Algoritmi i strukture podataka

03 Stek, red, dek

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2023

Python __Underscore__ methods

- ”Magic methods”
- Obezbeđuju posebne sintaksne odlike
- Preklapanje operatora

- http://www.siafoo.net/article/57

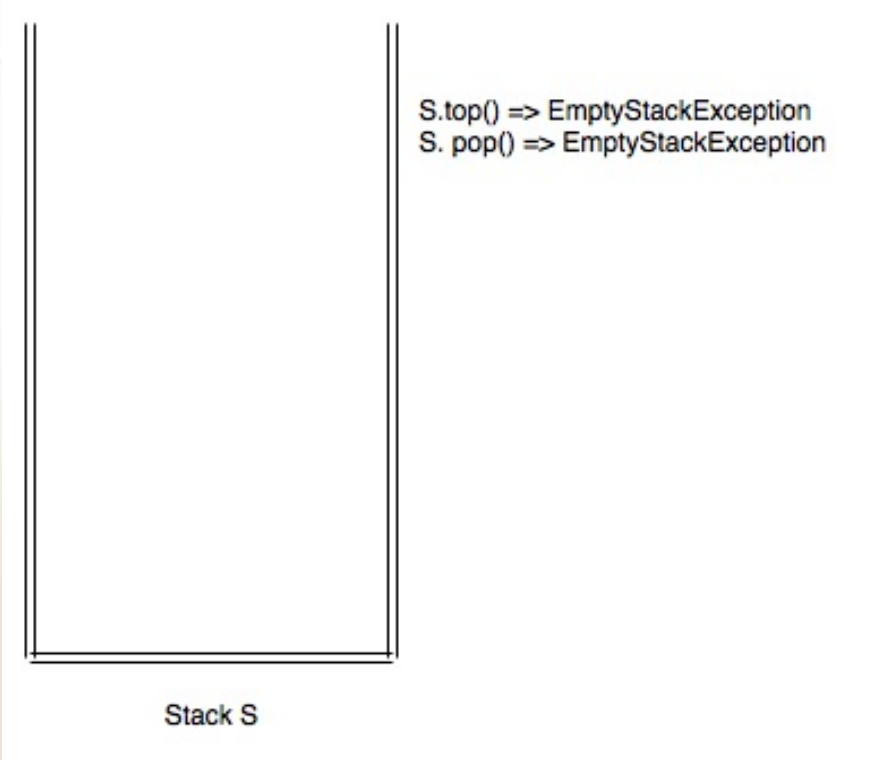
Python __Underscore__ methods

- Za ovaj termin vežbi će nam biti potrebne sledeće metode:
- __init__(self, [..])
  - Pokreće inicijalizaciju objekta klase.
  - Za inicijalizaciju objekta klase MyClass kao
  - my_object = MyClass(23, ‘foo’), gde se 23 i ‘foo’ prosleđuju kao dodatni argumenti __init__ metode
- __len__(self)
  - Vraća dužinu kontejnera. Primenljivo i na promenljive i na nepromenljive kontejnere.
  - Poziva se kao len(kontejner)

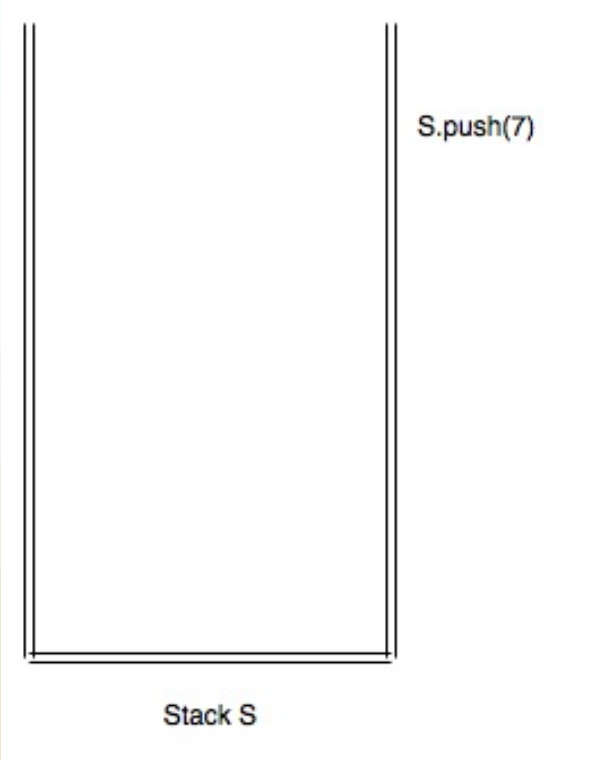
Stek

- LIFO struktura
- Operacije:
  - `S.push(e)` – Element *e* se dodaje na vrh steka
  - `S.pop()` – Destruktivno čitanje - uklanja se element na vrhu steka koji se vraća kao povratna vrednost metode
  - `S.top()` – Čitanje – vraća se element na vrhu steka
  - `S.is_empty()` – Proverava da li je stek prazan (rezultat tipa boolean)
  - `len(S)` – Pronalazi i vraća broj elemenata na steku *S*

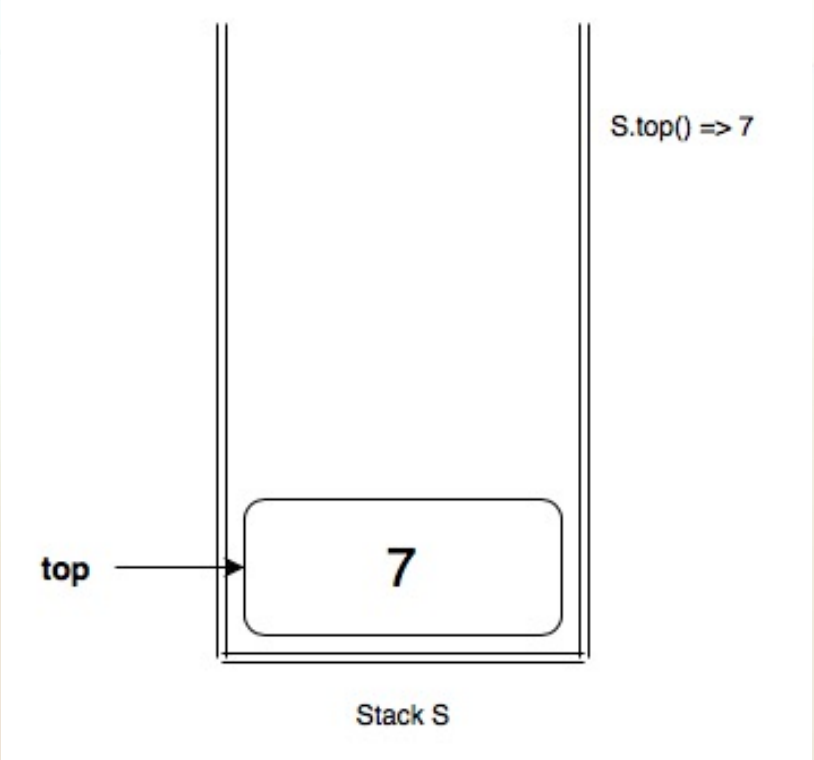
Stek – prikaz upotrebe



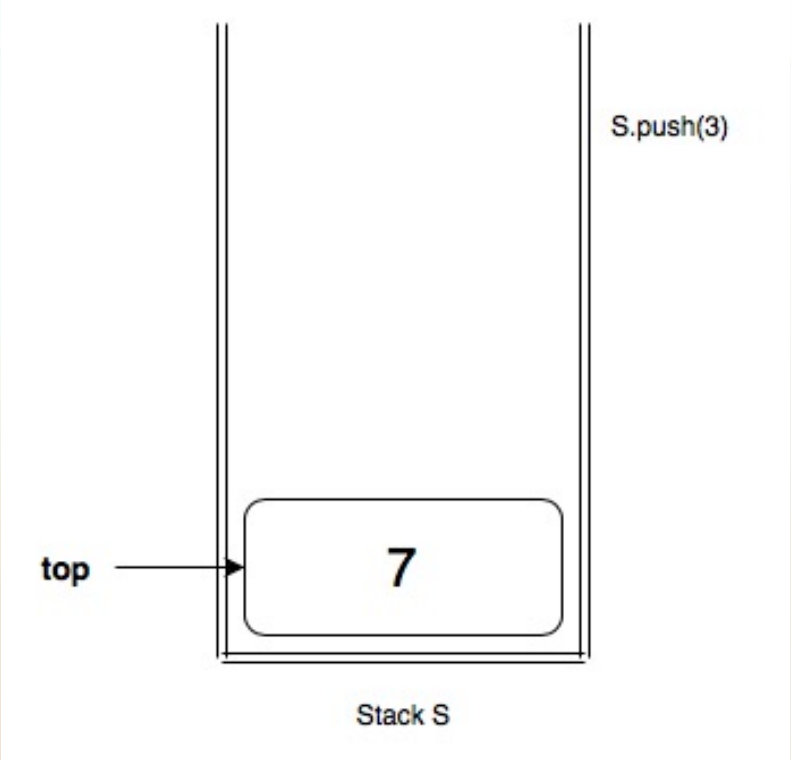
Stek – prikaz upotrebe



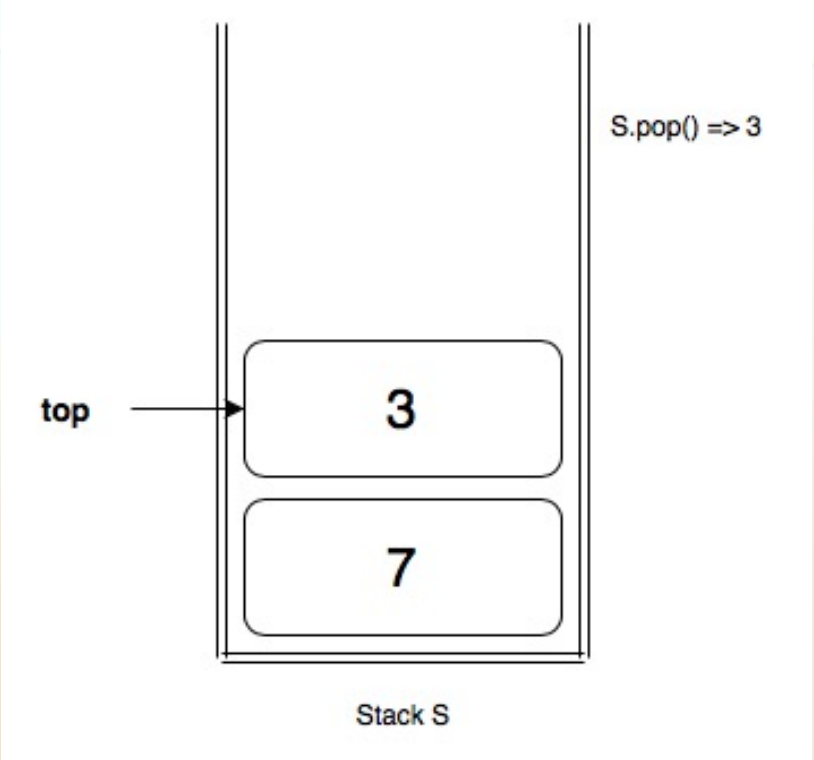
Stek – prikaz upotrebe



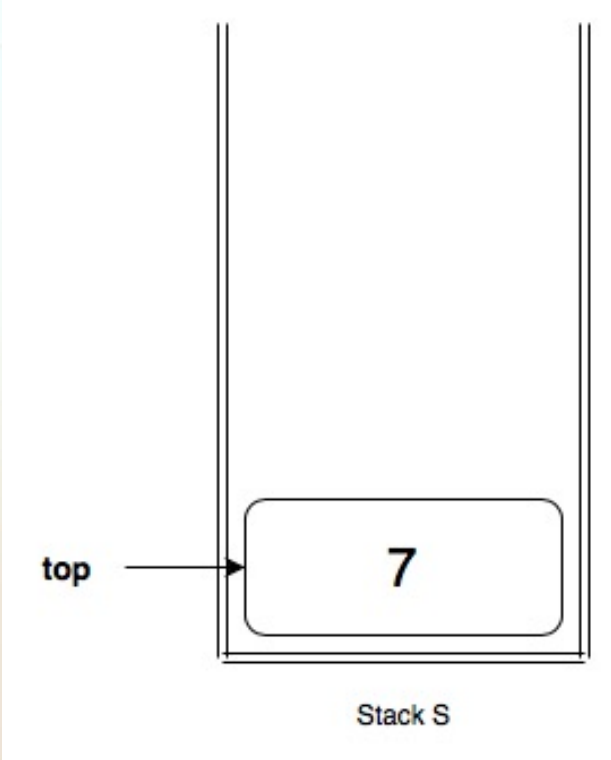
Stek – prikaz upotrebe



Stek – prikaz upotrebe



Stek – prikaz upotrebe



Red

- FIFO struktura
- Operacije:
  - `Q.enqueue(e)` – Dodavanje novog elementa *e* na kraj reda
  - `Q.dequeue()` – Destruktivno čitanje - uklanja se element na početku reda koji se vraća kao povratna vrednost metode
  - `Q.first()` – Čitanje prvog elementa reda
  - `Q.is_empty()` – Proverava da li je red prazan (rezultat tipa boolean)
  - `len(Q)` – Pronalazi i vraća broj elemenata reda *Q*

Dek

- **D**ouble-**e**nded **Q**ueue
- Operacije:
  - `D.add_first(e)` – Dodavanje novog elementa *e* na početak deka
  - `D.add_last(e)` – Dodavanje novog elementa *e* na kraj deka
  - `D.delete_first()` – Destruktivno čitanje prvog elementa - uklanja se element na početku deka koji se vraća kao povratna vrednost metode
  - `D.delete_last()` – Destruktivno čitanje poslednjeg elementa - uklanja se element na kraju deka koji se vraća kao povratna vrednost metode
  - `D.first()` – Čitanje prvog elementa deka
  - `D.last()` – Čitanje poslednjeg elementa deka
  - `D.is_empty()` – Proverava da li je dek prazan (rezultat tipa boolean)
  - `len(D)` – Pronalazi i vraća broj elemenata deka *D*

Zadatak 1

- Implementirati klasu **Stack**.
  - U prvoj implementaciji dozvoliti neograničen broj elemenata
  - Kreirati klasu LimitedStack čiji je broj elemenata ograničen (ograničenje se zadaje prilikom kreiranja). U slučaju da se pokuša dodavanje elementa u pun stek baca se FullStackException.

Zadatak 2

- Napisati program za konverziju pozitivnog celog broja zapisanog u dekadnom sistemu u string u zadatoj brojnoj osnovi uz upotrebu structure podataka Stack.

- ***Napomena:*** Brojne osnove mogu imati vrednost od 2 do 16.

Zadatak 2 - Primer

- 734 u dekadnom brojnom zapisu pretvaramo u brojnu osnovu 16.
- Cifre koje su dostupne su 0123456789ABCDEF.
- Postupak:
  - Delimo broj sa brojnom osnovom 16.
  - 734 | 16 = 45 (ostatak 14, odnosno E) – E je poslednja cifra.
  - 45 | 16 = 2 (ostatak 13, odnosno D) - D je pretposlednja cifra.
  - 2 | 16 = 0 (ostatak 2) – 2 je treća cifra od kraja, odnosno prva.
  - U rešenju cifre treba da se obrnu – 2DE. Zato koristimo stack.

Zadatak 3

- Implementirati klasu **Queue.**
  - U prvoj implementaciji, dozvolite neograničen broj elemenata
  - Prepraviti prvu implementaciju tako da broj elemenata bude ograničen na N. U cilju efikasnijeg upravljanja memorijskim prostorom, implementirati cirkularno smeštanje elemenata.

Zadatak 4

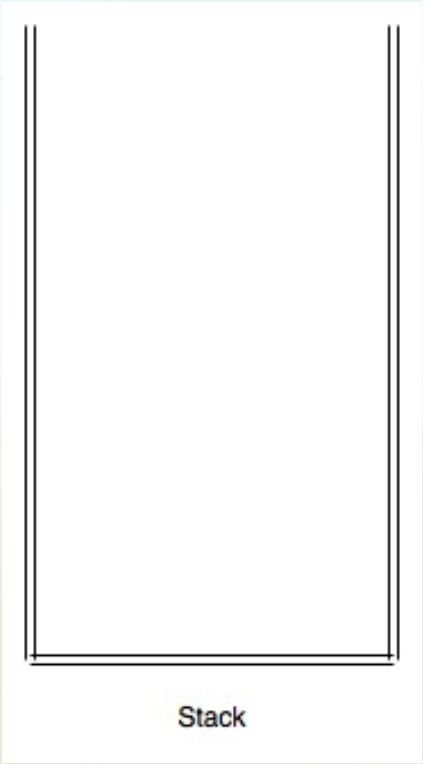
- Implementirati klasu **Deque.**

Zadatak 5

- Napisati program za konverziju izraza iz infiksne u postfiksnu notaciju uz upotrebu implementirane klase *Stack*.
- Primeri:
  a) 3 + 2 → 3 2 +
  b) 4 * 3 + 9 → 4 3 * 9 +
  c) 7 – 3 * 2 → 7 3 2 * -
- ***Napomene***:
  - Dozvoljeni operandi su cifre.
  - Dozvoljeni operatori su +, -, *, /
- Razmislite na koji način bismo mogli podržati grupisanje operanada pomoću zagrada

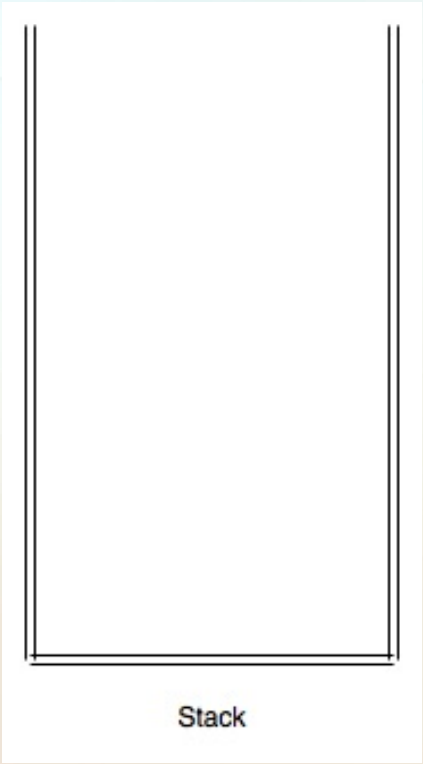
Zadatak 5a)

Izraz: 3 + 2
Izlaz:



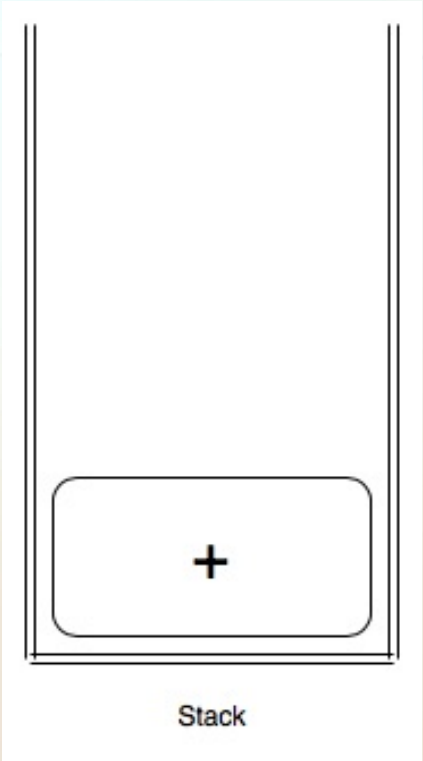
Zadatak 5a)

Izraz: 3 + 2
Izlaz: 3



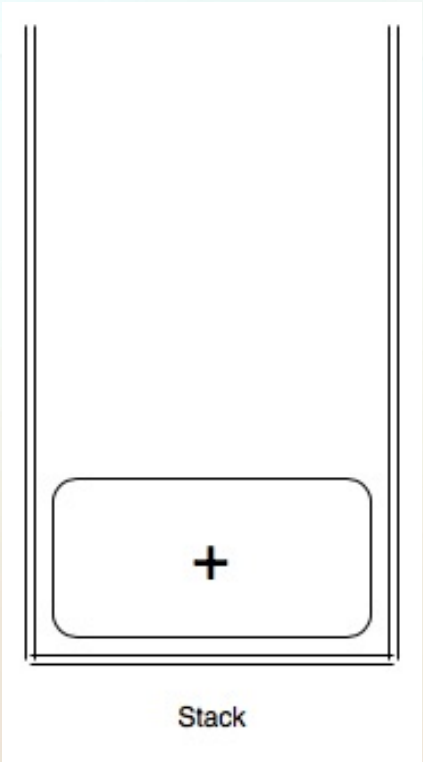
Zadatak 5a)

Izraz: 3 + 2
Izlaz: 3



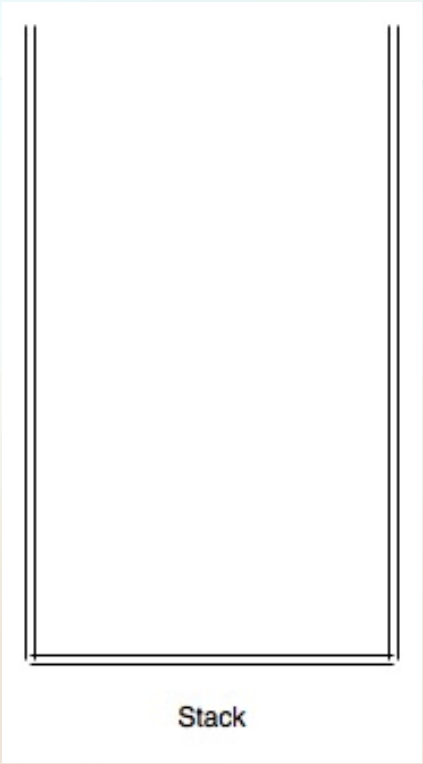
Zadatak 5a)

Izraz: 3 + 2
Izlaz: 3 2



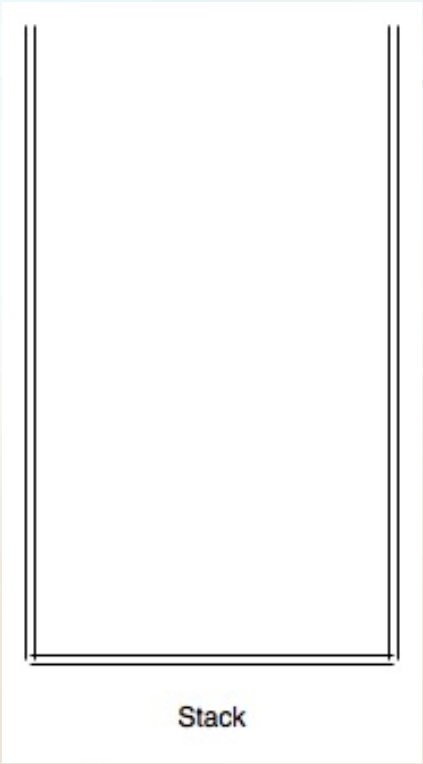
Zadatak 5a)

Izraz: 3 + 2
Izlaz: 3 2 +



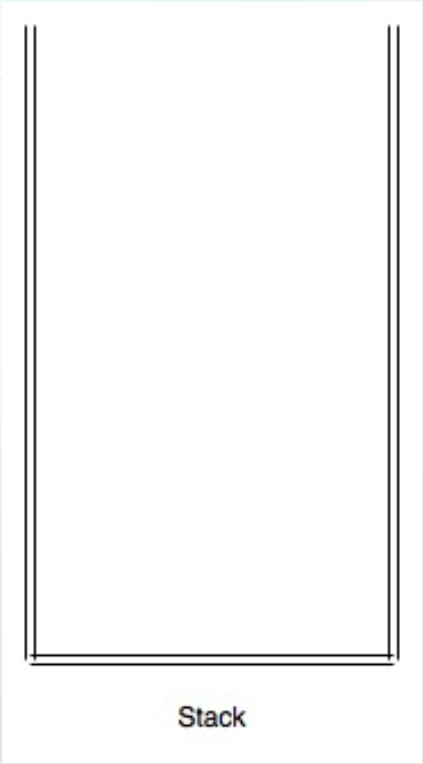
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz:



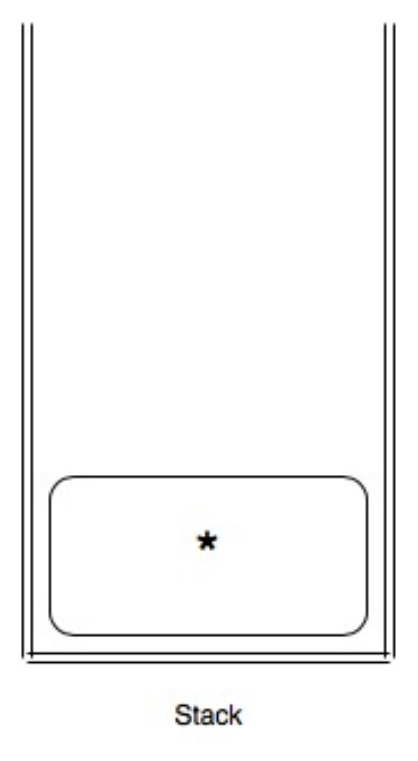
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4



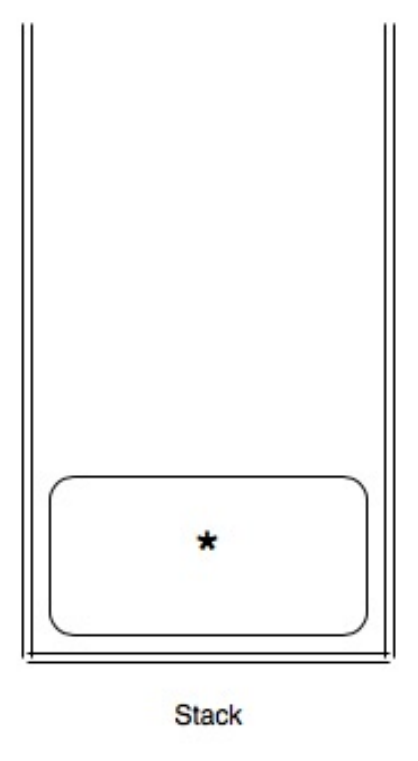
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4



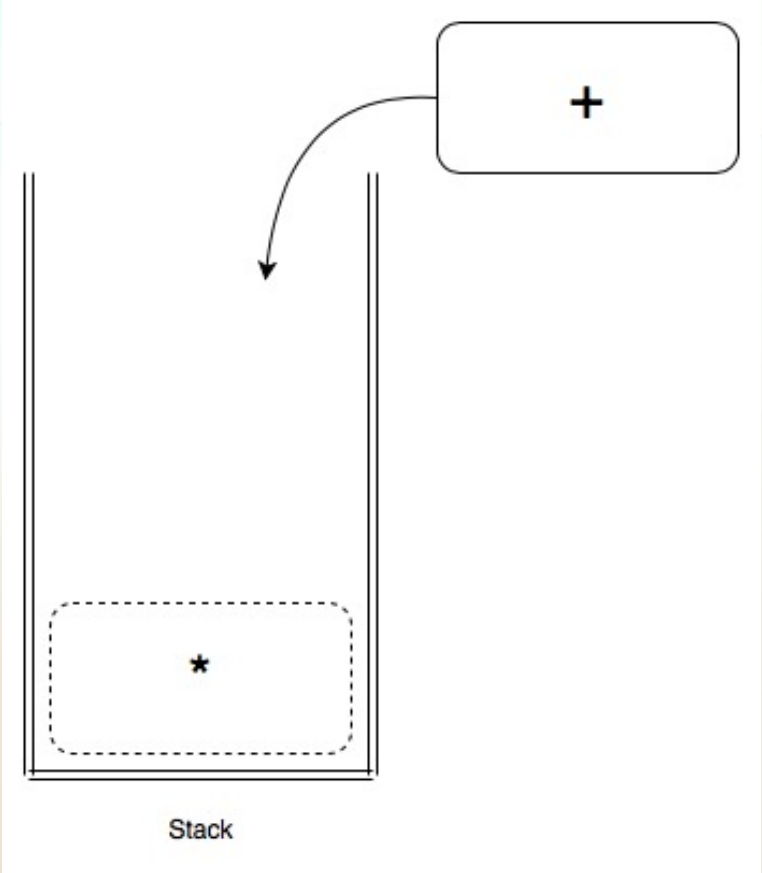
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4 3



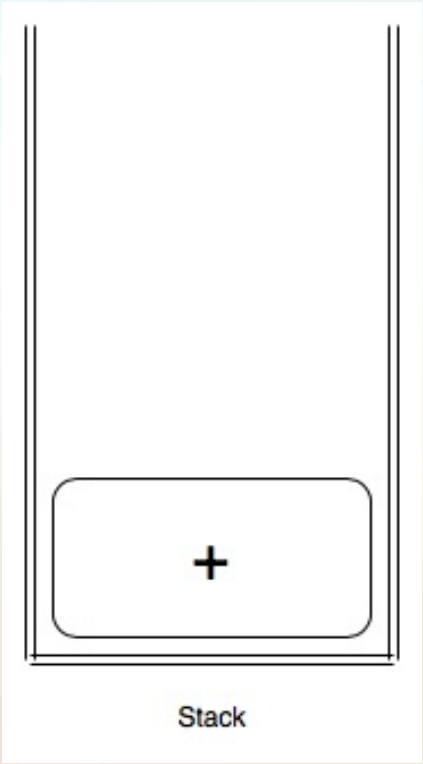
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4 3



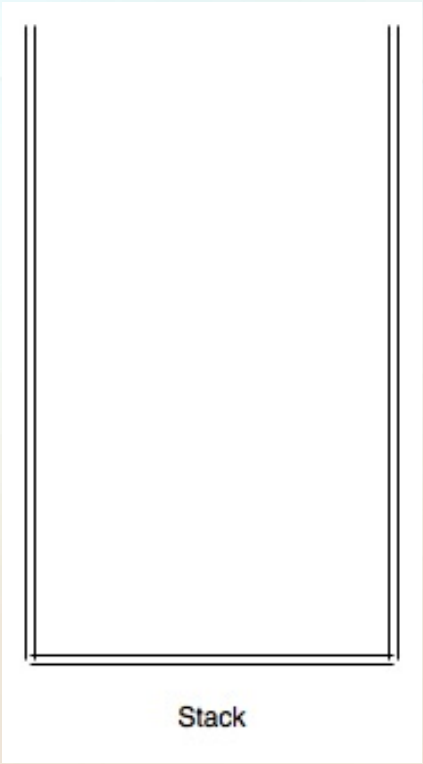
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4 3 * 9



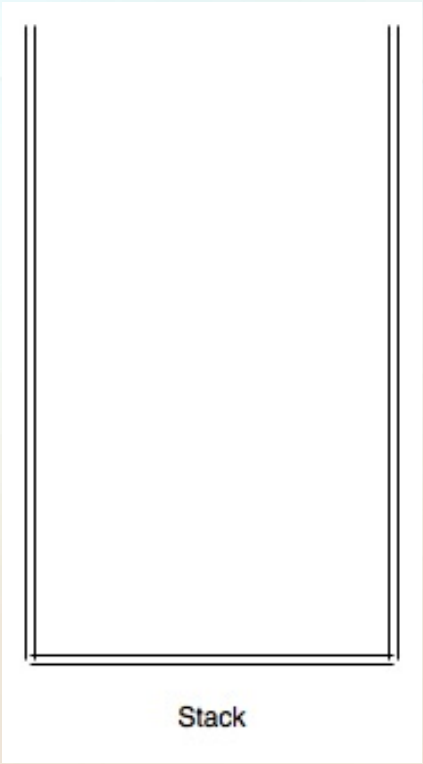
Zadatak 5b)

Izraz: 4 * 3 + 9
Izlaz: 4 3 * 9 +



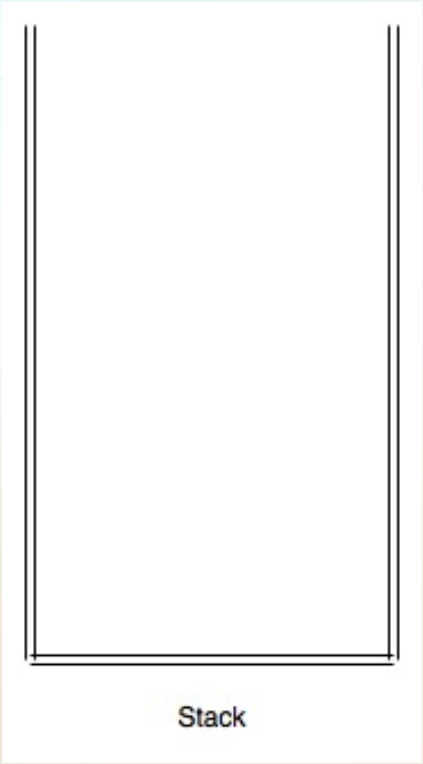
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz:



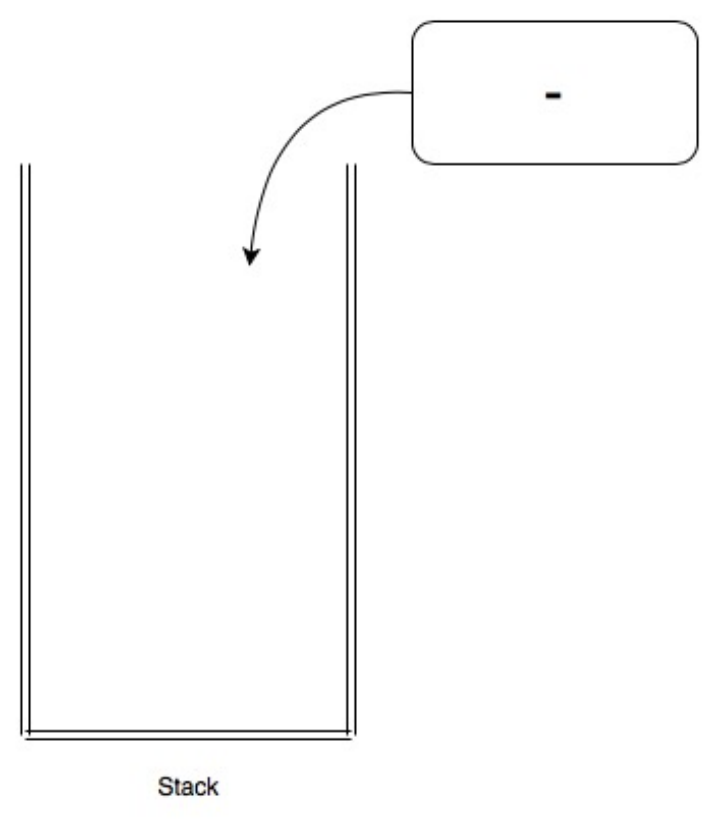
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7



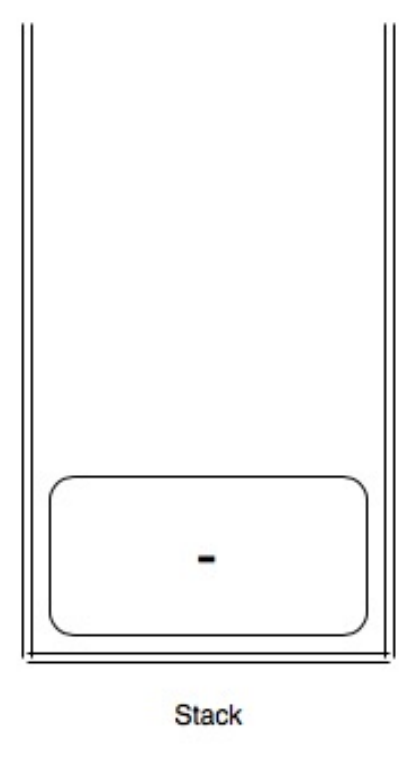
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7



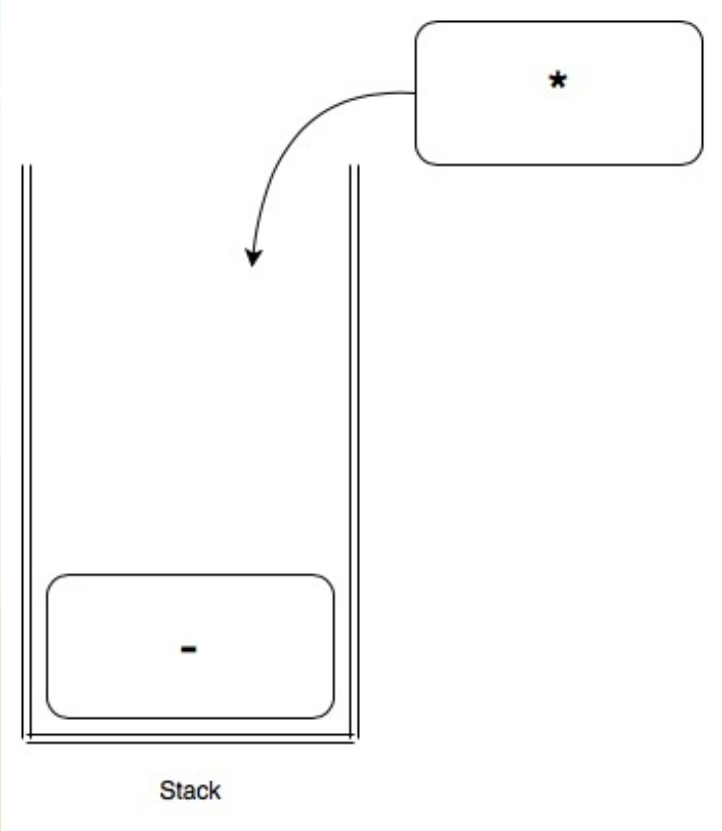
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7 3



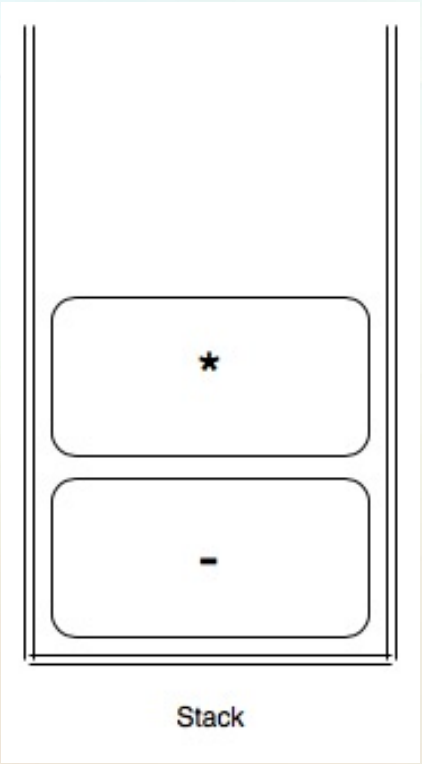
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7 3



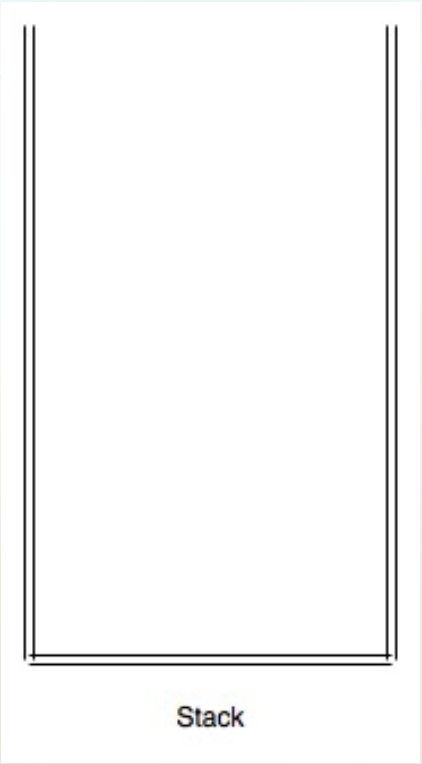
Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7 3 2



Zadatak 5c)

Izraz: 7 – 3 * 2
Izlaz: 7 3 2 * -



Zadatak 5

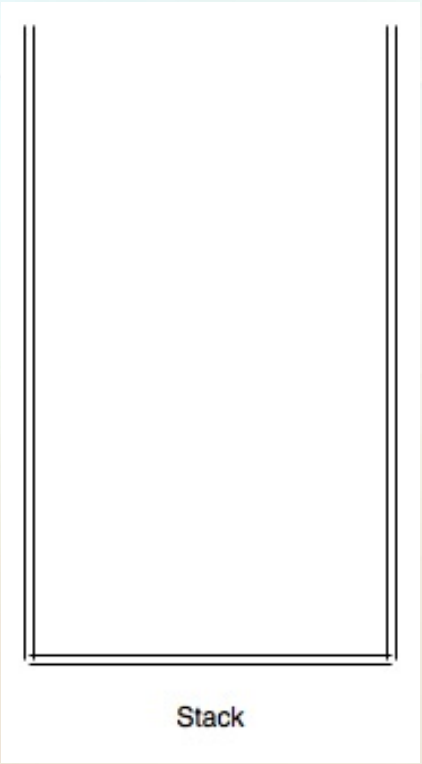
- Razmisliti:
  - Šta bi trebalo da se desi u slučaju da na stek dolazi operator istog prioriteta?
    - 3 + 4 - 2 → 3 4 + 2 –
  - Šta ćemo sa zagradama?
    - (2 + 3) * 7 → 2 3 + 7 *
    - 8 - (4 * (1 + 3) / 2 - 11) → 8 4 1 3 + * 2 / 11 - -

Zadatak 6

- Napisati program za izračunavanje vrednosti izraza zapisanog u postfiksnoj notaciji uz upotrebu implementirane klase *Stack*.
- Primeri:
  a) 3 2 * → 6
  b) 4 3 * 9 + → 21
  c) 7 3 2 * - → 1
- ***Napomene***:
  - Dozvoljeni operandi su cifre.
  - Dozvoljeni operatori su +, -, *, /

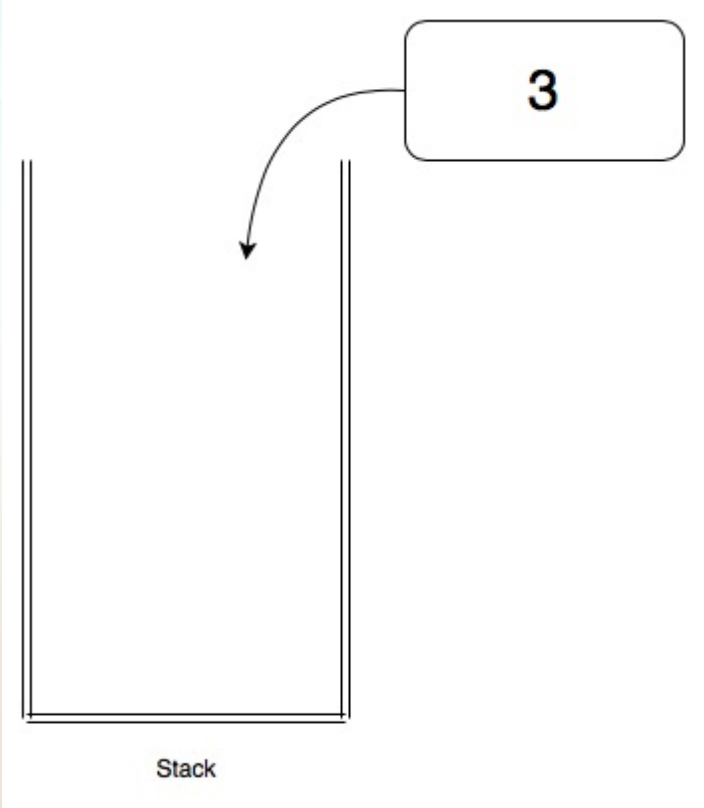
Zadatak 6a)

Izraz: 3 2 *



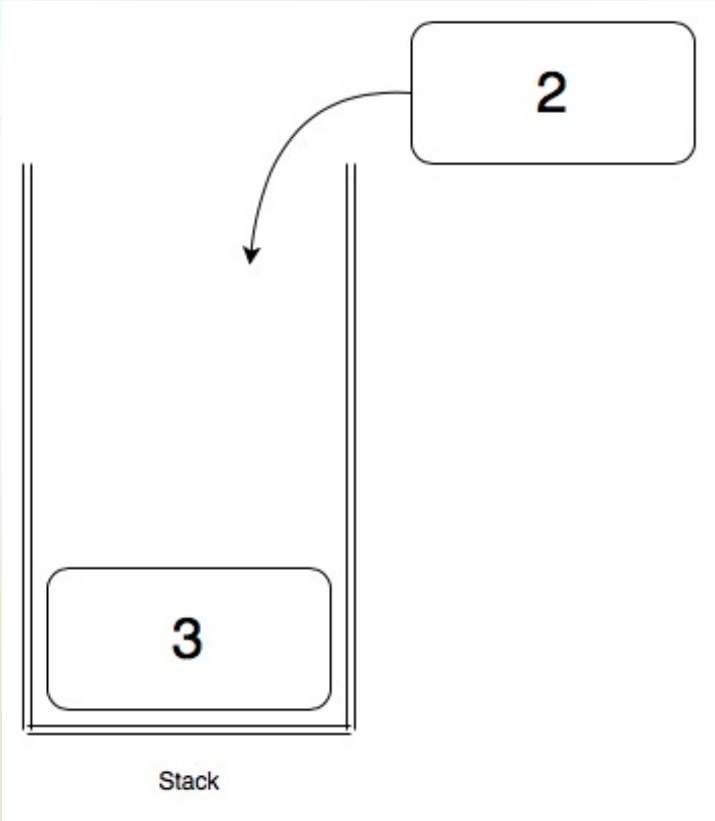
Zadatak 6a)

Izraz: 3 2 *



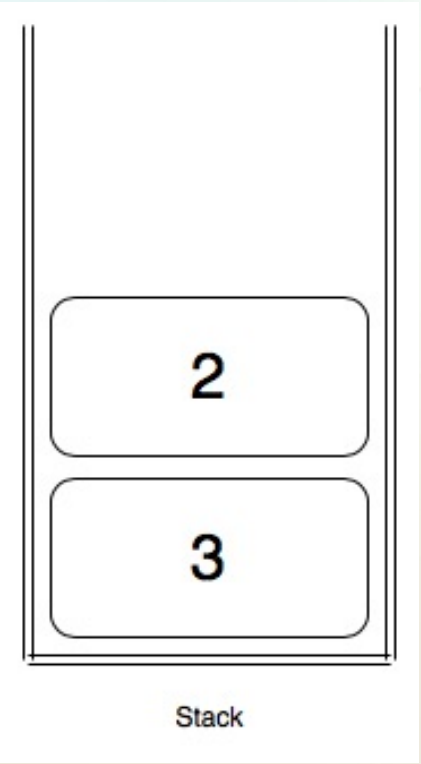
Zadatak 6a)

Izraz: 3 2 *



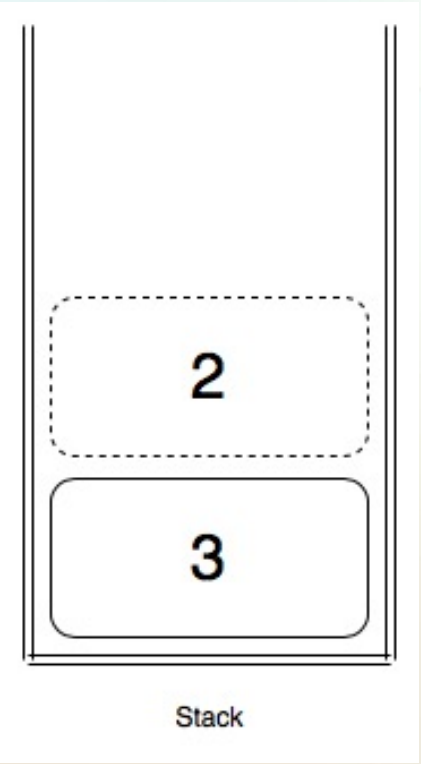
Zadatak 6a)

Izraz: 3 2 *



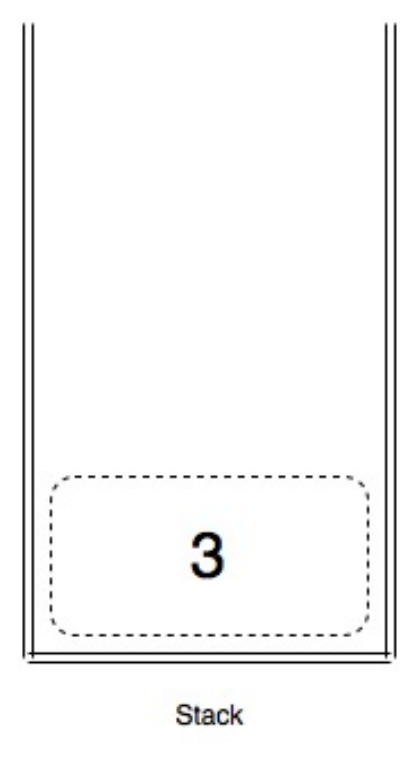
Zadatak 6a)

Izraz: 3 2 *



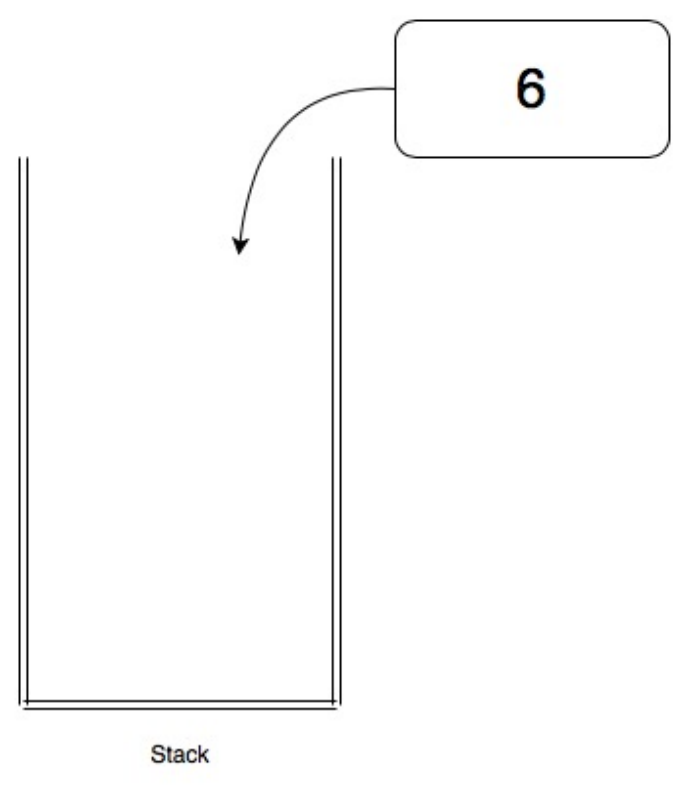
Zadatak 6a)

Izraz: 3 2 *



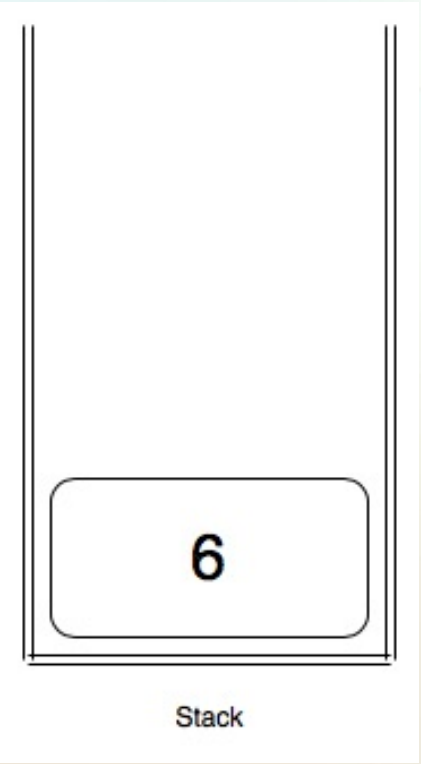
Zadatak 6a)

Izraz: 3 2 *



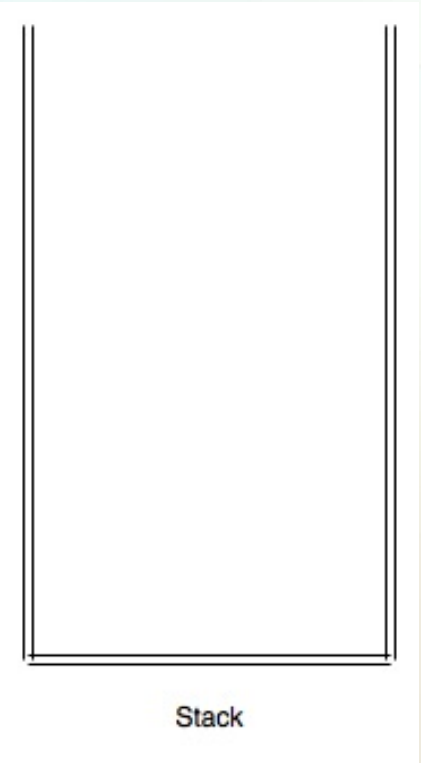
Zadatak 6a)

Izraz: 3 2 *



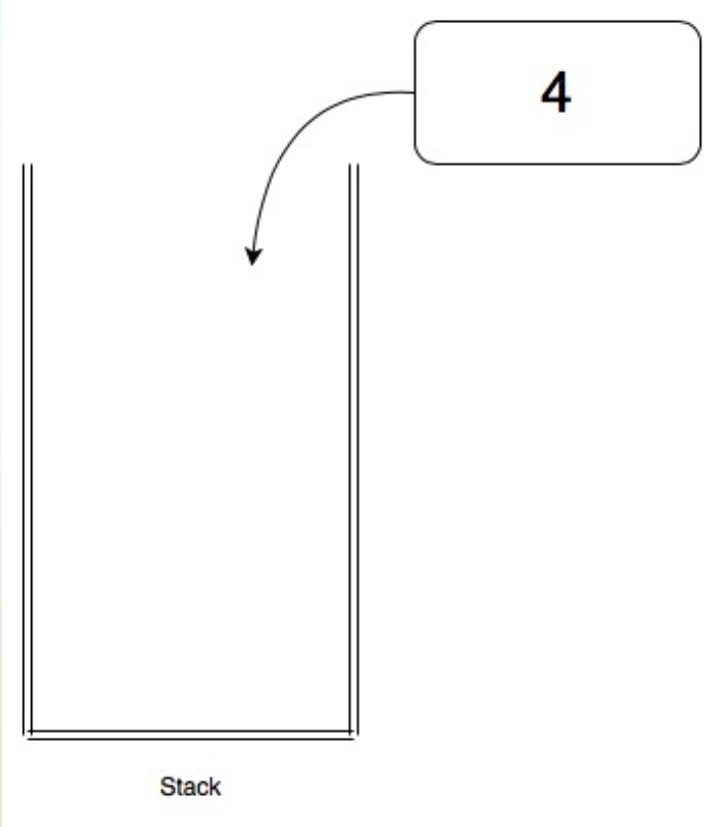
Zadatak 6b)

Izraz: 4 3 * 9 +



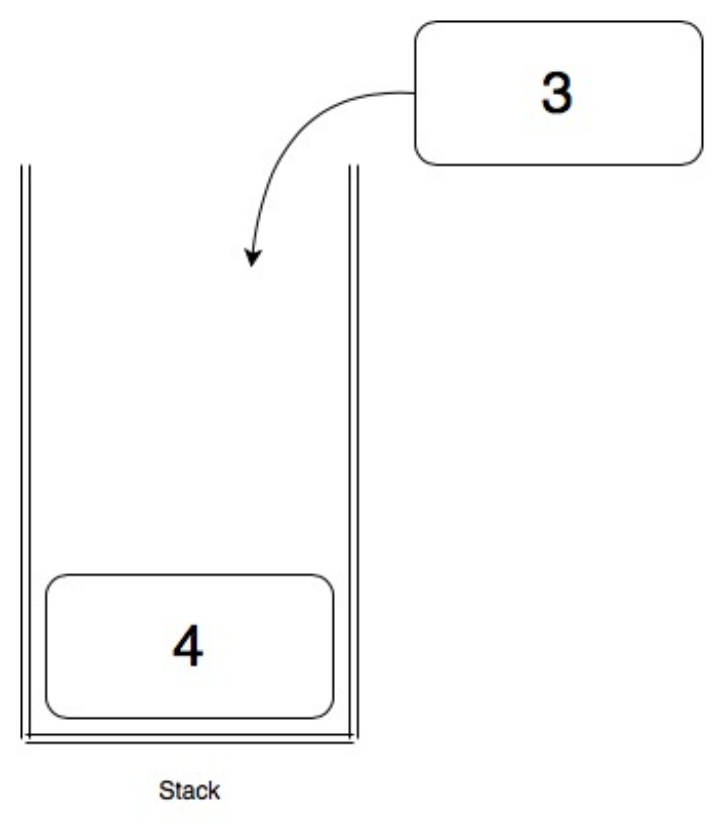
Zadatak 6b)

Izraz: 4 3 * 9 +



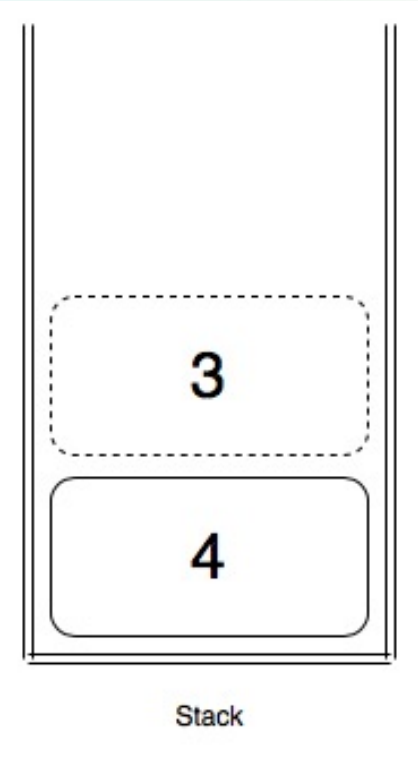
Zadatak 6b)

Izraz: 4 3 * 9 +



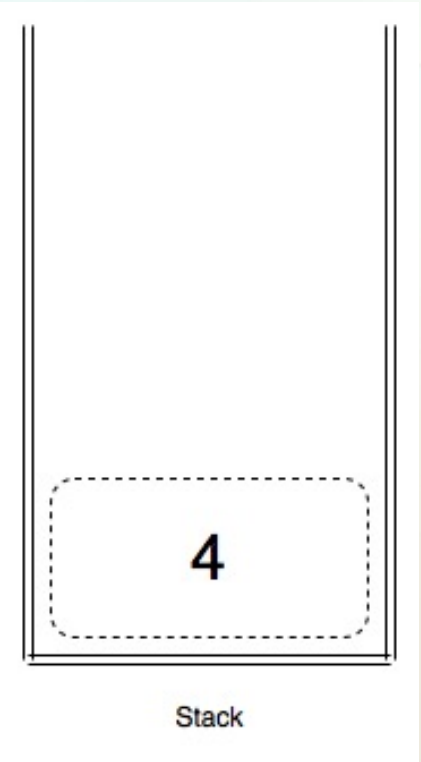
Zadatak 6b)

Izraz: 4 3 * 9 +



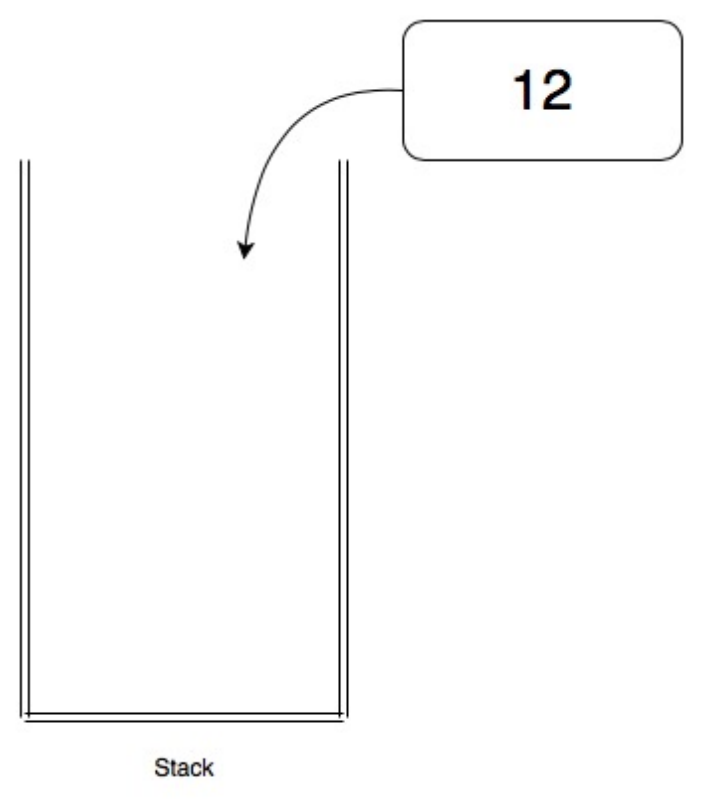
Zadatak 6b)

Izraz: 4 3 * 9 +



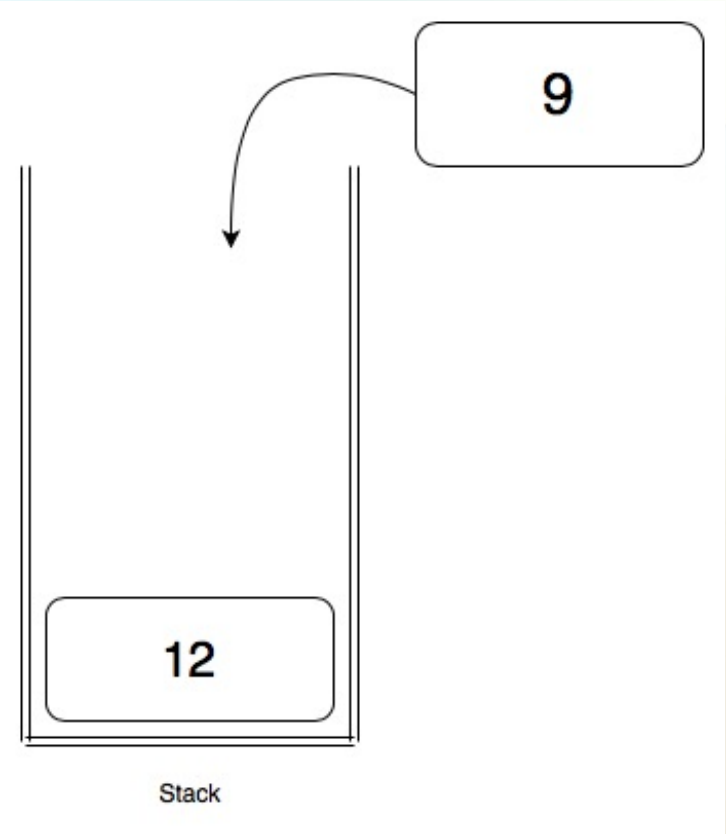
Zadatak 6b)

Izraz: 4 3 * 9 +



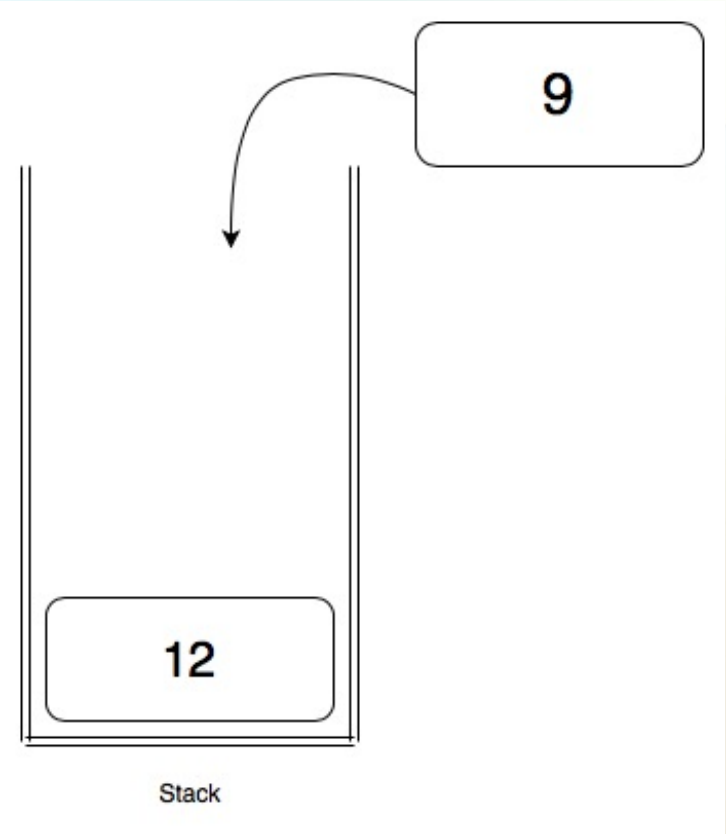
Zadatak 6b)

Izraz: 4 3 * 9 +



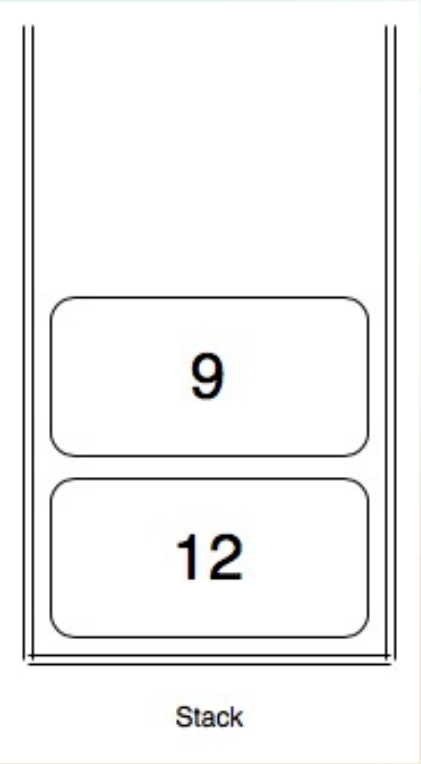
Zadatak 6b)

Izraz: 4 3 * 9 +



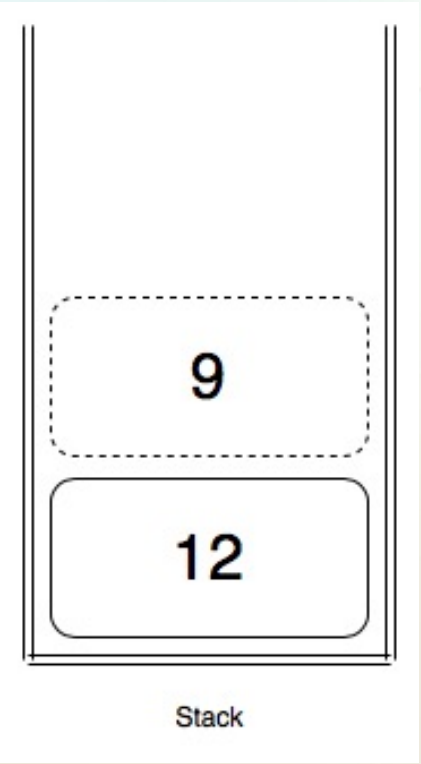
Zadatak 6b)

Izraz: 4 3 * 9 +



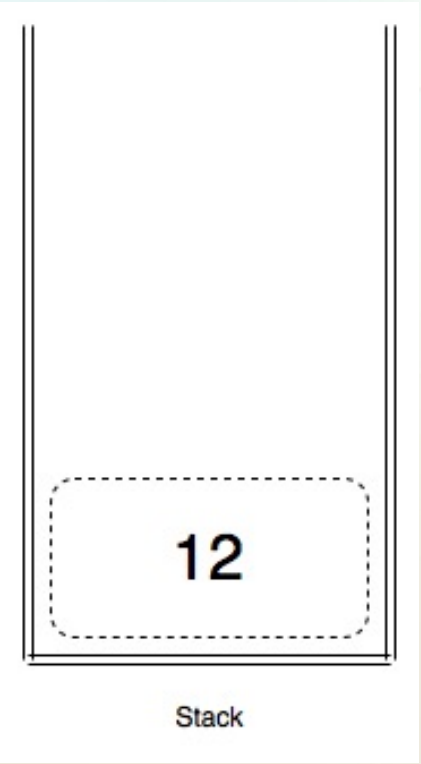
Zadatak 6b)

Izraz: 4 3 * 9 +



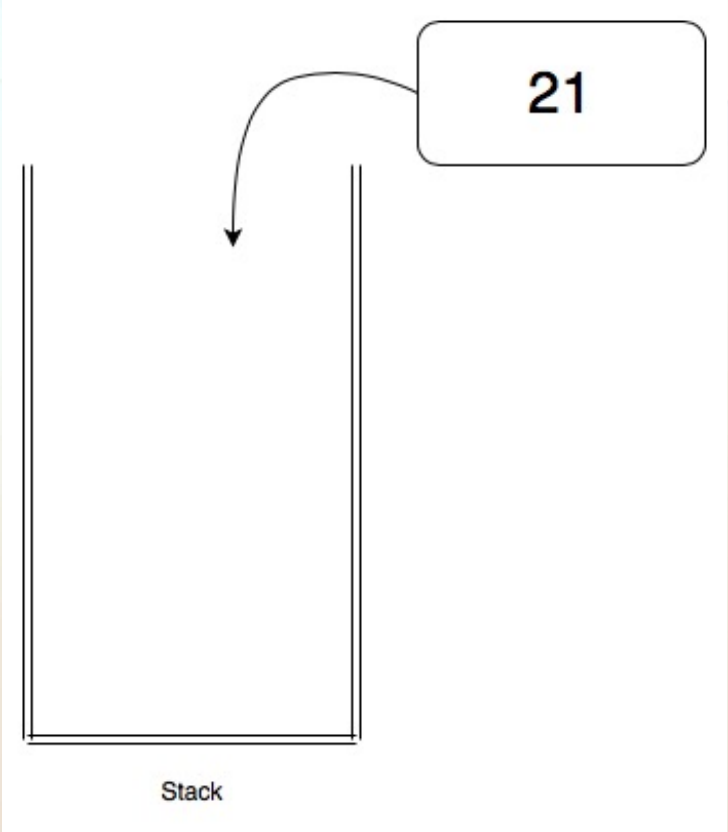
Zadatak 6b)

Izraz: 4 3 * 9 +



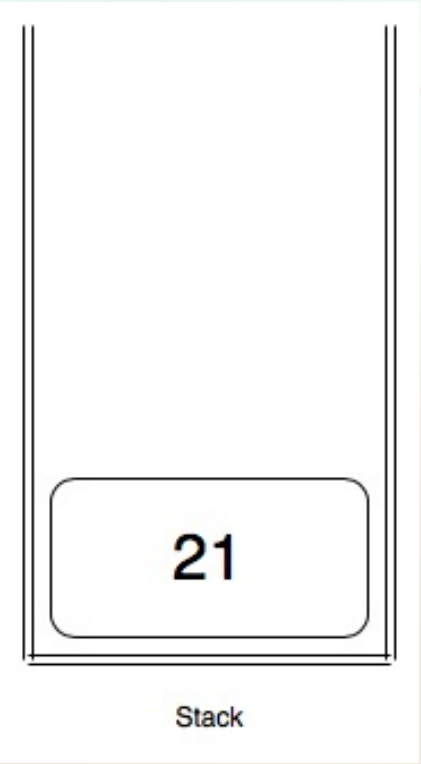
Zadatak 6b)

Izraz: 4 3 * 9 +



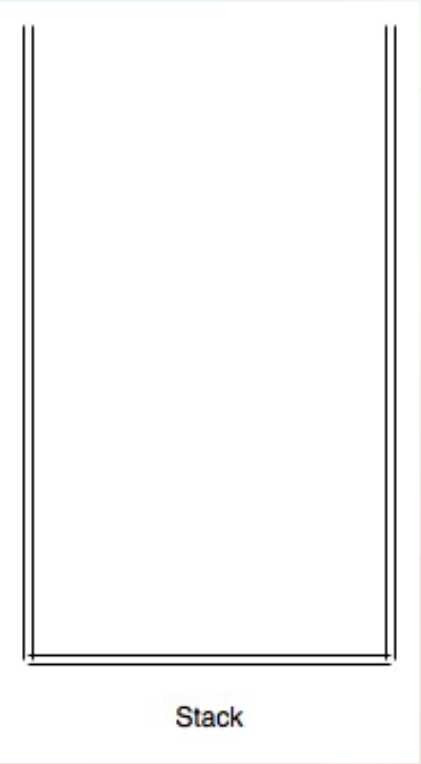
Zadatak 6b)

Izraz: 4 3 * 9 +



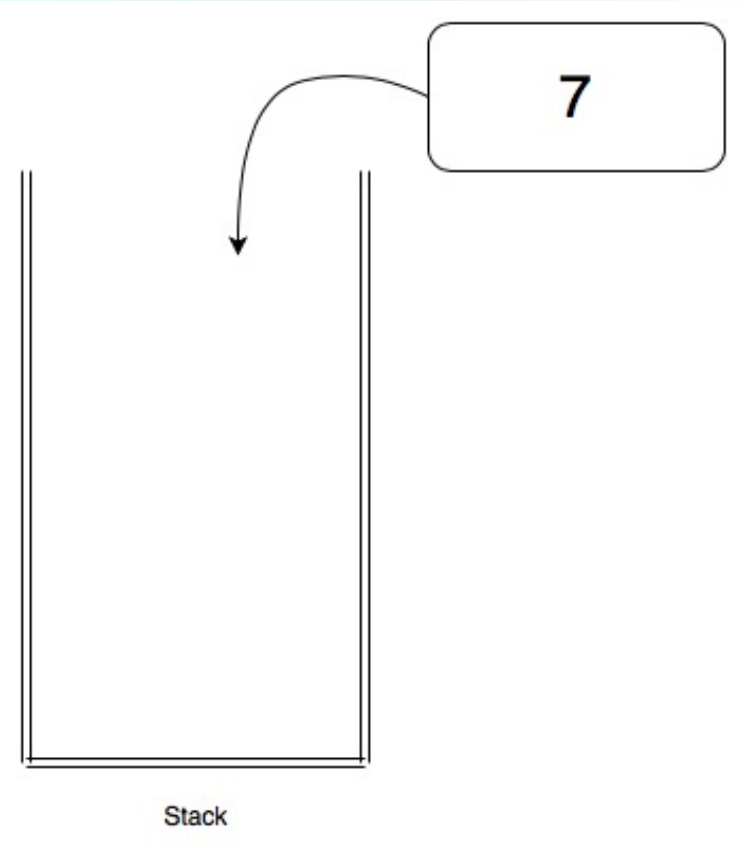
Zadatak 6c)

Izraz: 7 3 2 * -



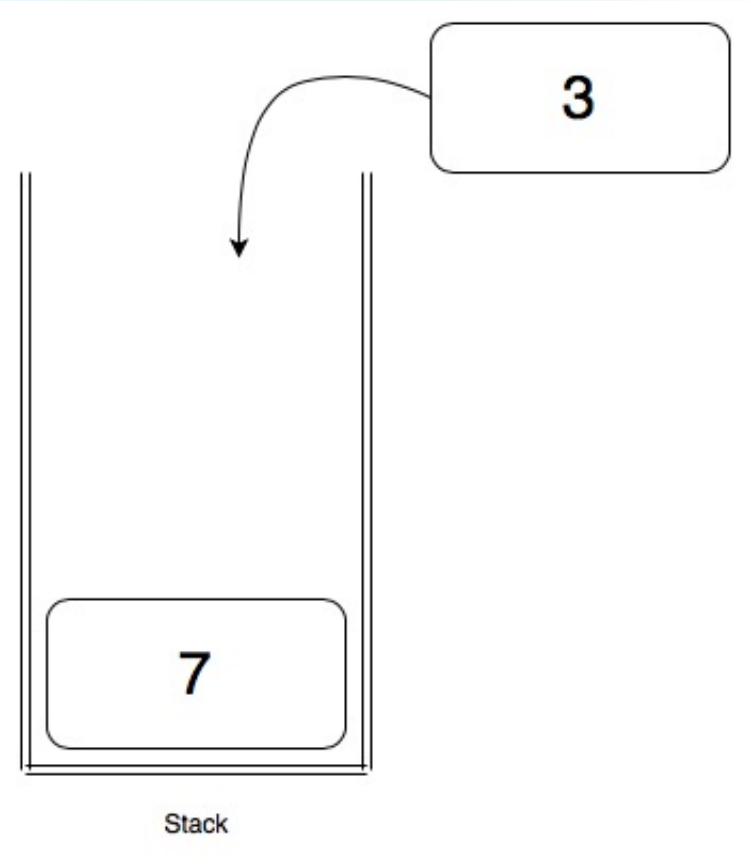
Zadatak 6c)

Izraz: 7 3 2 * -



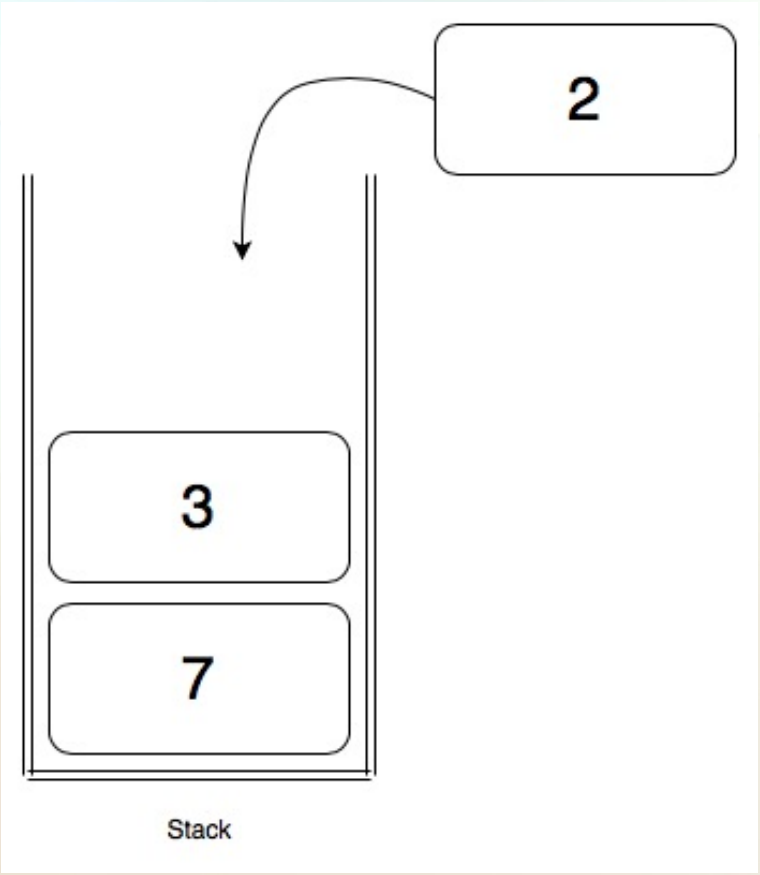
Zadatak 6c)

Izraz: 7 3 2 * -



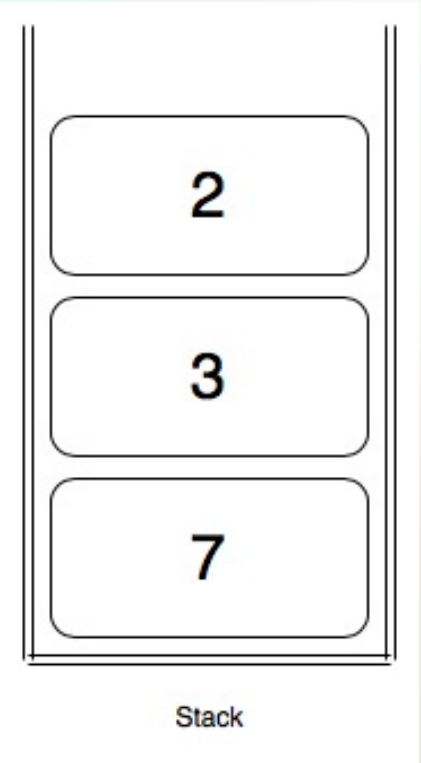
Zadatak 6c)

Izraz: 7 3 2 * -



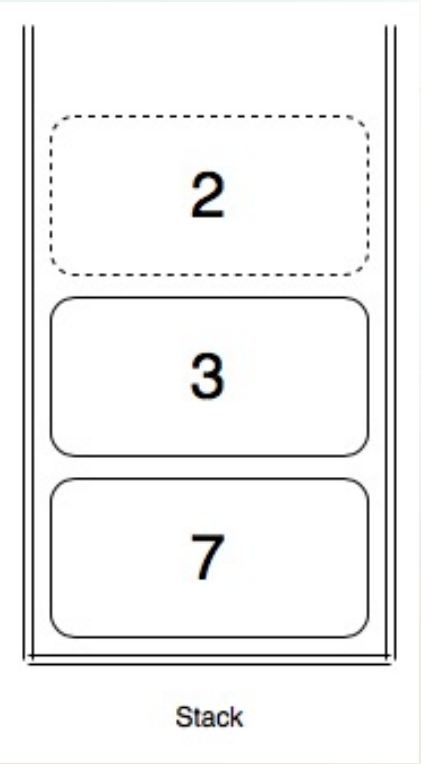
Zadatak 6c)

Izraz: 7 3 2 * -



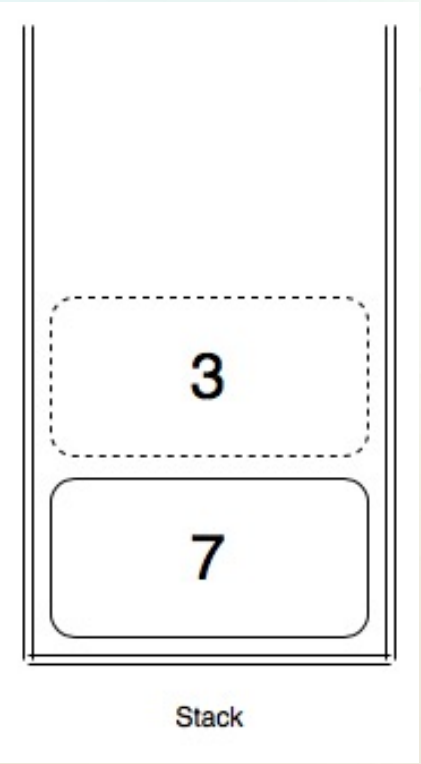
Zadatak 6c)

Izraz: 7 3 2 * -



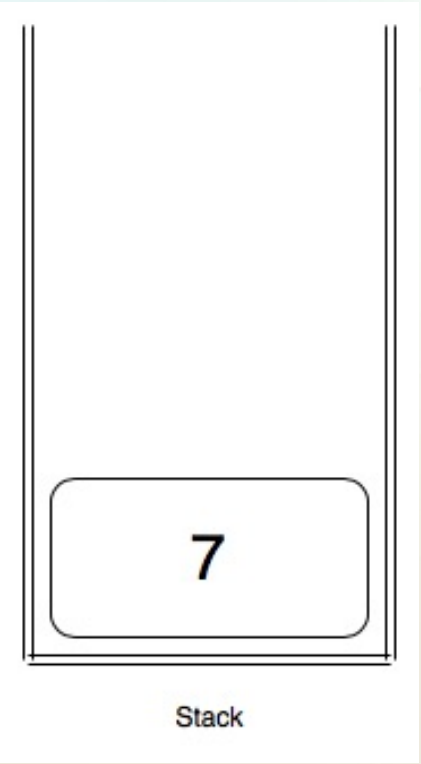
Zadatak 6c)

Izraz: 7 3 2 * -



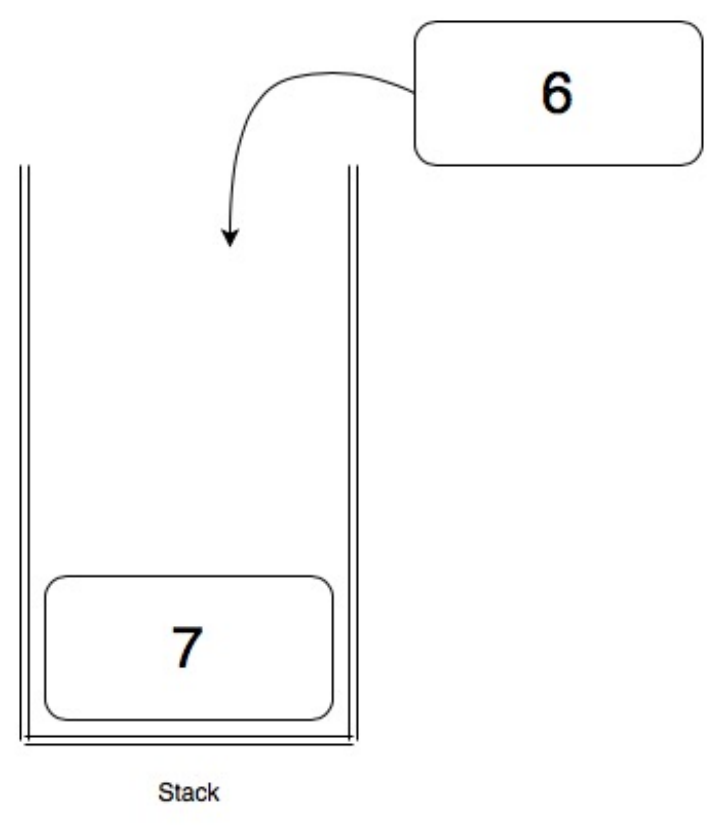
Zadatak 6c)

Izraz: 7 3 2 * -



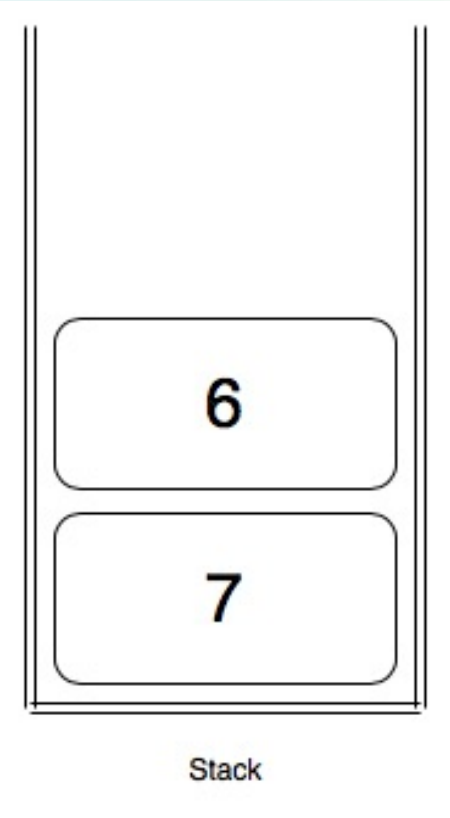
Zadatak 6c)

Izraz: 7 3 2 * -



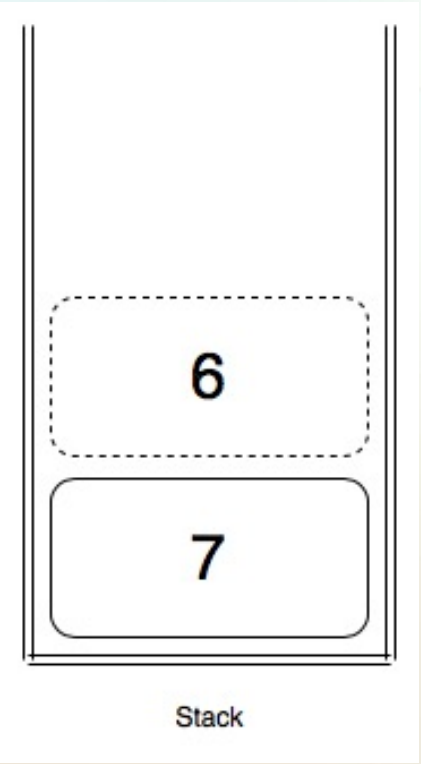
Zadatak 6c)

Izraz: 7 3 2 * -



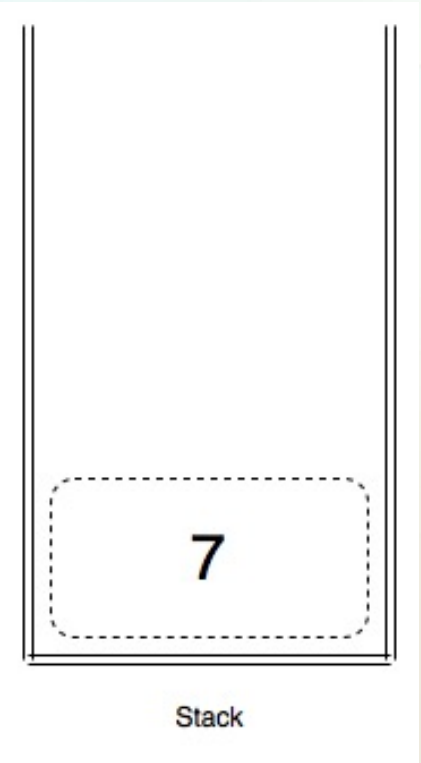
Zadatak 6c)

Izraz: 7 3 2 * -



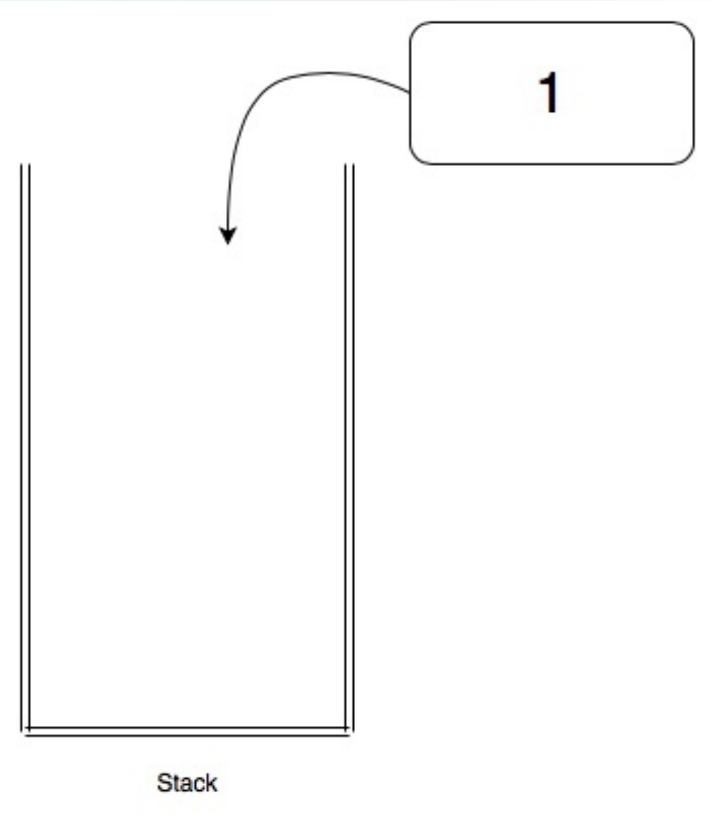
Zadatak 6c)

Izraz: 7 3 2 * -



Zadatak 6c)

Izraz: 7 3 2 * -



Zadatak 6c)

Izraz: 7 3 2 * -