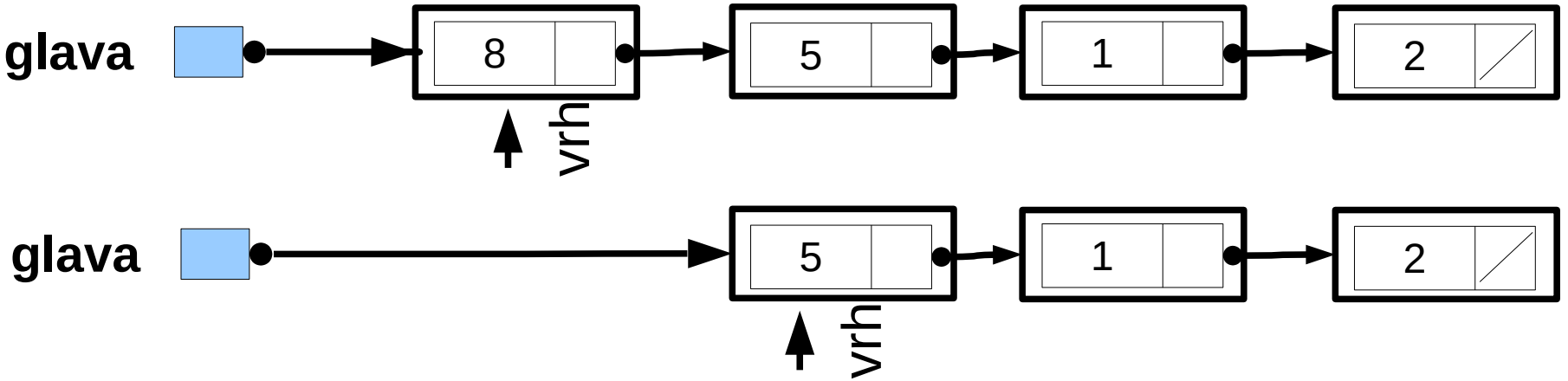
Čvor liste

```
template <typename T>
class Node
{
  public:
    T value;
    Node * next;

    template <typename U>
    Node(U && val) :
        value( std::forward<U>(val) ),
        next(nullptr) {}
    ...
};
```



Stack – pop

- Prilikom uklanjanja elementa (operacija pop) iz ovakvog stacka treba voditi računa o tome da se u nekim implementacijama mora vratiti vrijednost elementa koji se uklanja.




```
T pop()
{
  Node * temp = head;
  head = head->next;
  T v = std::move(temp->value);
  delete temp;
  return v;
}
```

O(1)

Ako nije potrebno vratiti element koji se uklanja:

```
void pop()
{
  Node * temp = head;
  head = head->next;
  delete temp;
}
```

Stack – još neke operacije

- Element na vrhu (element na koji pokazuje glava):

```
T& top()
{
  return head->value;
}
```

```
const T& top() const
{
  return head->value;
}
```

$O(1)$

- Provjera da li je stack prazan (ako glava ne pokazuje ni na jedan element – stack je prazan):

```
bool empty() const
{
  return (head == nullptr);
}
```

$O(1)$

Stack – mogućnosti implementacije

- Stack se može implementirati direktnim korištenjem neke druge strukture podataka na način da se metodi stacka mapiraju u odgovarajuće pozive metoda strukture na kojoj je baziran.
- Npr. ako je stack baziran na dvostruko povezanoj listi (std::list) ili vektoru (std::vector) onda se metodi mogu mapirati kao:

```
template <typename U>
Stack & push(U&& v){
  list_stack.push_back(std::forward<U>(v));
  return *this;
}

int size() const {
  return list_stack.size();
}
```

```
STACK       LIST
push()   → push_back()
pop()    → pop_back()
top()    → back()
size()   → size()
empty()  → empty()
```

Stog (stack) – STL implementacija

- Header `<stack>`, klasa `stack`
- Metodi:
  - `push(x)` – ubacuje element u stack (vraća: ništa)
  - `pop` – izbacuje element sa vrha stacka (vraća: ništa)
  - `top` – vraća element na vrhu stacka
  - `size` – broj elemenata u stacku
  - `empty` – vraća `true` ako je prazan, `false` ako nije
- Primjer korištenja:

```
#include<stack>
  ...
  stack<int> intStek;
  intStek.push(5);
  ...
```