При списъците, k-тият елемент се намира задължително между к-1 и к+1 елементи (ако има такива).
Свързан списък – когато елементите са свързани. Всеки елемент има указател към следващ елемент (с изключение на последния, чиито указател сочи към нищо);
Операции –

1) Запис на елемент в началото на списъка;
2) Запис на елемент след достигнат елемент;
3) Запис на елемент преди достигнат елемент;
4) Премахване на елементи;
5) Изтриване на всички елементи от списък;
6) Обединяване на 2 или повече списъци в 1;
7) Разделяне на свързан списък на 2 или повече;
8) Сортиране на елементите в списък в нарастващ или намаляващ ред;

Възможни са много други операции, за който може да се намери информация.

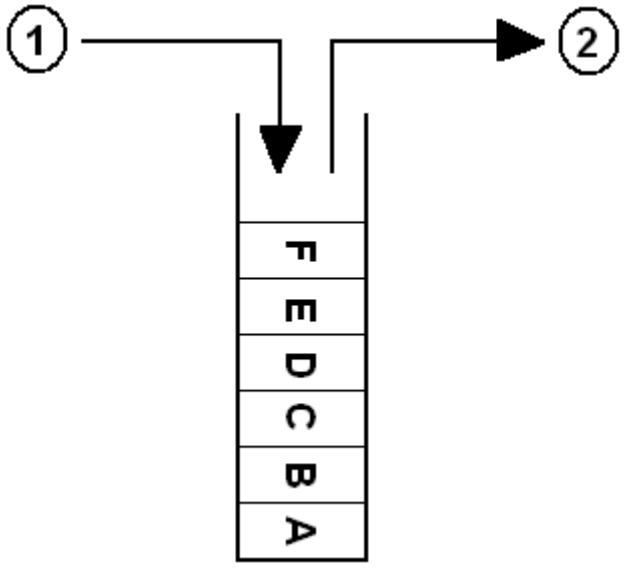
Стек (Stack) и Опашка (Queue) са структури, в които записаната информация се чете отдалечено във времето. В интервала между четенето и записа информацията остава в стека или опашката. Те са базирани на стратегиите LIFO (Last-In-First-Out) и съответно FIFO (First-In-First-Out). Стратегията LIFO се използва при стековете, като за да е по-ясно, можем да си представим стека като една кофа, която се пълни с определени елементи и първият вкаран елемент отива най-долу и за да бъде прочетен е нужно да се прочетат преди това всички над него и съответно последният вкаран елемент е най-отгоре и той ще бъде прочетен първи. Стратегията FIFO е на обратния принцип, като за да си я представим, можем да гледаме на нея като информационен кюнец. Тук първият записан елемент „минава" през кюнеца и при прочитане също се прочита първи. За да се прочете последният записан елемент е нужно преди него да се прочетат всички предхождащи го елементи. Съществува и двустранна опашка (deque), при която четенето и записът могат да се извършват от двете страни, но не едновременно.

Пример:

Опашка. На позиция (1) е източникът на информация а на позиция (2) е приемникът на информация, като те имат различна скорост на предаване и приемане на информацията. Идвайки от източника информацията се натрупва в опашката, ако това е необходимо преди да продължи към приемникът.

Структори като опашките се срещат при предаването на информация от
- компютър към принтер;
- скенер към компютър;
- различни входни устройства (мишки, клавиатури и тн.) към компютър или друга изчислителна машина;
- сензори към други устройства.

Стек: Отново на позиция (1) е източникът на информацията, а на позиция (2) е приемникът на информацията. Информацията при прочитането си се чете от последната записана такава и се чете до изчерпване на стека. Пример за използване на стек е преминаването към машинна аритметика.



Списък:

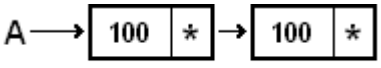
На C++ list <int> L; - В него всеки елемент има поне 2 полета, от които едното е указател към следващия елемент, а другото съдържа информационната стойност. За целта е необходима библиотека list!

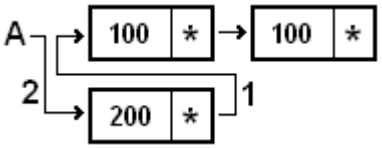
---

1) Запис на елемент в началото на списъка;

```
list<int> mylist (2,100);        // два елемента със стойност 100
  mylist.push_front (200);       // в началото на списъка добавяме
                                 // елемент със стойност 200
```

Последователност по който става това и може да бъде реализирано на различни езици без готови функции.

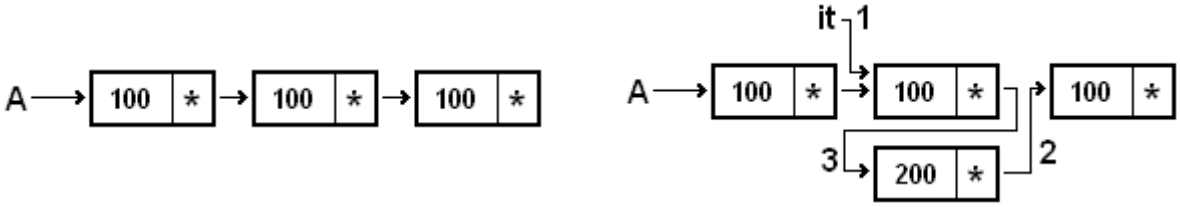




2) Запис на елемент след достигнат елемент;

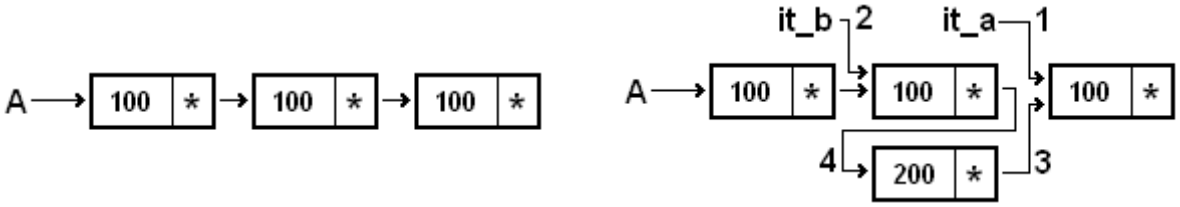
```
list<int> mylist;            // изграждаме списък
list<int>::iterator it;      // изграждаме итератор (указател, пойнтер) към списъка
it = mylist.begin();         // оказваме местоположението от списъка към което
                             // да сочи итератора
++it;                        // премествaме итератора да сочи следващия елемент
```

```
mylist.insert (it,10);      // добавяме новия елемент на сочената позиция
```



1 - Местим итератора до желания елемент; 2 - новия елемент се насочва към елемента след сочения от итератора (ако има такъв); 3 - елемента сочен от итератора се насочва към новия елемент.

3) Запис на елемент преди достигнат елемент;



1 - Местим втория итератор it_b на позицията на първия итератор it_a;

2 - местим първия итератор it_a на следващата позиция; (ако втория итератор е на желаната позиция продължаваме със стъпка 3, ако не е се връщаме на стъпка 1.);

3 - новия елемент се насочва към елемента след сочения от втория итератор (тоест сочения от първия);

4 - елемента сочен от втория итератор се насочва към новия елемент.

4) Премахване на елементи (+картинка от упражненията);

```
it1 = mylist.begin();
++it1;
it1 = mylist.erase (it1); // изтриваме сочения от указателя елемент и
                          // насочваме указателя към следващия елемент
```

1 – насочваме нов указател към елемента който искаме да изтрием; насочваме предходния елемент към следващия елемент; 3 – изтриваме желания елемент.

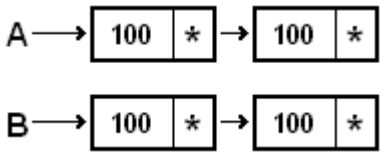
5) Изтриване на всички елементи от списък;

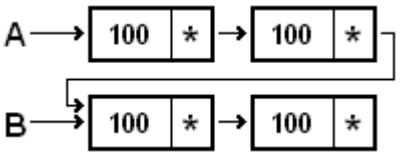
```
mylist.clear();
```

Обхождаме всички елементи на списъка и ги изтриваме по предходния начин внимавайки да не оставим елементи към който нямаме указател и не можем да ги достъпи ме .

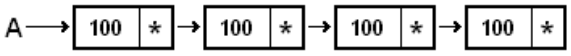
6) Обединяване на 2 или повече списъка в 1;

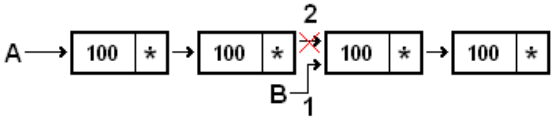
```
list<int> mylist1, mylist2;
list<int>::iterator it;
it = mylist2.begin();
mylist2.splice (it, mylist1);
```





7) Разделяне на списъци;





1 – изграждаме втория списък; 2 задаваме указателя на последния елемент от тези който искаме да останат в първия списък да сочи към нищо.

8) Сортиране на елементите в списък в нарастващ; Това е сложна процедура но в езици като C++ имаме готови функции за това. Пример:

```
mylist.sort();                  // сортиране на всякакви елементи дори
                                // букви по с значение на главните

mylist.sort(compare_nocase);    // сортиране на всякакви елементи дори
                                // букви без значение на главните
```

Различни сайтове с информация.
http://home.netcom.com/~tjensen/ptr/pointers.htm
http://www.cphpvb.net/java/4415-lists-arrays-stacks-queues/
http://www.cphpvb.net/pik-3/301-вектори-stdvector/
http://www.cplusplus.com/reference/
http://msdn.microsoft.com/en-us/library/6yd3wes9(VS.80).aspx
http://cslibrary.stanford.edu/106/
Не се ограничавайте само с тези.