Stack

Problem. Potrebno je simulirati sledeće poteze sa kartama. U ruci imamo ceo špil karata dok na stolu na početnu ne stoji ni jedna karta. U jednom trenutku možemo ili da uzmemo bilo koju kartu iz špila i da je stavimo na vrh trenutne gomile na stolu (na početku je gomila prazna) ili da pogledamo vrednost karte koja je na vrhu gomile i da je sklonimo sa gomile.

Analiziranjem ovog problema možemo da zaključimo da su nam dovoljne 3 metode, od toga jedna treba da ubacuje element u skup, druga da odgovara koji je element poslednji ubačen u skup, i treća da izbacuje iz skupa element koji je poslednji ubačen.

Struktura stek rešava ovaj problem. Pogledajmo kako možemo da realizujemo pomenute 3 metode. Prvo ćemo realizovati stek preko niza, da bismo kasnije to uradili korišćenjem listi.

Na početku napravimo prazan niz $stack$. Ukoliko se od nas traži da ubacimo element u skup, dodaćemo ga na kraj niza $stack$. Zbog načina na koji ubacujemo elemente u nizu, poslednji ubačen element u skupu se nalazi na kraju niza $stack$. Da bismo odgovorili koji je poslednji element ubačen u skup, dovoljno je da pogledamo poslednji element u nizu $stack$. Izbacivanje iz skupa se vrši tako što obrišemo poslednji element u nizu $stack$.

Sledi implementacija pomenutih metoda. Koristićemo jedan brojač koji će da pokazuje na poslednji element u nizu. Dodavanjem/oduzimanjem jedinice od brojača ćemo simulirati ubacivanje i izbacivanje iz skupa, tj. kad ubacujemo element u skup mi ćemo povećati brojač za 1 i na to novo mesto staviti novi element. Dok kod brisanja ćemo simulirati izbacivanje iz niza, tako što ćemo brojač da smanjimo za 1.

```
================================================================
01    // ubacivanje u skup
02    function Push( ref stack, ref lastElement, newElement )
03          if ( lastElement = SizeOfArray - 1 ) then
04                return ERROR
05          end if
06          lastElement = lastElement + 1
07          stack[ lastElement ] = newElement
08    end function
09
10    // funkcija vraća element iz skupa koji je poslednji ubačen
11    function Seek( stack, lastElement )
12          if ( lastElement = 0 ) then
13                return ERROR_EMPTY
14          end if
15          return stack[ lastElement ]
16    end function
17
18    // izbacivanje iz skupa
19    function Pop( stack, ref lastElement )
20          if ( lastElement = 0 ) then
21                return ERROR_EMPTY
22          end if
23          lastElement = lastElement - 1
24    end function
```

```
25
26      // proverava da li je skup prazan
27      function Empty( stack, lastElement )
28              if ( lastElement = 0 ) then
29                      return true
30              end if
31              return false
32      end function
33
34      // inicijalizuje stek
35      function Init( stack, ref lastElement )
36              lastElement = 0
37      end function
================================================================
```

Primetimo da sve metode imaju vremensku složenost $O(1)$.

Koristeći niz mi smo se ograničili na broj elemenata koji mogu biti u skupu, međutim ovaj problem možemo da rešimo tako što ćemo koristiti listu umesto niza. Kako je nama potreban samo element koji je poslednji ubačen u skup, tj. poslednji element u nizu $stack$, pravićemo listu tako da je taj element uvek na početku liste. Ukoliko ubacivanje elementa u skup simuliramo ubacivanje elementa na početak liste, sve operacije će i dalje imati vremensku složenost $O(1)$.

Queue

Ovde ćemo kao i kod steka krenuti od analiziranja problema koji je potrebno rešiti.

Problem. Potrebno je simulirati red u prodavnici. Mogući događaju su da nova mušterija stane na kraj reda, da naplatimo mušteriji koja je prva u redu, i da ta mušterija koja je platila ode.

Analiziranjem problema možemo doći do zaključka da su nam i ovde potrebne 3 metode. Prva metoda treba da ubacuje element u skup, druga da izbacuje iz skupa element koji je prvi ubačen ukoliko posmatramo samo elemente koji se trenutno nalaze u skupu, dok treća treba da odgovara na pitanje koji je element od trenutnih u skupu prvi bio ubačen.

Koristeći strukturu queue možemo rešiti ovaj problem. Ovde ćemo isto kao kod steka prvo koristiti niz pri implementaciji, da bismo posle objasnili kako da ovaj problem rešimo koristeći listu.

Kao i kod steka, ovde ćemo početi sa praznim nizom $queue$. Ubacivanje elementa u skup ćemo simulirati ubacivanjem tog elementa na kraj niza $queue$. Zbog načina na koji ubacujemo elemente u skup, element koji je prvi ubačen u skup se nalazi na početku niza $queue$, te onda možemo da odgovorimo na pitanje koji je prvi element ubačen u skup ukoliko posmatramo samo elemente koji su trenutnu u skupu. Brisanje odgovarajućeg elementa radimo tako što obrišemo prvi element u nizu $queue$.

Sledi implementacija opisanih metoda, kao i nekih pomoćnih. Korišćenjem dva brojača koji pokazuju na prvi i poslednji element u nizu, te uvećavanjem oba za 1 pri ubacivanju/izbacivanju elementa iz niza ćemo simulirati ubacivanje/izbacivanje iz skupa.

```
================================================================
01     // ubacivanje elementa
02     function Push( ref queue, firstElement, ref lastElement, newElement )
03           if ( lastElement = SizeOfArray + 1 ) then
04                 return ERROR
05           end if
06           queue[ lastElement ] = newElement
07           lastElement = lastElement + 1
08     end function
09
10     // vraća prvi element u skupu
11     function Seek( queue, firstElement, lastElement )
12           if ( firstElement = lastElement ) then
13                 return ERROR_EMPTY
14           end if
15           return queue[ firstElement ]
16     end function
17
18     // izbacivanje prvog elementa iz skupa
19     function Pop( queue, ref firstElement, lastElement )
20           if ( firstElement = lastElement ) then
21                 return ERROR_EMPTY
22           end if
23           firstElement = firstElement + 1
24     end function
25
26     // proveravanje da li je skup prazan
27     function Empty( queue, firstElement, lastElement )
28           if ( firstElement = lastElement ) return true
29           return false
30     end function
31
32     // inicijalizacija queue
33     function Init( queue, ref firstElement, ref lastElement )
34           firstElement = 1
35           lastElement = 1
36     end function
================================================================
```

Primetimo da je vremenska složenost opisanih metoda $O(1)$.

Kao i kod steka realizovanog preko niza, i ovde smo ograničili broj elemenata koji mogu da se dodaju u skup. Ovo rešavamo slično kao i kod steka korišćenjem liste, međutim kako kod liste mi imamo pokazivač samo na prvi element u listi, potrebno je pamtiti jos jedan pokazivač koji će da pokazuje na poslednji element u listi. Pri ubacivanju elementa u skup ćemo koristiti pokazivač koji pokazuje na poslednji element u listi i ubaciti element posle poslednjeg, te promeniti da pokazivač pokazuje na element koji smo ubacili, dok ćemo pri izbacivanju elementa iz skupa koristiti pokazivač koji pokazuje na prvi element u listi.