ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Еднодимензионални податочни структури

АЛГОРИТМИ И ПОДАТОЧНИ СТРУКТУРИ

- предавања -

Еднодимензионални податочни структури

- Апстрактни податочни типови - АПТ
- Стек (Stack)
- Редица (Queue)
- Приоритетна редица (Priority queue)

Апстрактни податочни типови

- Апстрактен податочен тип
  - Специфицирање на множеството вредности и операциите над тие вредности
  - Нема спецификација на начинот на кој ќе бидат претставени податоците и имплементирани операциите
- Дефинирање на АПТ
  - “Договор”
  - Какви се вредностите
  - Какви се операциите – како сакаме да се однесуваат

Спецификација vs. имплементација

- Договор – спецификација
  - Содржи информации за тоа кои вредности се можни
  - Кое е бараното однесување на операциите
  - Нема никаква информација за самото преставување на податоците и имплементација на операциите
- Имплементација
  - Се однесува на одреден договор
  - Дава одговор на прашањата:
    - Кои фундаментални структури ќе се користат за претставување на податоците
    - Кои алгоритми ќе се користат за имплементација на операциите
  - Можни се и повеќе имплементации на еден ист договор

Предности од АПТ

- Поделба на одговорноста
  - Оној кој бара АПТ и кој подоцна ќе користи АПТ, треба само добро да специфицира
  - Оној кој имплементира АПТ, треба да го испочитува договорот
- Повеќекратни имплементации
  - Различна имплементација, за различна намена
  - Доколку сите различни имплементации го почитуваат истиот договор, можат да бидат менувани по потреба
    - Се постигнува баланс на користење на ресурси и перформанси

Стек

- Еднодимензионална линеарна секвенца од елементи според принципот последен-внесен-прв-изваден
  - Елементите во секвенцата можат да се додаваат и вадат само на едниот нејзин крај – **врв** на стекот
- Длабочина на стек – број на елемени што ги содржи
  - Празен стек има длабочина 0

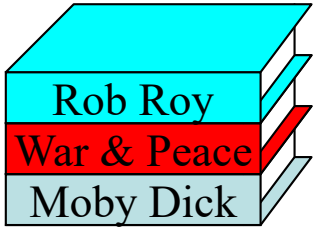
Илустративен пример

- Стек од книги

Илустративен пример

- Стек од книги

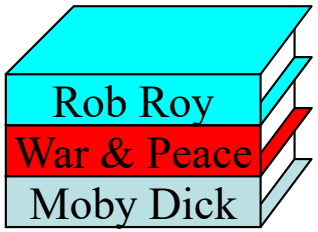
На почеток:



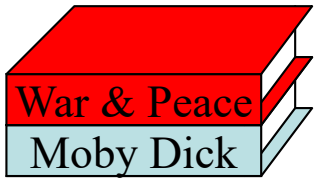
Илустративен пример

- Стек од книги

На почеток:



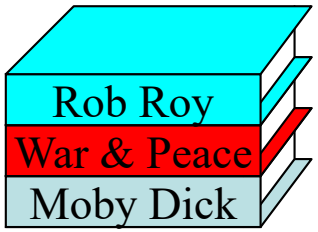
По вадење на книга:



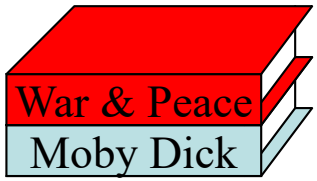
Илустративен пример

- Стек од книги

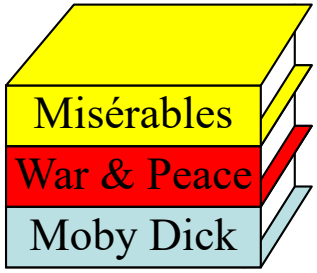
На почеток:



По вадење на книга:



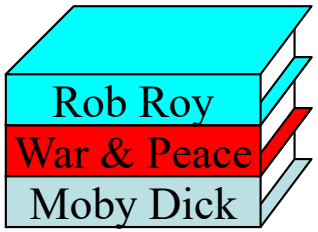
По додавање на книга:



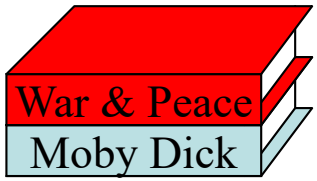
Илустративен пример

- Стек од книги

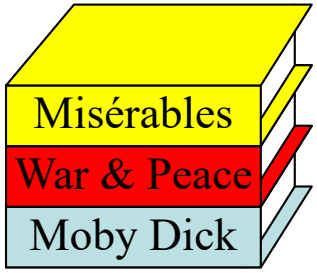
На почеток:



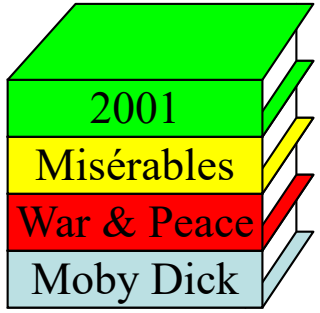
По вадење на книга:



По додавање на книга:



По додавање на уште една книга



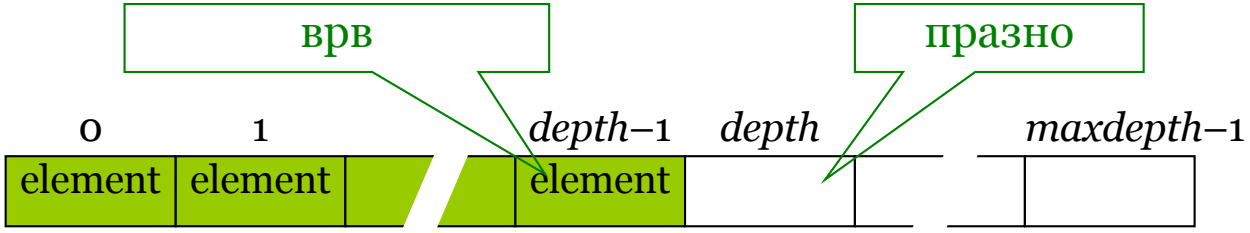
Побарувања

- Апстрактен податочен тип стек
- Операции:
  - Празнење на целиот стек
  - Проверка дали стекот е празен
  - Додавање елемент на врвот на стекот (операција “push”)
  - Вадење елемент од врвот на стекот (операција “pop”)
  - Дополнително, проверка на првиот елемент во стекот без негово вадење.

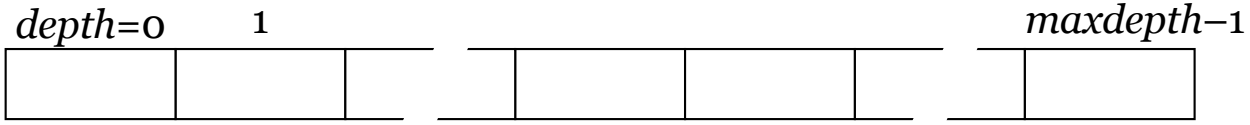
Стек – имплементација со низа

- Ограничена имплементација
- Стекот е со големина помала или еднаква на однапред дефинирана максимална големина (*maxdepth*)
  - Променлива *depth* која ја содржи тековната длабочина
  - Низа од елементи *elems* со должина *maxdepth* во која се содржат елементите на позиците *elems[0..depth-1]*

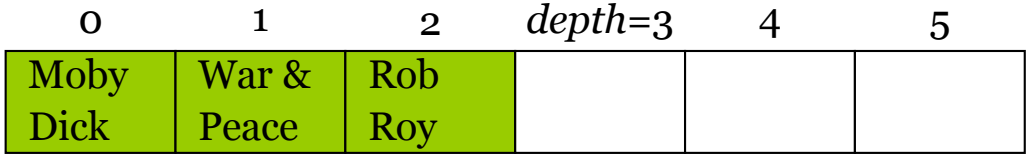
Опис



Празен стек

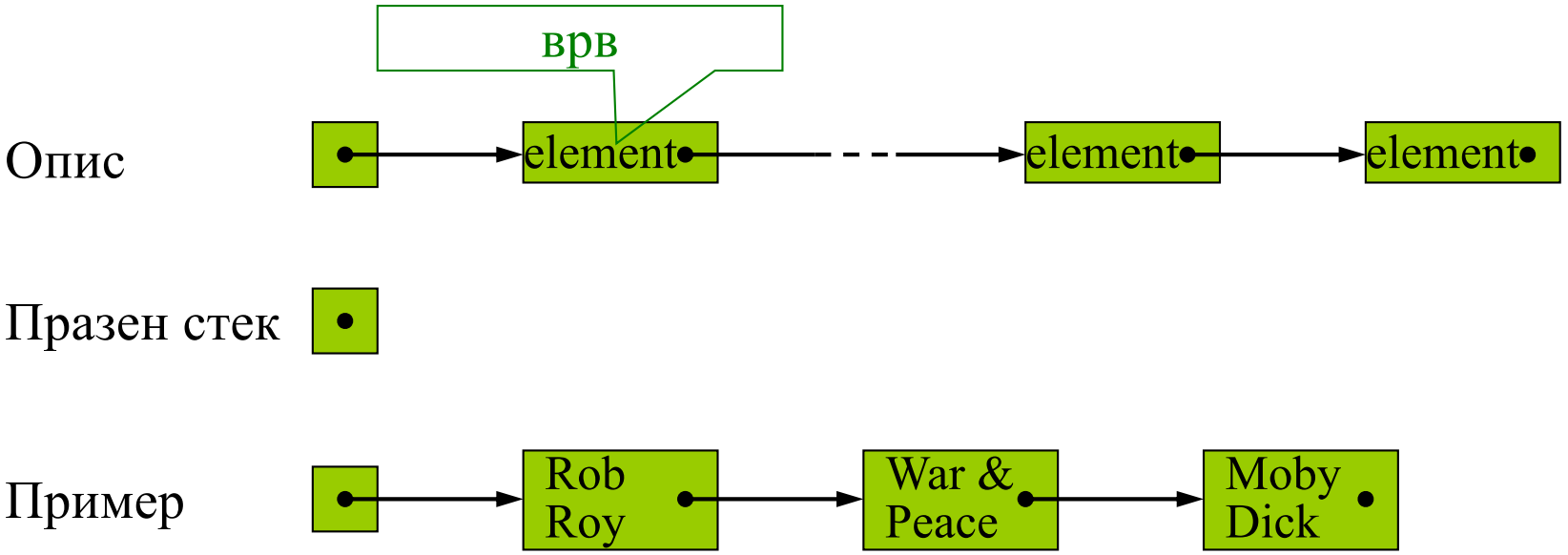


Пример (*maxdepth* = 6):



Стек – имплементација со поврзана листа

- Неограничен стек, преставен со едностарно поврзана листа, каде првиот елемент од листата го претставува врвот на стекот



Примери и примена

- Превртување на редоследот на елементи (записи) во една датотека
- Проверка на добро поставени загради
- Пресметување на вредност на израз во постфикс нотација
- Повикување на под-процедури
- ...

Превртување на редослед

- Проблем: дадена е датотека со податоци подредени во одреден редослед. Да се напише алгоритам за нивно едноставно превртување (првиот да биде последен, ...)

Направи празен стек.
Се додека има линии во влезната датотека, **повторувај**:
    Прочитај линија
    Додај ја на врвот на стекот.
Се додека стекот има елементи, **повторувај**:
    Извади линија од врвот на стекот
    Запиши ја во излезната датотека.
Крај.

Проверка на загради во изрази

- Проблем: проверка на добро поставени загради во аритметички израз
- Правила:
  - Секоја отворена заграда мора да има затворена заграда
  - Доколку се користат загради од различен тип (мали, средни, големи), тие треба да се вгнездени
- Еве неколку примери за добро поставени загради
  - $s \times (s - a) \times (s - b) \times (s - c)$
  - $\{- b + \sqrt{[b^2 - 4(ac)]}\} / 2a$
- Следните примери се лошо поставени загради
  - $s \times (s - a) \times (s - b \times (s - c)$
  - $s \times (s - a) \times s - b) \times (s - c)$
  - $\{- b + \sqrt{[b^2 - 4ac)}\} / 2a$

Алгоритам за проверка со стек

Направи празен стек.
Се додека има симболи во изразот (од лево кон десно), **повторувај**:
Прочитај симбол
Ако симболот е отворена заграда.
Додај го на стекот.
Ако симболот е затворена заграда
Ако стекот е празен
врати погрешно и **заврши**.
Ако на врвот од стекот се наоѓа различен тип отворена заграда
врати погрешно и **заврши.**
Извади го елементот од врвот од стекот.
Ако стекот е празен
Врати точно и **заврши**
Во спротивно
Врати погрешно и **заврши**
Крај.

Евалуација на израз во постфикс нотација

- Различни начини на запишување на аритметички изрази (нотации)
- Најблизок за разбирање на луѓето е инфикс, кога операторот се наоѓа помеѓу операндите.
  - (5+9) * 2+6 * 5
- Префиксна или полска нотација, каде најпрвин се запишуваат операторите, а потоа операндите.
  - + * + 5 9 2 * 6 5
- Обратната нотација, каде операторите се запишуваат после операндите се нарекува обратна полска нотација или постфикс нотација.
  - 5 9 + 2 * 6 5 * +
- Кај префикс и постфикс, нема потреба од загради

Алгоритам за евалуација на израз во постфикс нотација

Направи празен стек.

Се додека има симболи во изразот (од лево кон десно), **повторувај**:

Прочитај симбол

Ако симболот е операнд.

Додај го на стекот.

Ако симболот е оператор

Извади елемент од стекот

Извади елемент од стекот

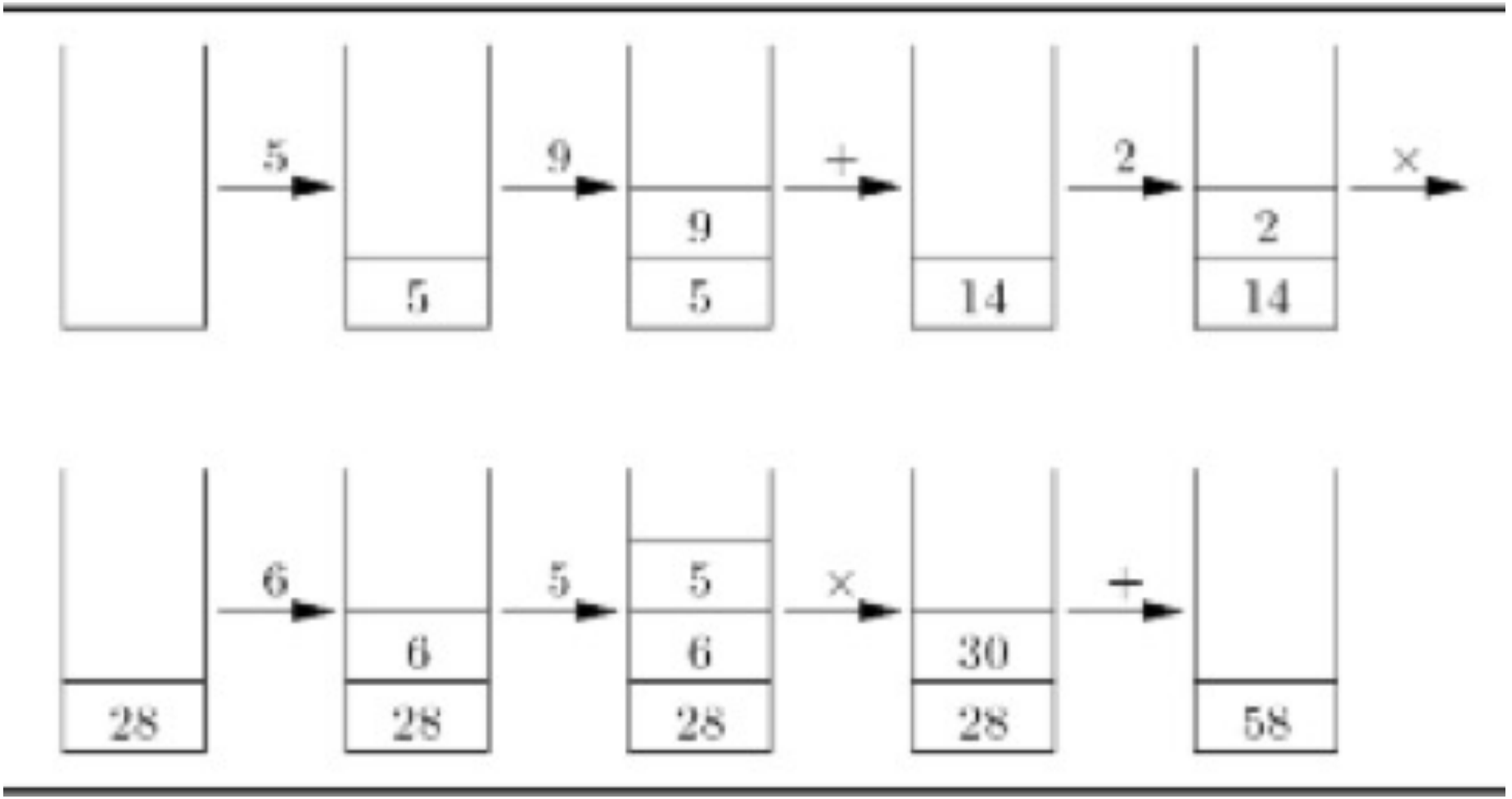
Примени ја операцијата што ја дефинира операторот врз двата операнди извадени од стекот

Додај го на стекот резултатот

Извади го елементот од стекот

Печати го како резултат

Извршување на алгоритмот



Редица

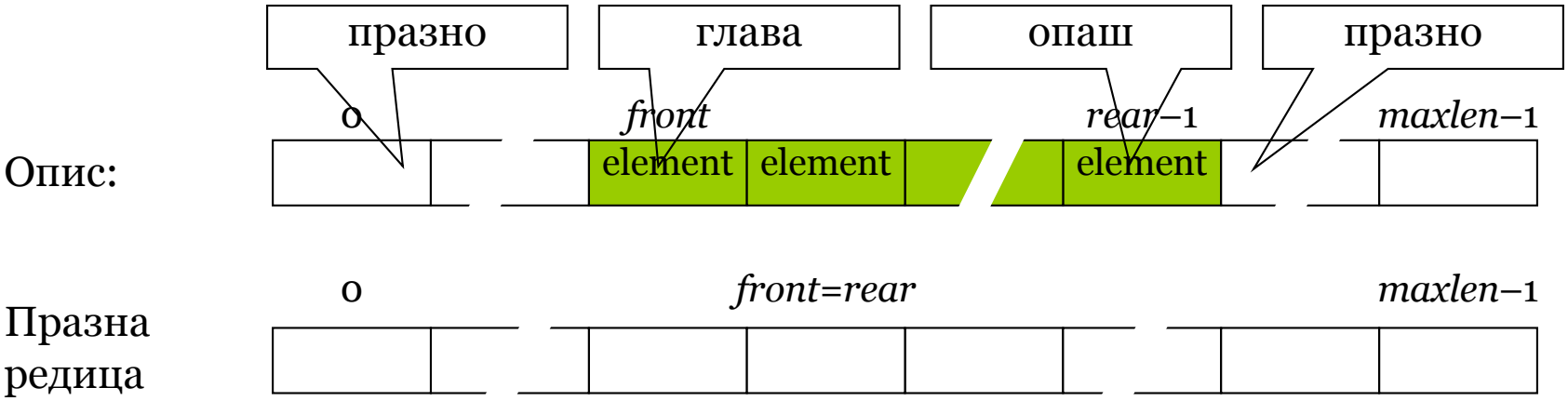
- Еднодимензионална линеарна секвенца од елементи според принципот прв-внесен-прв-изваден
  - Елементите во секвенцата можат да се додаваат на едниот крај (**опашот**), а вадат другиот крај (**главата**) на редицата
- Должина на редица– број на елементи што ги содржи
  - Празна редица има длабочина 0

Побарувања

- Апстрактен податочен тип редица
- Операции:
  - Празнење на целата редица
  - Проверка дали редицата е празна
  - Додавање елемент на опашот од редицата (операција “en-queue”)
  - Вадење елемент од главата на редицата (операција “de-queue”)
  - Дополнително, проверка на елементот на главата на редицата без негово вадење.

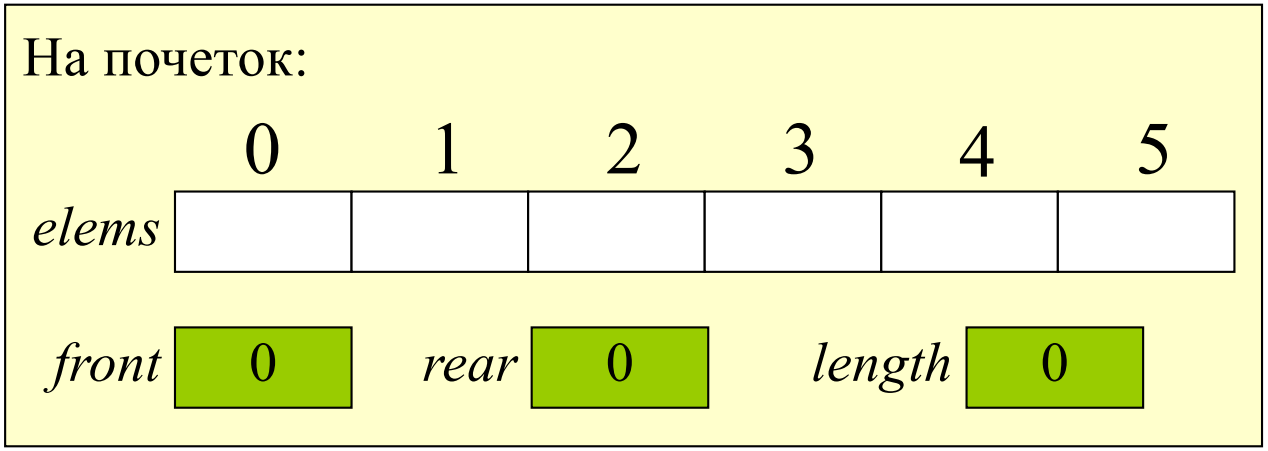
Редица – имплементација со низа

- Ограничена имплементација
- Редицата е со капацитет помал или еднаков на однапред дефинирана максимална должина (*maxlen*)
  - Променлива *length*, која ја содржи моменталната должина
  - Променливи глава (*front*) и опаш (*rear*)
  - Низа од елементи *elems*, кои ги содржат елементите на редицата на позициите *elems[front…rear-1]*



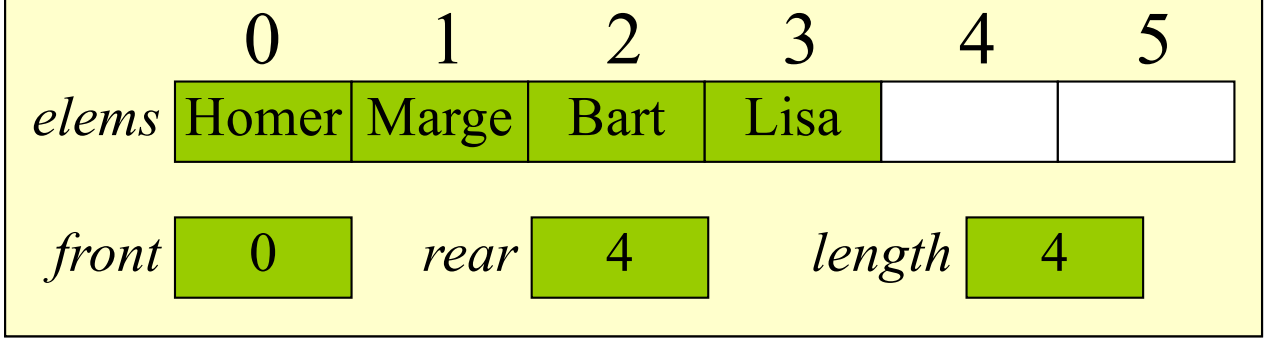
Илустрација

Илустрација

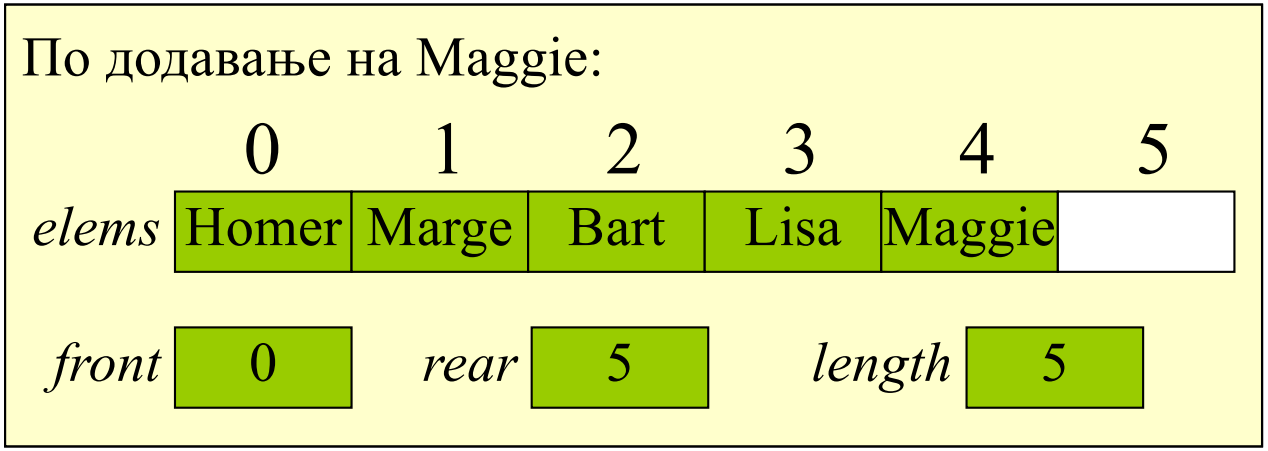


Илустрација

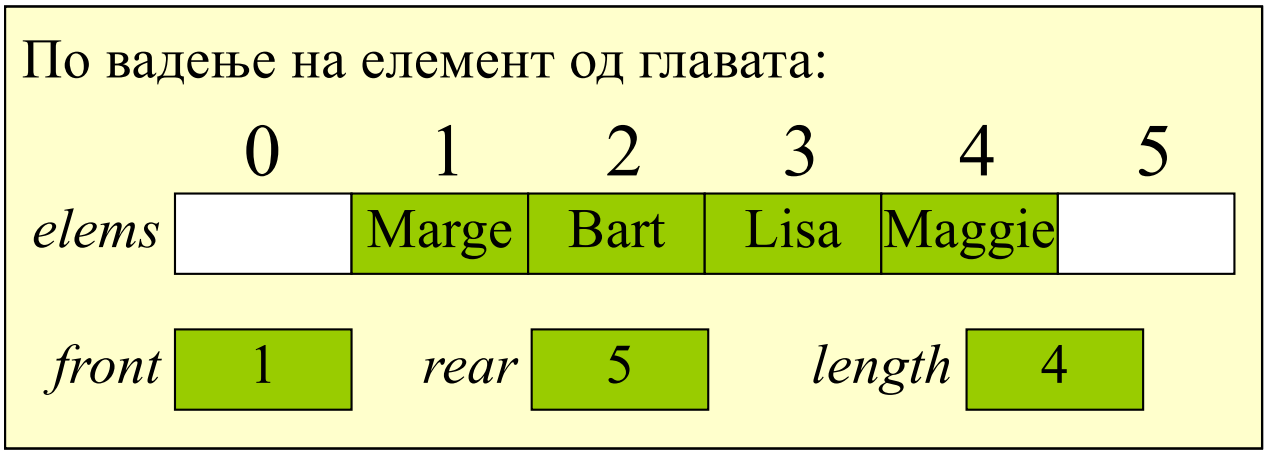
По додавање на Homer, Marge, Bart, Lisa:



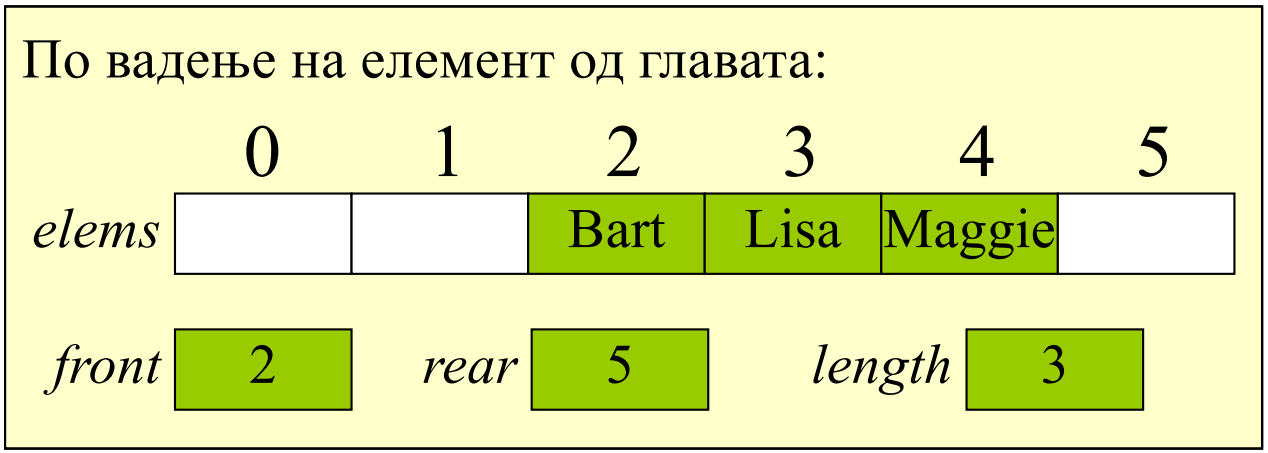
Илустрација



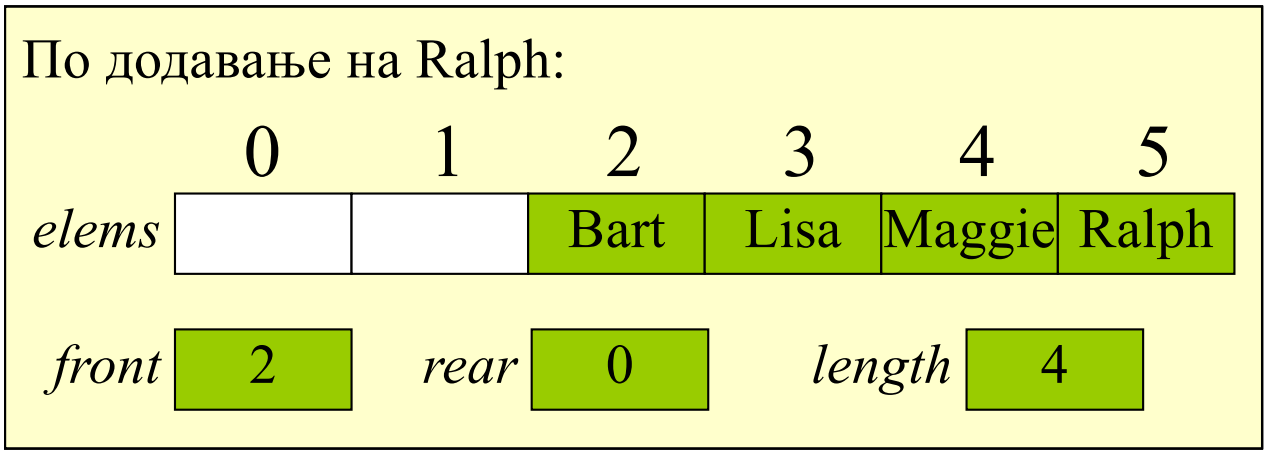
Илустрација



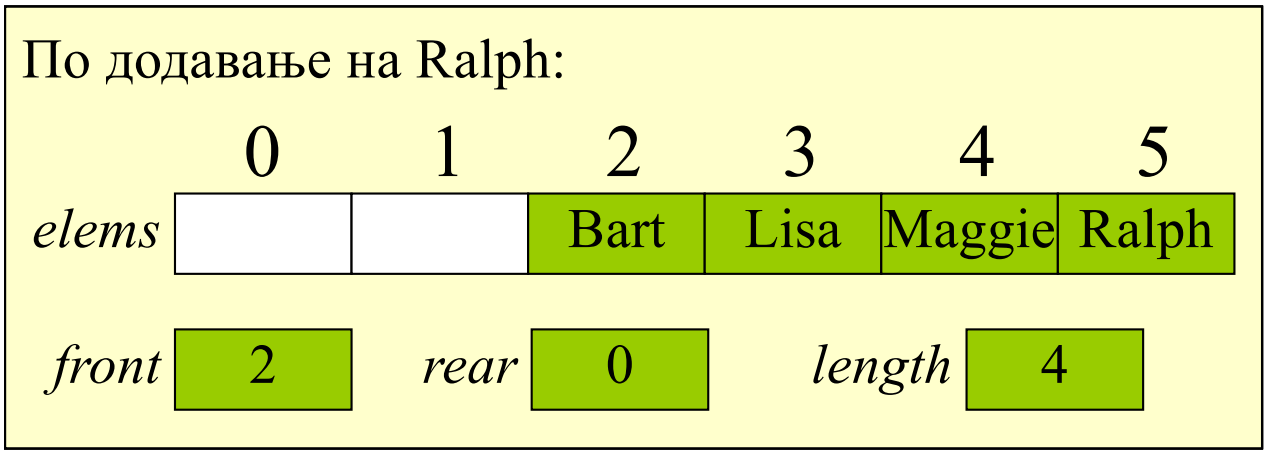
Илустрација



Илустрација



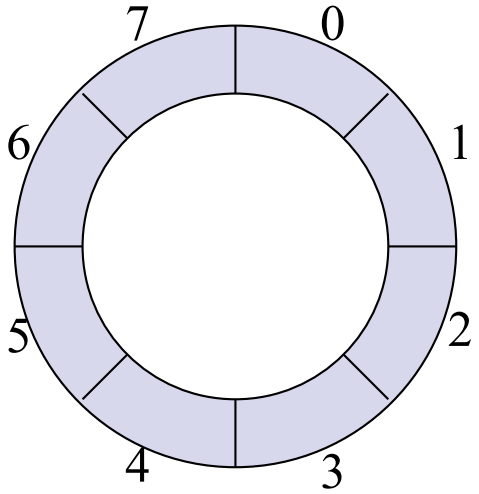
Илустрација

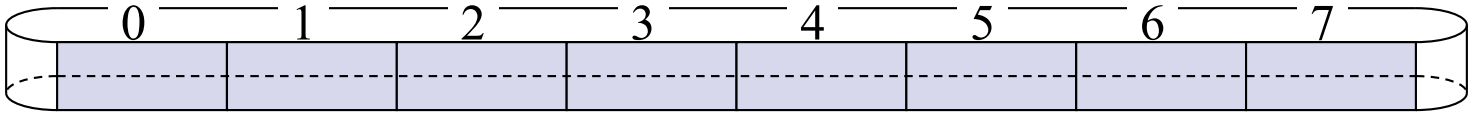


Што ако сега сакаме да додадеме уште еден елемент?
Како да се справиме со поместувањето?

Решение – циклична низа

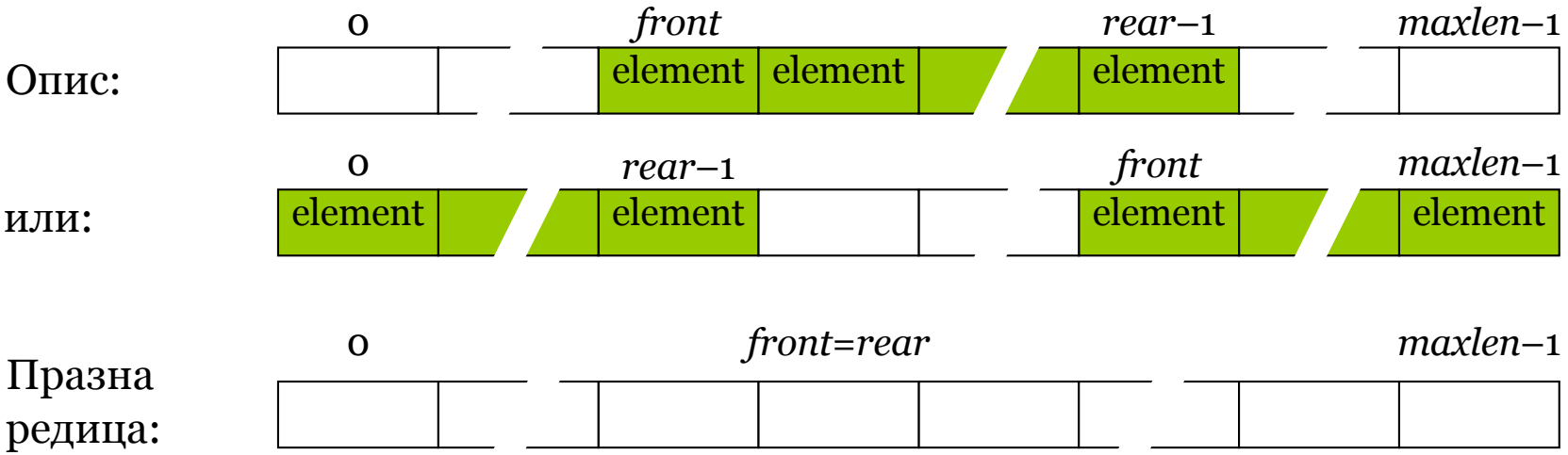
- Во циклична низа *a* со должина *n* секој елемент има и претходни и следбеник. При тоа, претходник на *a[0]* е *a[n-1]*, а следбеник на *a[n-1]* е *a[0]*.
- Визуелно ги замислуваме вака





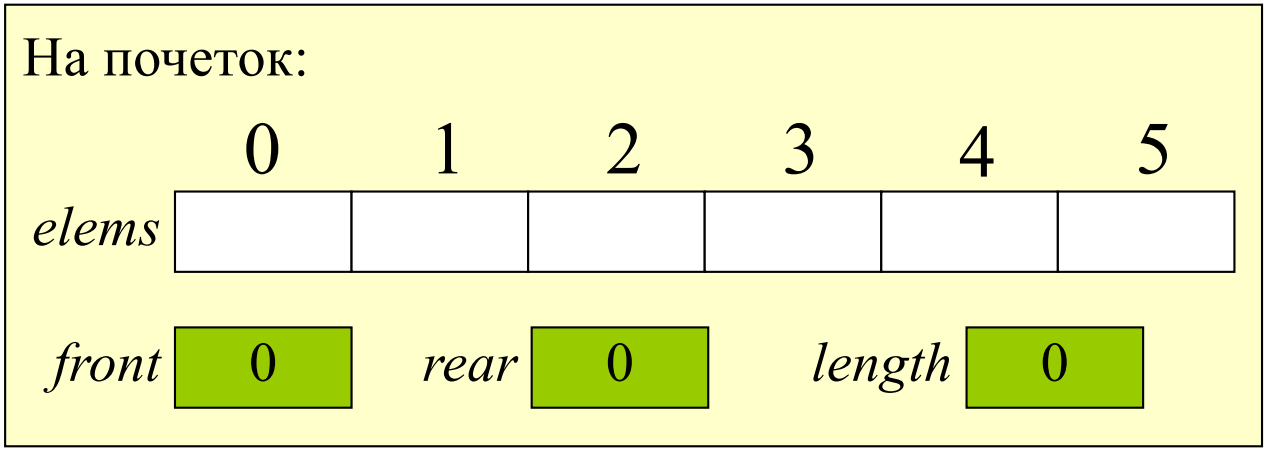
Редица – имплементација со циклична низа

- Ограничена редица, со капацитет помал или еднаков на однапред дефинирана максимална должина (*maxlen*)
  - Променлива *length*, која ја содржи моменталната должина
  - Променливи глава (*front*) и опаш (*rear*)
  - Циклична низа од од елементи *elems*, кои ги содржат елементите на редицата на позициите *elems[front…rear-1]* или *elems*[*front*…*maxlen*–1] и *elems*[0…*rear*–1]

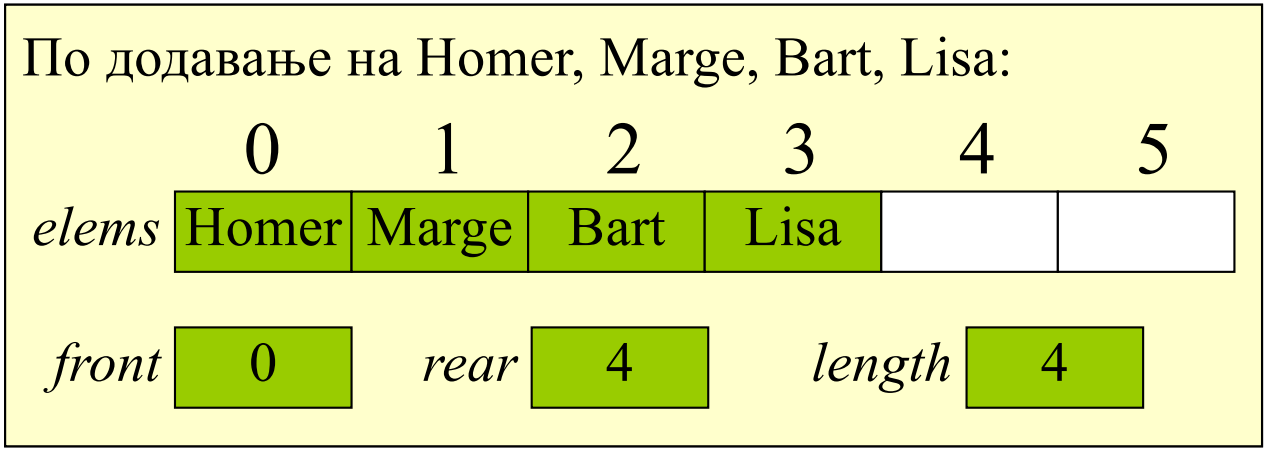


Илустрација – циклична низа

Илустрација – циклична низа

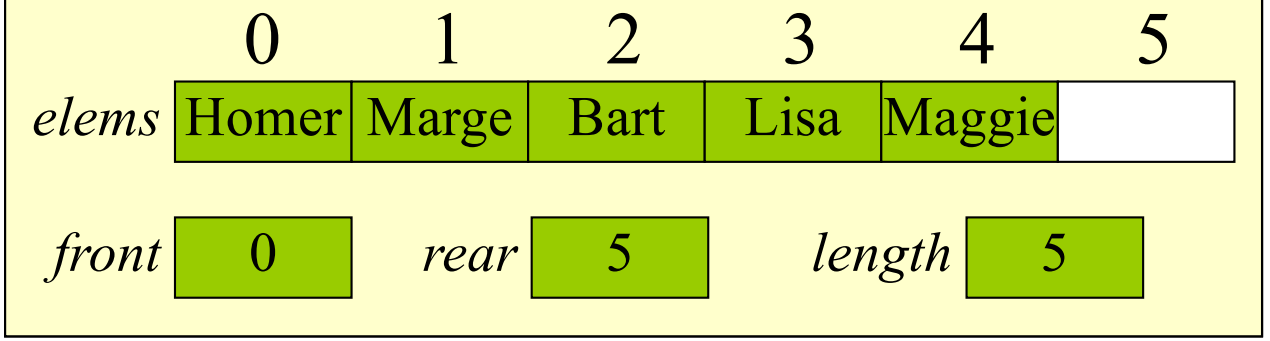


Илустрација – циклична низа

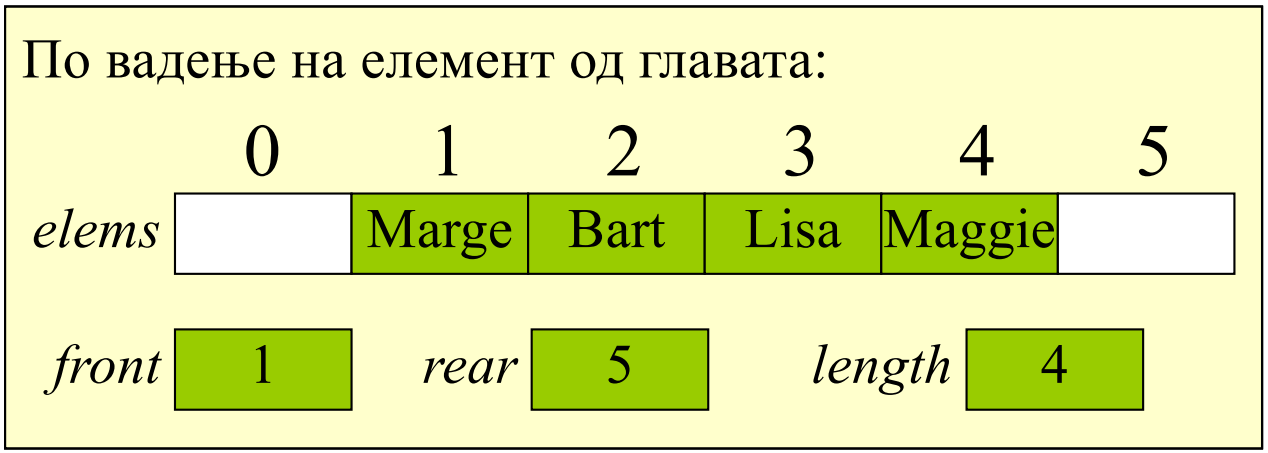


Илустрација – циклична низа

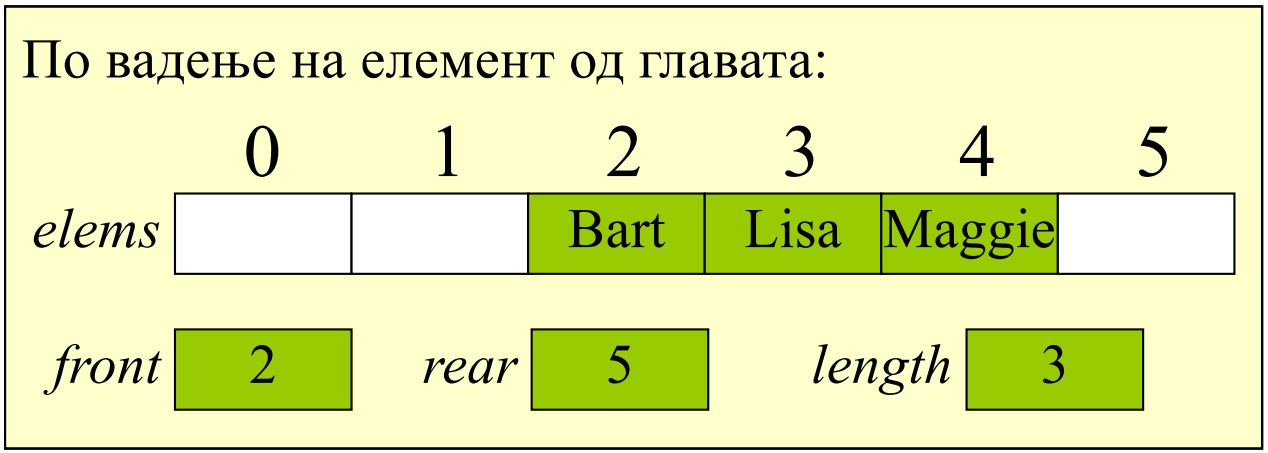
По додавање на Maggie:



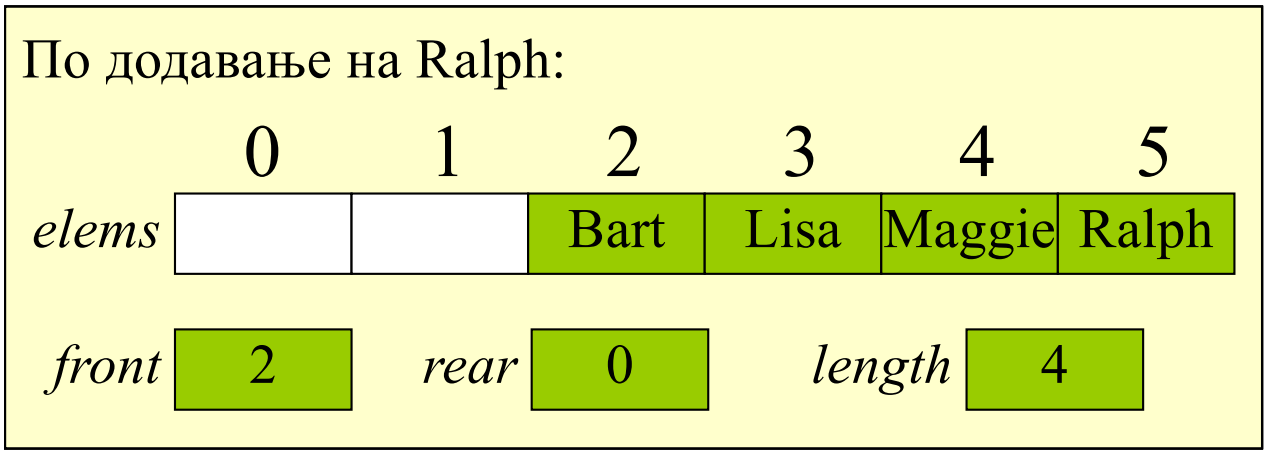
Илустрација – циклична низа



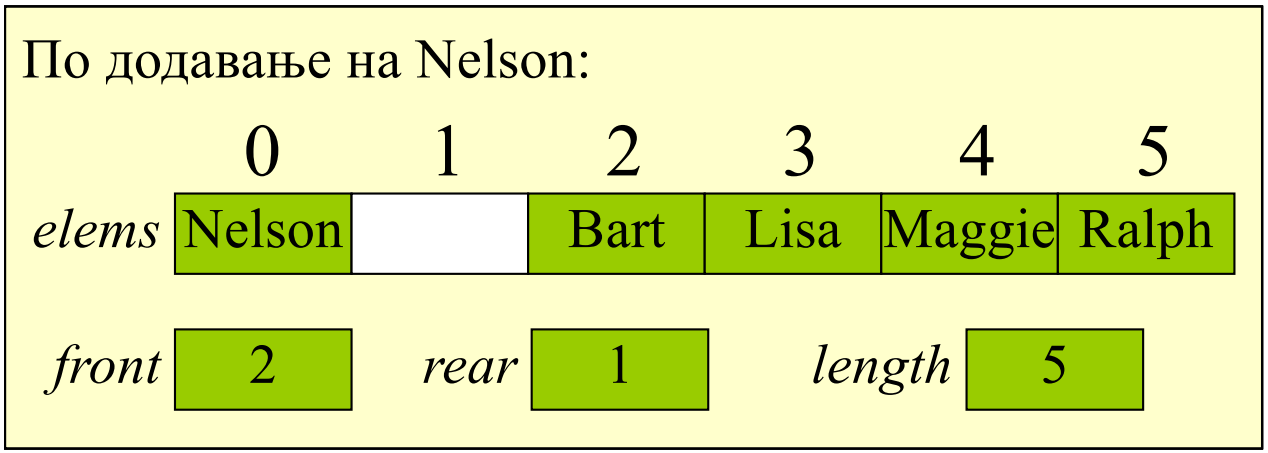
Илустрација – циклична низа



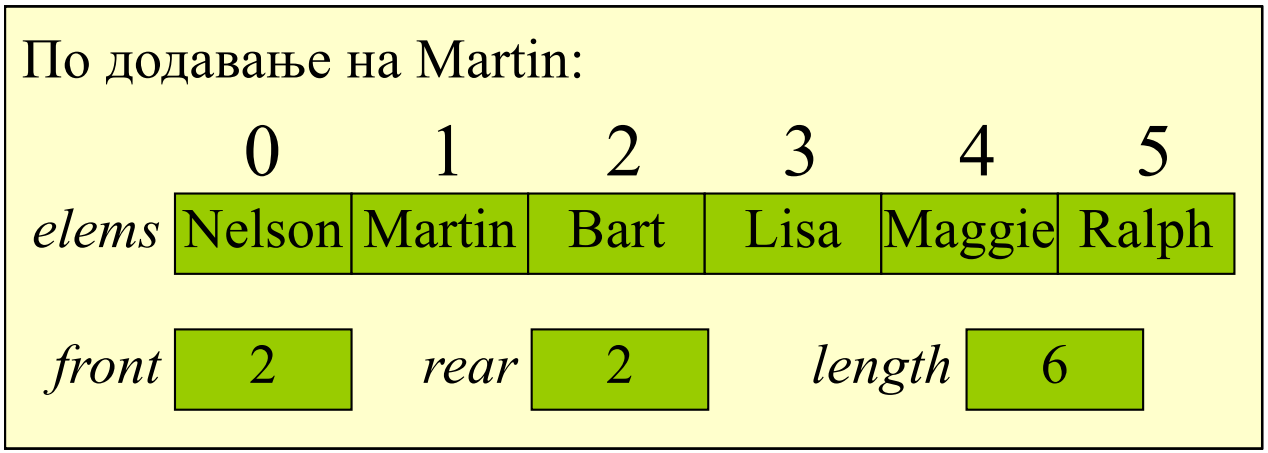
Илустрација – циклична низа



Илустрација – циклична низа

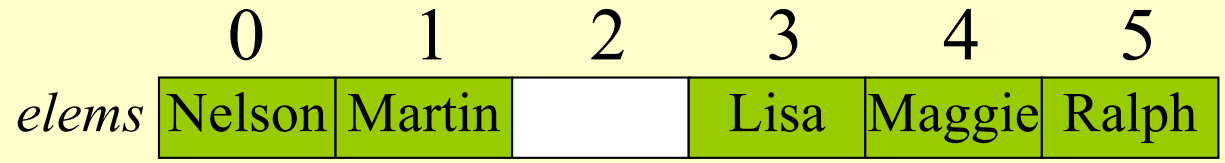


Илустрација – циклична низа



Илустрација – циклична низа

По вадење на елемент од главата:

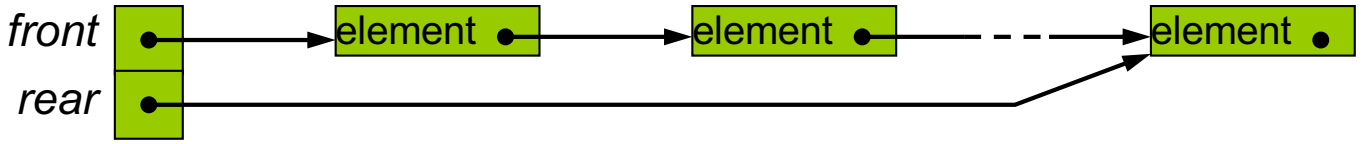


front 3 *rear* 2 *length* 5

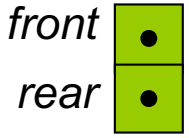
Редица – имплементација со поврзана листа

- Неограничена редица, претставена со
  - Поврзана листа, каде првиот елемент е главата на редицата, а чие заглавие содржи два покажувачи: еден кон првиот елемент (глава - *front*) и еден кон последнот елемент (опаш - *rear*)
  - Промелива *length*

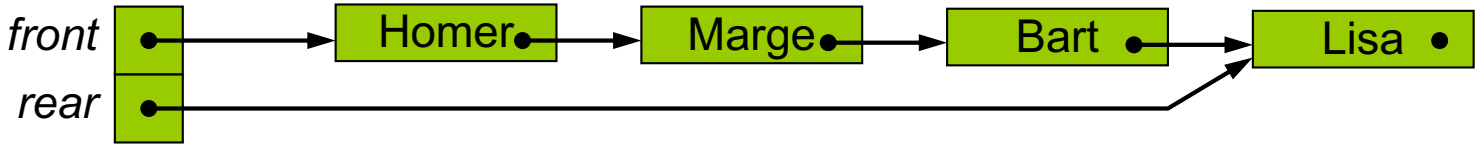
Опис:



Празна редица:



Пример:



Примери и примена

- Принт сервер – редица на чекање на задачи на мрежен принтер
- Диск драјвер – редица на чекање на барања за пристап до податоци од диск
- Распределувач на процеси кај оперативен систем
- Разделување на подредена датотека на делови (кои се исто така подредени)

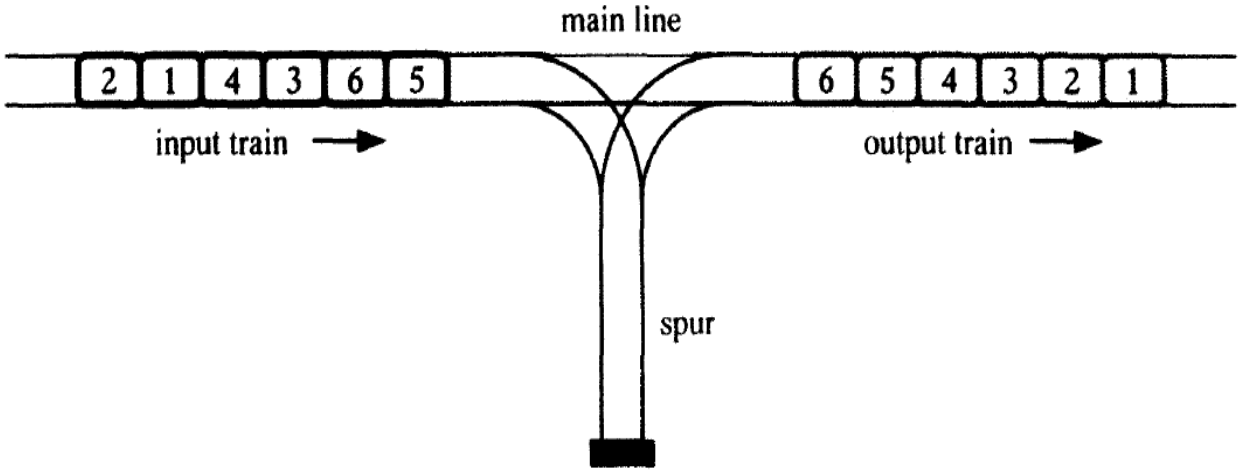
Разделување - demerging

- Дадена е датотека со податоци за студентите, подредена според нивниот број на индекс.
- За одредена обработка, потребно е да се подели оваа датотека на две, една со машки, една со женски студенти, но сепак да се задржи подредувањето според бројот на индекс.

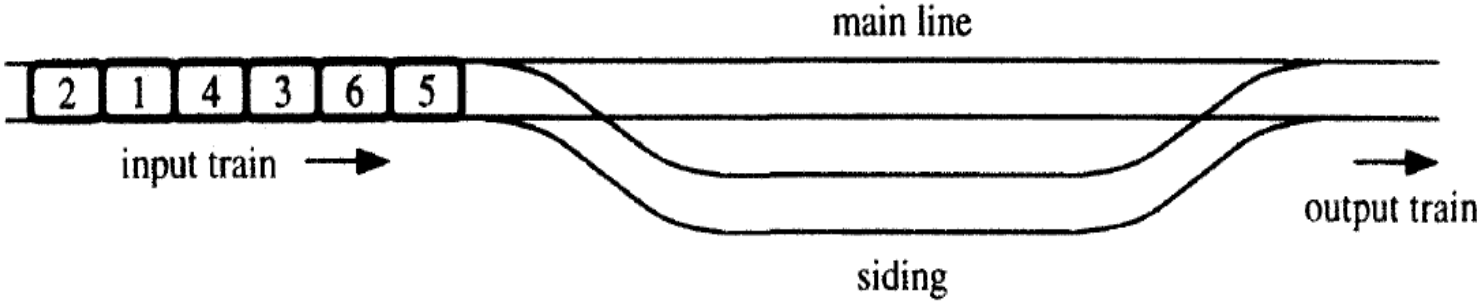
Направи празни редици машки и женски
Се додека има податоци во влезната датотека,
повторувај:
    Прочитај запис од влезна датотека
    Ако атрибут пол е машко
        Постави запис во редица машки
    Во спротивно
        Постави запис во редица женски
Се додека има податоци во редица машки,
повторувај:
    Извади запис од редица
    Запиши во датотека машки
Се додека има податоци во редица женски,
повторувај:
    Извади запис од редица
    Запиши во датотека женски
Крај.

За размислување

- Подредување на вагони
- Случај 1



- Случај 2



Приоритетна редица

- Еднодимензионална линеарна секвенца од елементи, каде секој елемент има свој приоритет.
- Елементот со наголем приоритет секогаш прв се вади од листата
- Должина на приоритетна редица е бројот на елементи што ги содржи
  - Празната приоритетна редица има должина 0

Примена

- Распределувачи на процеси со приоритет кај оперативни системи
- Подредување

Пример - Подредување

- Отвори датотека
- Се додека има елементи
  - Прочитај елемент и постави во приоритетна редица
- Затвори датотека
- Отвори излезна датотека
- Се додека има елементи во приоритетна редица
  - Извади елемент од приоритетна редица (оној со најголем приоритет) и запиши го во излезна датотека
- Затвори излезна датотека

Пример

- Отвори датотека
- Се додека има елементи
  - Прочитај елемент и постави во приоритетна редица
- Затвори датотека
- Отвори излезна датотека
- Се додека има елементи во приоритетна редица
  - Извади елемент од приоритетна редица (оној со најголем приоритет) и запиши го во излезна датотека
- Затвори излезна датотека

Побарувања

- Апстрактен податочен тип приоритетна редица
- Операции:
  - Празнење на целата приоритетна редица
  - Проверка дали приоритетната редица е празна
  - Додавање елемент во приоритетната редица
  - Вадење елементот со најголем приоритет од редицата
  - Дополнително, проверка на елементот со најголем приоритет во редицата без негово вадење.

Приоритетна редица - имплементација

- Две можности при имплементација
  1. Подредена редица
  2. Неподредена редица
- И двете се независни од фундаменталниот податочен тип што ќе се користи (во однос на перформансите)
  - Со низа – ограничена приоритетна редица со предефиниран капацитет
  - Со поврзана листа – неограничена приоритетна редица

Варијанта 1 – подредена приоритетна редица

- Имплементација со подредена низа или подредена поврзана листа
- И во двата случаи, додавањето на нов елемент значи
  1. Наоѓање на позицијата (според приоритетот) каде треба да се постави тој елемент
  2. Вметнување на самиот елемент
- При имплементација со низа
  1. Пребарување низ низата за одредување на вистинското место
  2. Поместување на елементите за да се отвори место
- При имплементација со листа
  1. Изминување на еден по еден елемент за пронаоѓање на местото
  2. Вметнување на елементот

Варијанта 2 – неподредена приоритетна редица

- Имплементација со неподредена низа или неподредена поврзана листа
- Додавање на нов елемент
  - При имплементација со низа – додавање на крај
  - При имплементација со листа – додавање на почеток
- Вадење на елемент со наголем приоритет
  - И во двете имплементации, поради неподреденоста на елементите, тие мора да се пребаруваат од почеток до крај

Споредба на имплементациите

Операција	Подредена поврзана листа	Неподредена поврзана листа	Подредена низа	Неподредена низа
`додај`	$O(n)$	$O(1)$	$O(n)$	$O(1)$
`извади`	$O(1)$	$O(n)$	$O(1)$	$O(n)$

Назад на примерот со подредување

Комплексност на додавање на n елементи во приоритетна редица, а потоа нивно вадење

- При имплементација со подредена низа или листа, комплексноста е n*O(n) + n*O(1), односно $O(n^2)$
- При имплементација со неподредена низа или листа, комплексноста е n*O(1) + n*O(n), односно $O(n^2)$

Назад на примерот со подредување

Додавање на n елементи во приоритетна редица, а потоа нивно вадење

- При имплементација со подредена низа или листа, комплексноста е n*O(n) + n*O(1), односно $O(n^2)$
- При имплементација со неподредена низа или листа, комплексноста е n*O(1) + n*O(n), односно $O(n^2)$

А може ли подобро?
Одговорот во втората половина од курсот...