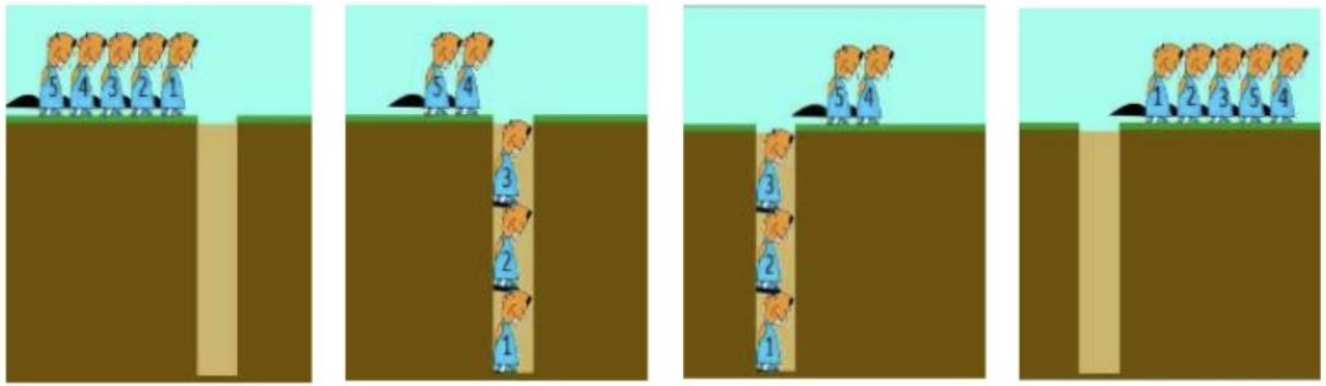
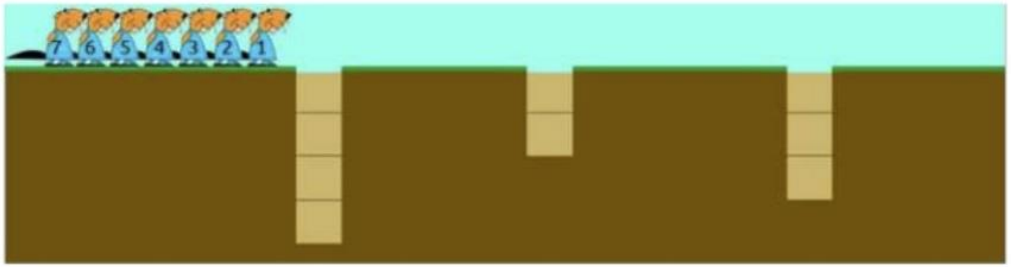
Primjer 12. Zečje rupe

Dabrovi su krenuli u šetnju po šumi i kreću se u redu, jedan za drugim. Međutim, zečevi su u šumi napravili rupe, baš na putu na kome se dabrovi šetaju. Rupe su dovoljno duboke da nekoliko dabrova može ući u njih. Kada se rupa napuni dabrovima, ostali dabrovi, koji su bili iza njih u redu, prolaze preko rupe, a zatim izlaze dabrovi koji su ušli u rupu. Na primjer, ako imamo pet dabrova označenih brojevima: 1, 2, 3, 4 i 5 koji naiđu na rupu u koju može upasti 3 dabra, imamo sljedeću situaciju (dabar 1 je prvi u redu, a dabar 5 je posljednji):



Dabar, prezentacija, Lidija Kralj, 2016
https://www.slideshare.net/LidijaKralj/dabar-prva-prezentacija

U redu je sedam dabrova (počevši od broja 1 pa do broja 7). U prvu rupu na putu mogu ući 4 dabra, zatim nailaze na rupu u koju mogu ući dva dabra, i na kraju nailaze na rupu u koju ulaze 3 dabra. Koji je redoslijed dabrova nakon treće rupe?



A. 3216574

B. 7435612

C. 1234756

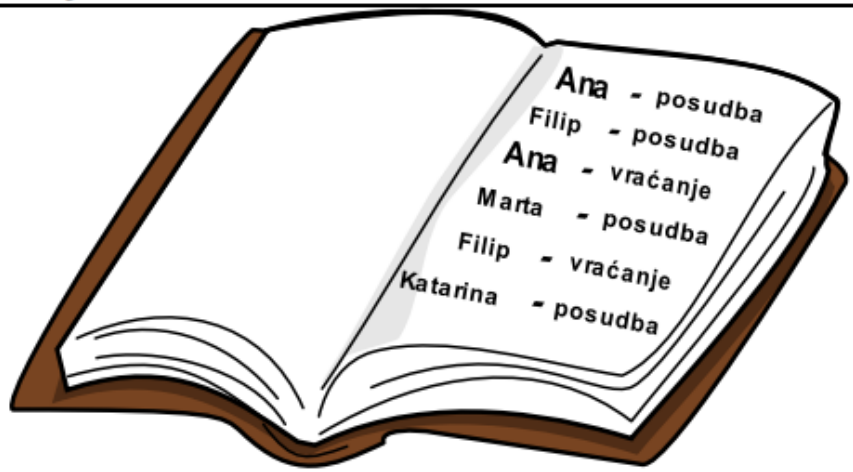
D. 734165

Dabar, prezentacija, Lidija Kralj, 2016
https://www.slideshare.net/LidijaKralj/dabar-prva-prezentacija

Knjižnica u Dabrogradu nema puno knjiga. Kad neki dabar želi posuditi knjigu, knjižničar zabilježi dabrovo ime i uzima knjigu s vrha hrpe knjiga. Kada dabar vrati knjigu, knjižničar zabilježi dabrovo ime i stavlja vraćenu knjigu na vrh hrpe knjiga.

Na početku tjedna hrpa knjiga bila je posložena kao na slici:

Knjižničareva knjiga posudbe pokazuje nam sljedeće informacije:



Koju je knjigu posudila Katarina?

Algoritmi i strukture podataka

- predavanja -

6. Stog

Stog

- struktura podataka kod koje se posljednji pohranjeni podatak prvi uzima u obradu (Last In First Out – LIFO)
- potrebne operacije:
  - dodavanje (*push*) elemenata na vrh stoga
  - brisanje (*pop*) elemenata s vrha stoga
  - inicijalizacija praznog stoga
  - po potrebi
    - uvid u sadržaj vrha stoga (*peek*)
- pojedina operacija *push* ili *pop* zahtijeva jednako vremena bez obzira na broj pohranjenih podataka
- može se realizirati statičkom strukturom podataka

  StackStatic.cpp
- najbrža realizacija: O(1)
- postoji mogućnost prepunjenja

Stog

- Realizacija dinamičkim poljem

  StackDynamic.cpp

- Vjerojatnost prepunjenja vrlo mala
- Može se dogoditi O(N)

Stog

- Realizacija jednostruko povezanom listom

  StackList.cpp
- Vjerojatnost prepunjenja vrlo mala
- Uvijek O(1)

- Primjer primjene stoga:
  - Evaluacija izraza napisanog u RPN

Evaluacija izraza u *postfix* notaciji

- izraz u *infix* notaciji:

8 * (9 +3) / 16 = 6

- može se pretvoriti u postfix notaciju (ili Reverse Polish Notation - RPN):

8 9 3 + * 16 /

- kod RPN notacije operandi se pišu prije operatora, a operator se primjenjuje na operande koji se nalaze točno ispred njega
- prednost postfix notacije - nemamo zagrade, jednostavna evaluacija izraza korištenjem stoga

Ručna pretvorba *infix* → *postfix*

- dodati sve implicitne zagrade u *infix* izraz:

8 * (9 + 3) / 16

((8 * (9 + 3)) / 16)

- krenuvši od unutrašnje zagrade, slijed operand – operator – operand zamijeniti u operand – operand – operator

((8 * (9 + 3)) / 16)

((8 * 9 3 +) / 16)

((8 * 9 3 +) / 16)

(8 9 3 + * / 16)

(8 9 3 + * / 16)

8 9 3 + * 16 /

- Strojni algoritam: Shunting Yard (kasnije, potreban *red*)

Evaluacija izraza u *postfix* notaciji

- algoritam evaluacije izraza u *postfix* notaciji korištenjem stoga:
  - izraz se evaluira s lijeva na desno, token po token
  - ako je token broj, stavi se na stog (*push*)
  - ako je token operator, skinu se zadnja dva broja sa stoga (*pop*), na njih se primijeni taj operator, a rezultat se stavlja natrag na stog (*push*)
- Primjer: 8 * (9 + 3) / 16 = 6 → 8 9 3 + * 16 /

Ulaz	8	9	3	+	*	16	/
			3				
		9	9	12		16	
Stog	8	8	8	8	96	96	6

RPN.cpp

Primjer: izravna evaluacija *infix* izraza

- Dva stoga:
  - stog operatora
  - stog vrijednosti

- Pojedinačna evaluacija:
  - uzmi operator sa stoga operatora
  - uzmi operande sa stoga vrijednosti
  - primijeni operator nad operandima
  - stavi rezultat na stog vrijednosti

InfixEvaluation.cpp

Primjer: izravna evaluacija *infix* izraza

- Algoritam:
  - dok ima znakova na ulazu
    - ako slijedi vrijednost, stavi je na stog vrijednosti
    - ako slijedi otvorena zagrada, stavi je na stog operatora
    - ako slijedi zatvorena zagrada
      - dok na vrhu stoga operatora nije otvorena zagrada
        - obavi pojedinačnu evaluaciju
      - skini otvorenu zagradu sa stoga operanada
    - ako slijedi operator
      - dok ima operatora na stogu koji su većeg ili jednakog prioriteta
        - obavi pojedinačnu evaluaciju
      - stavi novi operator na stog operatora
  - dok ima operatora na stogu
    - obavi pojedinačnu evaluaciju
  - uzmi rezultat sa stoga vrijednosti

InfixEvaluation.cpp