CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Динамични структури – **СТЕК и ОПАШКА**

CODE ACADEMY

Linked List – предимства

1 . Linked lists са динамична структура от данни – създава се, променя се и се изтрива по време на изпълнение на програмата.

2. Linked lists използват паметта ефективно. Паметта не се заделя предварително. Памет се заделя, когато се изисква и се освобождава, когато повече не е нужна.

3. Лесно се вмъкват и изтриват елементи. При свързания списък има гъвкавост при вмъкване и изтриване на данни на определена позиция.

4. С помощта на linked lists лесно могат да се направят сложни приложения.

CODE ACADEMY

Linked List – недостатъци

1 . Linked lists използва много пространство, защото всеки възел изисква допълнителен указател, където да съхранява адресъ към следващия възел

2. Търсенето на определен елемент в Linked lists е трудно и консумиращо време.

CODE ACADEMY

Сравнение между масив и Linked List

ARRAY	LINKED LIST
Size of an array is fixed	Size of a list is not fixed
Memory is allocated from stack	Memory is allocated from heap
It is necessary to specify the number of elements during declaration (i.e., during compile time).	It is not necessary to specify the number of elements during declaration (i.e., memory is allocated during run time).
It occupies less memory than a linked list for the same number of elements.	It occupies more memory.
Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.	Inserting a new element at any position can be carried out easily.
Deleting an element from an array is not possible.	Deleting an element is possible.

Разположение в паметта на Linked List

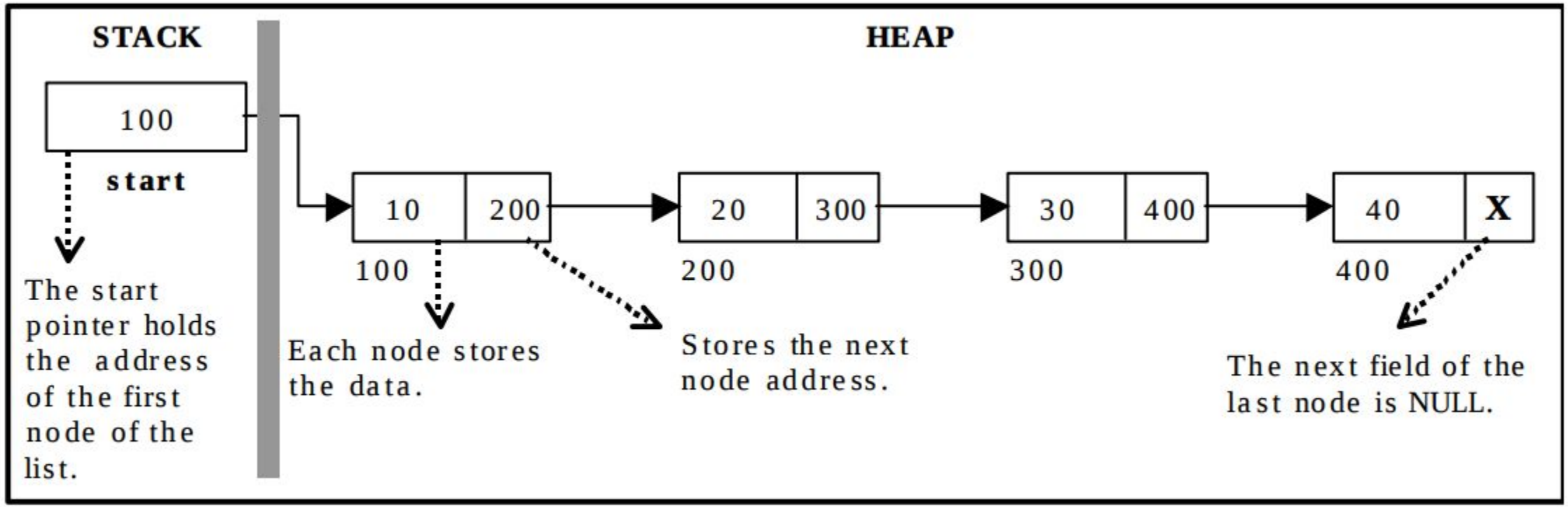


Figure 3.2.1. Single Linked List

Стек

Линейните списъци и масивите позволяват да се вмъква и изтрива на всяко място – в началото, в края и по следата.

Има определени ситуации, в които трябва да се ограничат вмъкванията и изтриванията, така че да могат да се извършват само в началото или в края на списъка, а не в средата.

Две полезни такива структури са:

- Stack - Стек
- Queue - Опашка

CODE ACADEMY

STACK Стек

Стекът е списък от елементи, в който може да се вмъква или трие само в единия край, наречен **top** на стека.

Стекът има организация на данните известна като **LIFO** (last input, first output).

Тъй като данните се добавят и изтриват от единия край, то последният добавен се изтрива първи.

Двете основни операции, асоциирани със стека са:

- **Push**: вмъкване на елемент в стека
- **Pop**: изтриване на елемент от стека

Представяне на Стек

Нека да разгледаме стек с капацитет от 6 елемента. Броят елементи в стека се нарича размер - size на stack.
Броят елементи, които се добавят трябва да е <= от капацитета на стека.

Ако се опитаме да добавим нов елемент след максималния размер на стека, ще възникне прерълване на стека или **stack overflow.**

Ако се опитаме да изтрием елементи повече отколкото е капацитета или под основата на стека, ще възникне **stack underflow.**

Когато се добавят елементи в стека, операцията се изпълнява от **push()**.

Когато се изтриват елементи в стека, операцията се изпълнява от **pop()**.

CODE ACADEMY

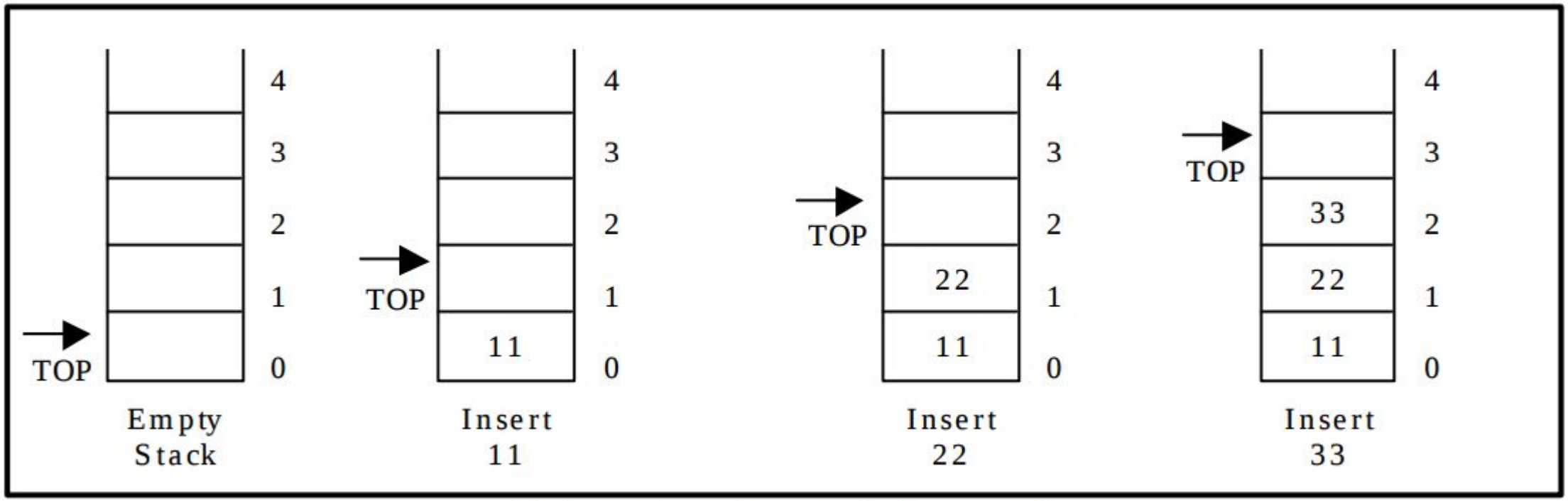


Figure 4.1. Push operations on stack

CODE ACADEMY

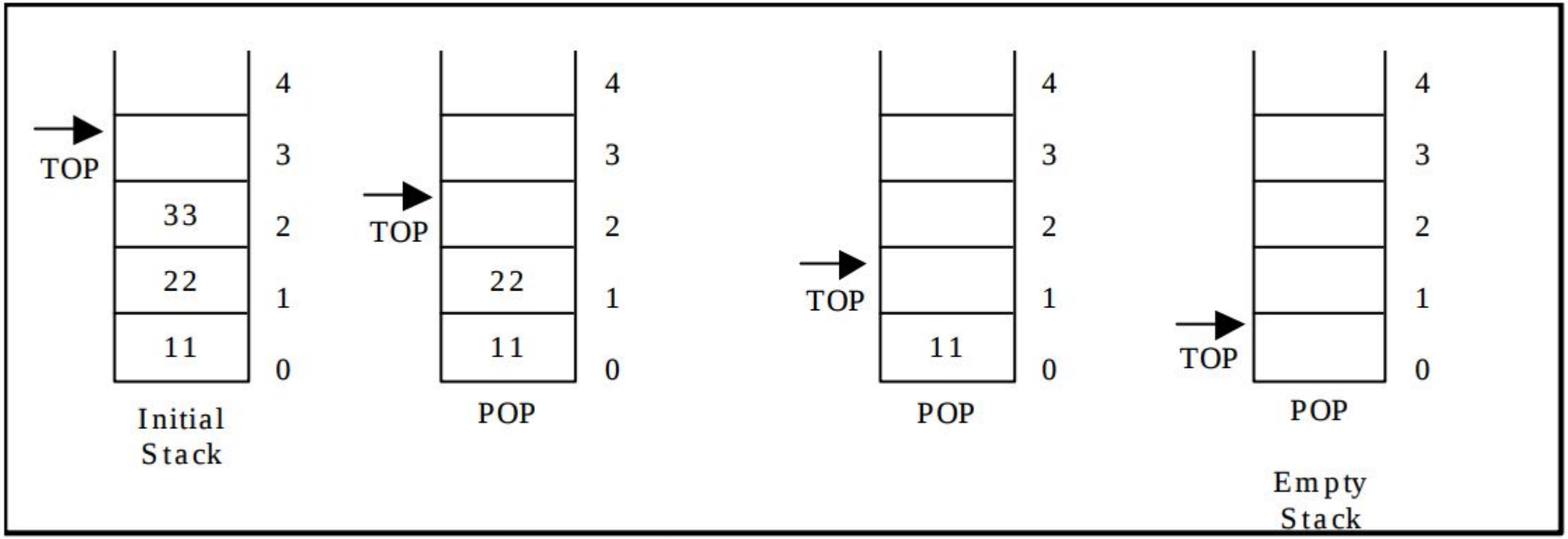


Figure 4.2. Pop operations on stack

CODE ACADEMY

Стек

- Стек - използваме списък като добавяме и изтриваме само от началото на списъка (LIFO - Last In First Out)

```
push/pop <=> X1 ---> X2 ---> X3 --->… ---> XN
```

```
struct _stack {
item_type* array;
int max_size;
int top;
};
```

CODE ACADEMY

Стек

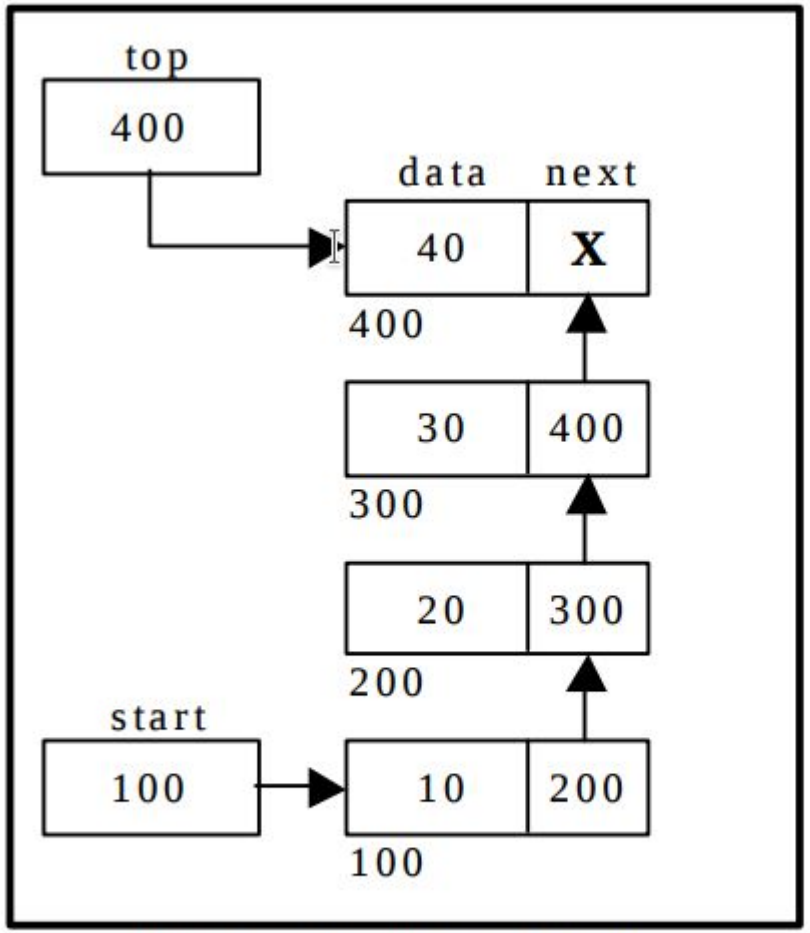


Figure 4.3. Linked stack representation

CODE ACADEMY

Приложение на Стек

- Стека се използва от компилатора да провери дали скобите в изразите са балансирани.
- За изчисляване на изрази – в postfix или в prefix форма
- При рекурсия всички междинни аргументи и върнати стойности се съхраняват в стека на процесора.
- По време на извикване на функция адресът за връщане и аргументите се изтласкват в стека и при връщане се изваждат.
- При алгоритми за решаване на сложни задачи

CODE ACADEMY

Задачи

1) Напишете програма, която имплементира стек с помощта на масив, като съставя стек с помощта на променлива

2) Напишете програма, която имплементира стек с помощта на масив, като съставя стек с помощта на указател

3) Напишете програма, която имплементира стек с помощта на свързан списък.

CODE ACADEMY

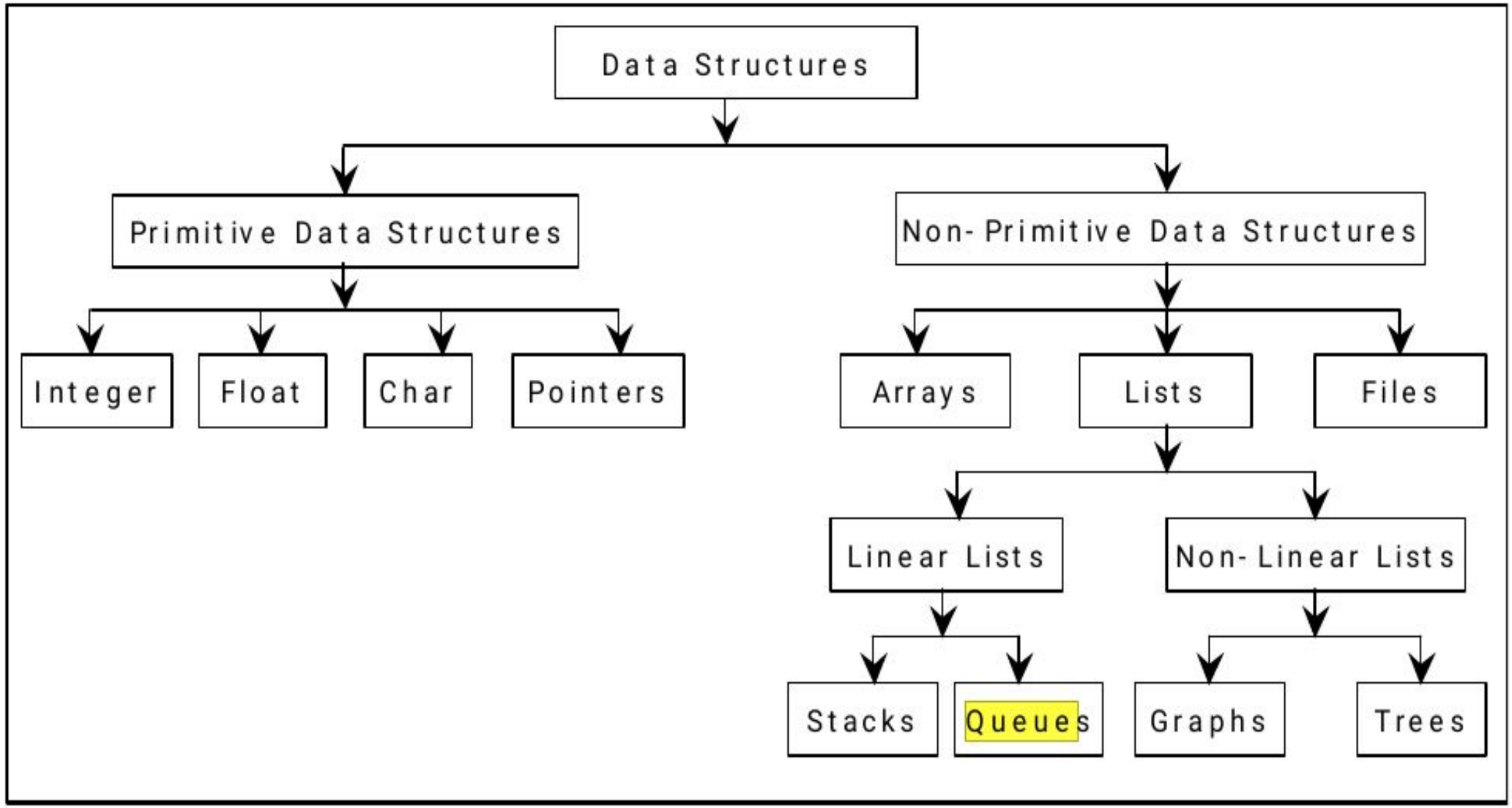


Figure 1.1. Classification of Data Structures

CODE ACADEMY

Опашка

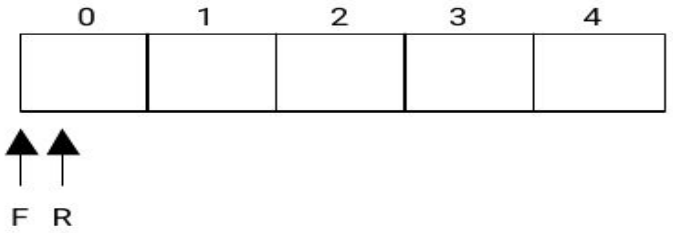
Опашката е друг специален вид списък, където елементите се вмъкват в единия край, наречен заден, и се изтриват в другия край, наречен отпред. Опашката е структура данни подчинена на принципа „FIFO“ или „Първи дошъл-първи излязъл“.

Операциите за опашка са аналози на тези за стек, разликата е, че вмъкванията отиват в края на списъка, а не в началото.

enqueue: за вмъкване на елементи в края на опашката.
dequeue: изтрива елементи от началото на опашката.

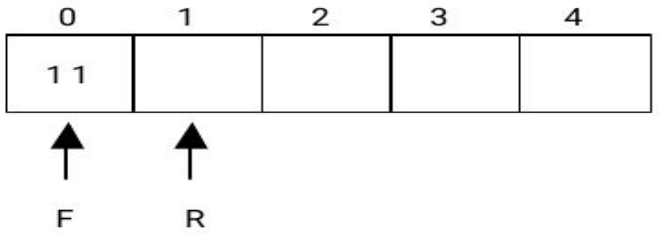
Представяне на Опашка

Ако разгледаме опашка, която съдържа максимум 5 елемента.



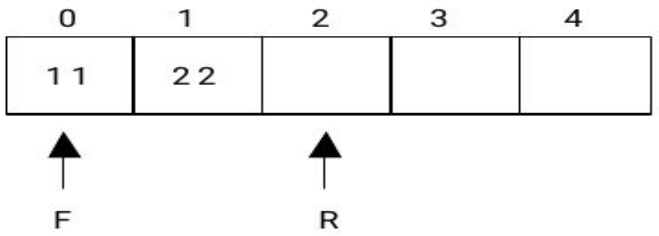
Queue Empty
FRONT = REAR = 0

Now, insert 11 to the queue. Then queue status will be:



REAR = REAR + 1 = 1
FRONT = 0

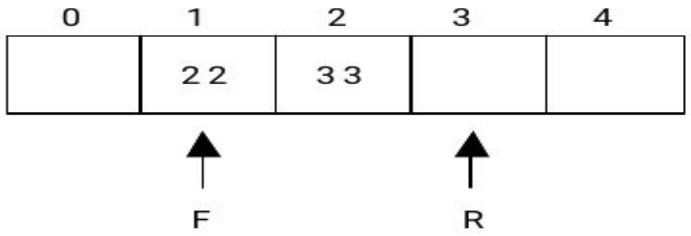
Next, insert 22 to the queue. Then the queue status is:



REAR = REAR + 1 = 2
FRONT = 0

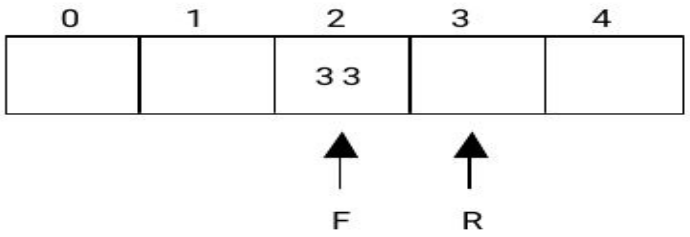
Представяне на Опашка

Ако изтрием елемент от опашката.



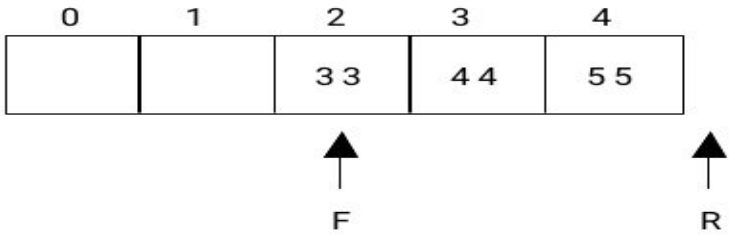
REAR = 3
FRONT = FRONT + 1 = 1

Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The queue status is as follows:



REAR = 3
FRONT = FRONT + 1 = 2

Now, insert new elements 44 and 55 into the queue. The queue status is:



REAR = 5
FRONT = 2

CODE ACADEMY

Опашка

- Опашка - използваме списък като добавяме елементи в началото на списъка, а ги изтриваме от края на списъка (FIFO - First In First Out)

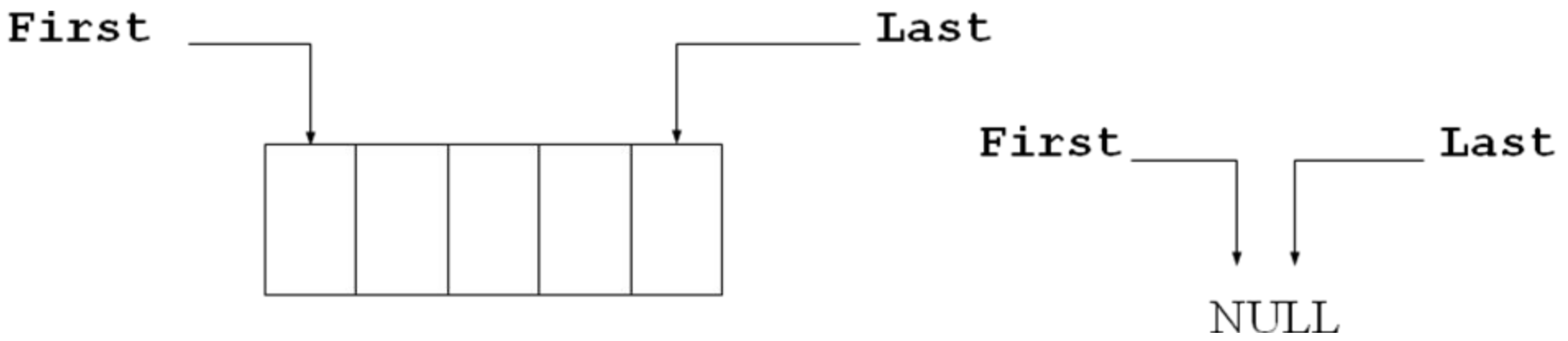
```
write => X1 ---> X2 ---> X3 --->… ---> XN => read

typedef struct node_t{
   int data;
   node_t *next;
}node_t;

extern node_t *first;
extern node_t *last;
```

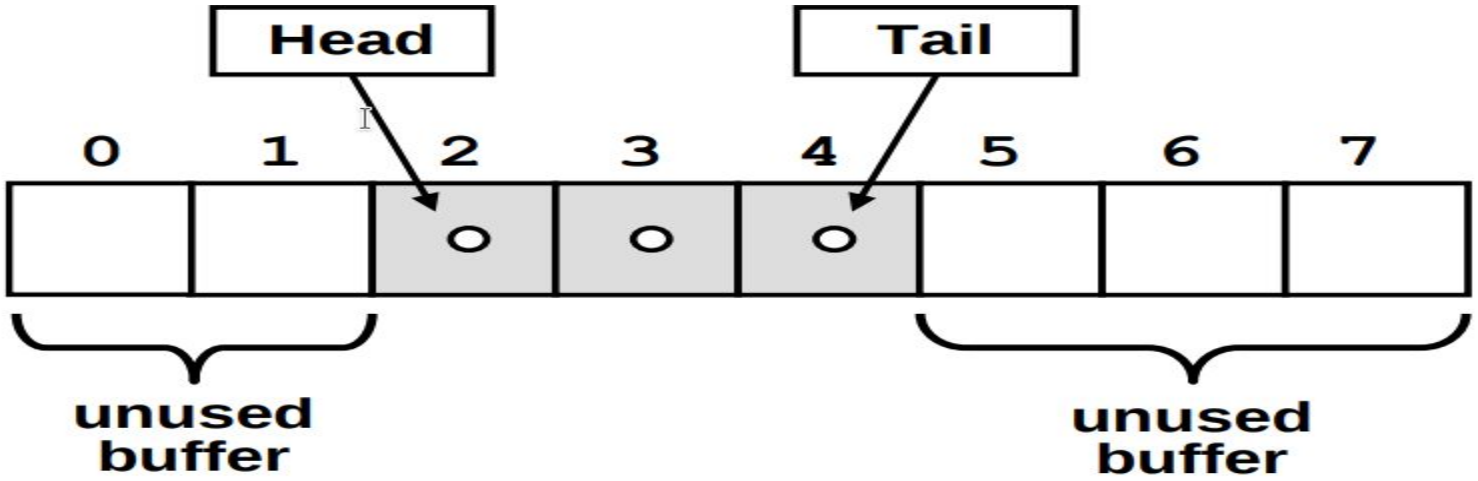
Опашка основни операции

1. Инициализиране на опашката (init).
2. Добавяне на елемент в края на опашката (write).
3. Извличане на елемент от началото на опашката (read).



Задачи

A. Да се имплементира опашка от цели числа с операциите добавяне на елемент в края на опашката и изтриване на елемент в началото на опашката -отпред. За целта да се използва масив.



Задачи

В. Да се имплементира опашка от имена с операциите добавяне на елемент в края на опашката и изтриване на елемент в началото на опашката -отпред. За целта да се използва свързан списък.

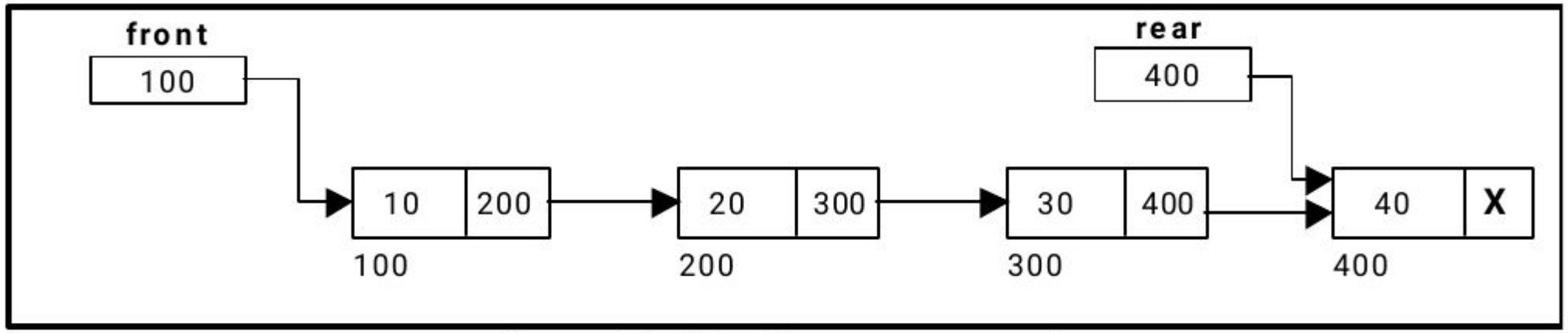


Figure 4.4. Linked Queue representation

Приложение на опашка

1. Използва се при разпределяне на работата на процесите от централния процесор - CPU.

2.Когато много потребители изпращат задание към принтера, всеки от документите за отпечатването се запазва в опашката на принтера. Тогава принтера работи съгласно принципа FIFO

3. Много от алгоритмите по графи използват структурата данни queue, за да намерят елемент от графа. Например, такъв е алгоритъма BFS

CODE ACADEMY

Добри практики

- динамичните структури изискват по-сериозен анализ от статичните такива, преди да започнем да ги използваме трябва да проиграем различни сценарии
- има дебели книги с алгоритми за динамични структури от данни, които е добре да разлистим
- динамичните структури са гъвкави, но имат ограничения и не пасват на всяка задача, има т.н. design pattern
- когато ползваме динамични структури винаги трябва да мислим за тяхната сериализация и десериализация в / от стринг
- хубаво е да знаем, че сложните динамични структури се обработват бавно - обработващите алгоритми имат експоненциална сложност и отнемат много време и ресурси

CODE ACADEMY

Задачи за самостоятелна работа

1. Напишете функция, с която да разделите даден едносвързан свързан списък от цели числа, в два списъка по следния начин.
   Ако първоначалният списък е L = (l0, l1, ….., ln), то
   новополучените списъци мжгот до са R1 = (l0, l2, l4, …..) и R2 = (l1, l3, l5, …..).

2. Напишете функция, с която да вмъкнете възел „t“ преди възел, посочен с „X“ в единичен свързан списък „L“.

3. Напишете функция, с която да да изтриете възел, посочен с „p“ от едносвързан списък „L“.

4. Да предположим, че подреден списък L = (l0, l1, …..,ln) е представен от единичен свързан списък. Добавете списъка L = (ln, l0, l1, ….., ln) след друг подреден списък M, представен от единично свързан списък

5. Използвайте динамична реализация на стек за въвеждане на поредица от цели положителни числа и нейното извеждане върху екрана в обратен ред. За край на поредицата от клавиатурата се въвежда 0.

CODE ACADEMY

Задачи за самостоятелна работа

6. Дадени с два стека с числа и числата в тях са сортирани във низходящ ред отдолу нагоре.Да се напише програма на С, която слива двата стека в сортиран масив

7. В стек са записани няколко думи, чийто брой не е известен. Да се напише програма, която прочита дума, въведена от клавиатурата и проверявя дали тази дума е записана в стека.

8. Даден е стек от произволни реални числа. Да се напише програма, която изтрива от този стек всички отрицателни числа.

CODE ACADEMY

Задачи за самостоятелна работа

9. Като използвате **стек**, напишете програма, която проверява дали в едни аритметичен израз, записан чрез стринг, на всяка отваряща скоба отговаря затваряща. Проверете за трите вида скоби ( ), [ ], { } .

10. Като използвате **опашка**, отпечатайте първите n числа на Фибоначи.
Напомняне F_0=1, F_1=1, F_2=F_0+F_1=2
или всяка следващо е сбора на предишните две числа

CODE ACADEMY

Задачи за самостоятелна работа

11. Напишете програма, която въвежда едно голямо цяло число, записано в масив и като използвате **стек и опашка** запишете това число в обратен ред, четено отзад напред.

12. Като използвате стек, напишете програма, която прочита едно цяло положително число и преобразува това число в двоична бройна система, т.е. записва го в двоичен вид.