Свързан списък

Лекция 3 по СДА, Софтуерно Инженерство
Зимен семестър 2019-2020г
Милен Чечев

Защо ни трябва структура различна от масив?

Характеристики масив “+”

1. Константна сложност на достъп до всеки един елемент по-индекс
2. Добавяме в края на масива(ако има място) с константна сложност

Характеристики масив “-”

1. Фиксираме размера на масива при създаването му, като ако не е запълнен хабим памет без да я използваме, ако се запълни трябва да създадем нов масив и да копираме.
2. За да премахнем елемент по-средата трябва да преместим всички елементи след него с едно напред.
3. За да добавим елемент трябва да преместим всички елементи след

Списък

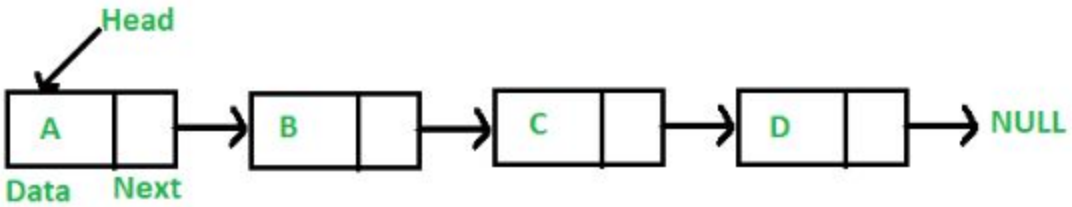
Характеристики “+”:

- Добавянето на елемент в началото и в края е с константна сложност
- Размера е динамичен и за това се ползва памет пропорционална на елементите в него
- Може лесно да добавим или премахнем елемент.

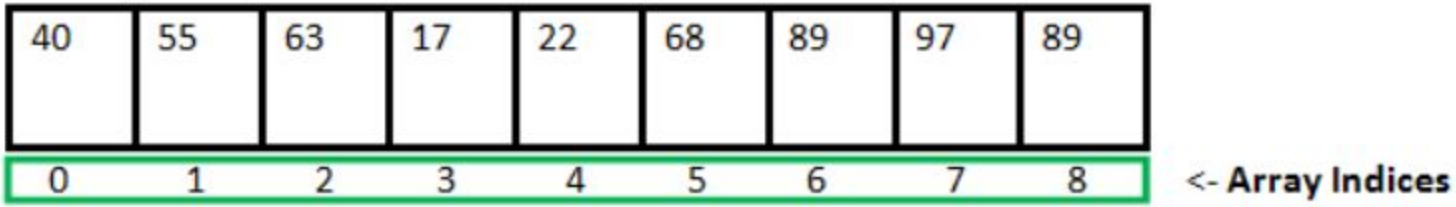
“-”:

- Линеен достъп до елемент по индекс
- Използва допълнителна памет(в сравнение с масив) за съхранение на адресите на следващите клетки.

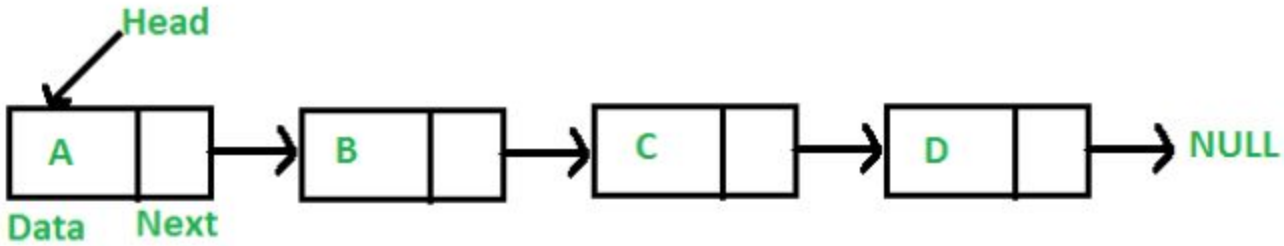
Как изглежда списък?



Сравнение масив и списък



Array Length = 9
First Index = 0
Last Index = 8

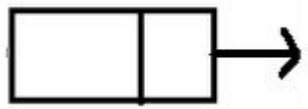


Основни операции

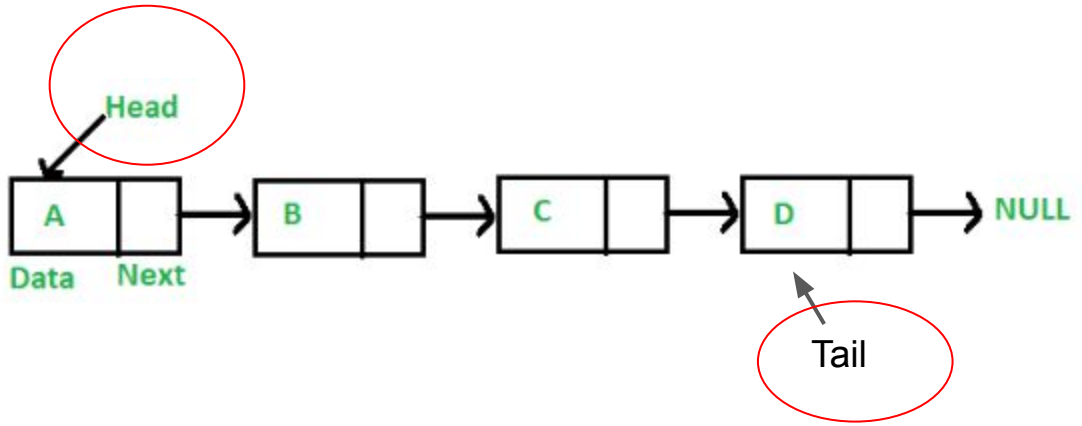
- създаване на празен списък
- добавя елемент x на позиция pos
- изтрива елемент на позиция pos
- търси дали елемент x се среща в списъка

Основен компонент - възел (node)

```
struct Node
{
    int data;
    struct Node *next;
};
```



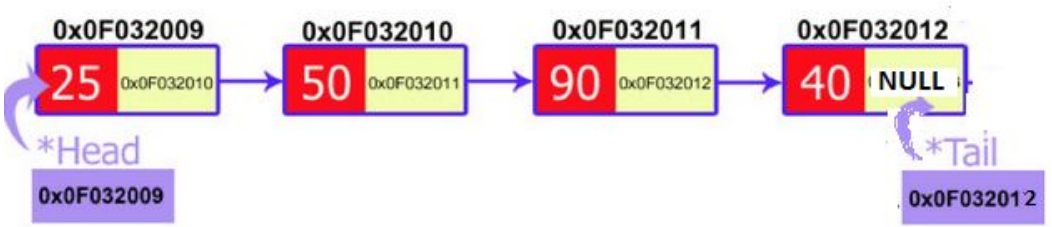
Имайки Възел, какво още ни трябва?

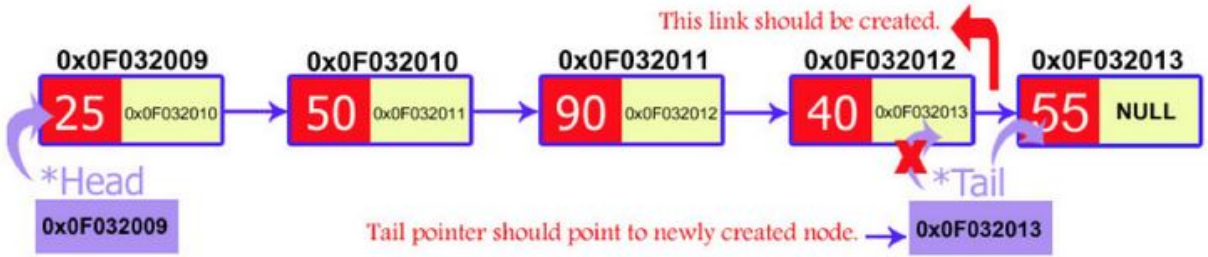


Реализация

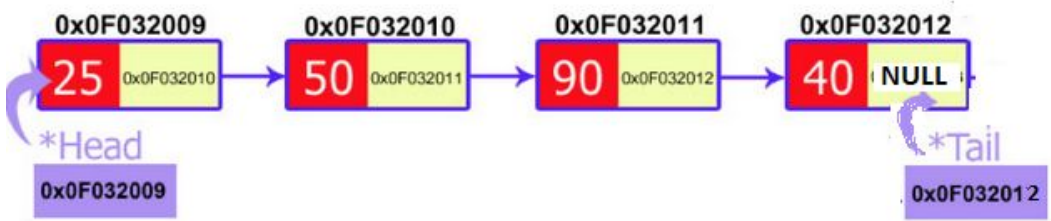
```
class List
  {
        private:
        Node *head, *tail;
        public:
        List()
        {
              head=NULL;
              tail=NULL;
        }
...
};
```

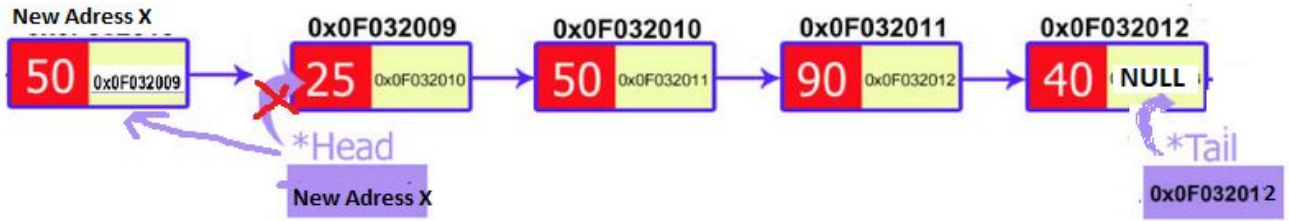
```
void insertAtEnd(int value){
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        if(head==NULL){
                head=temp;
                tail=temp;
        }
        else{
                tail->next=temp;
                tail=temp;
        }
}
```



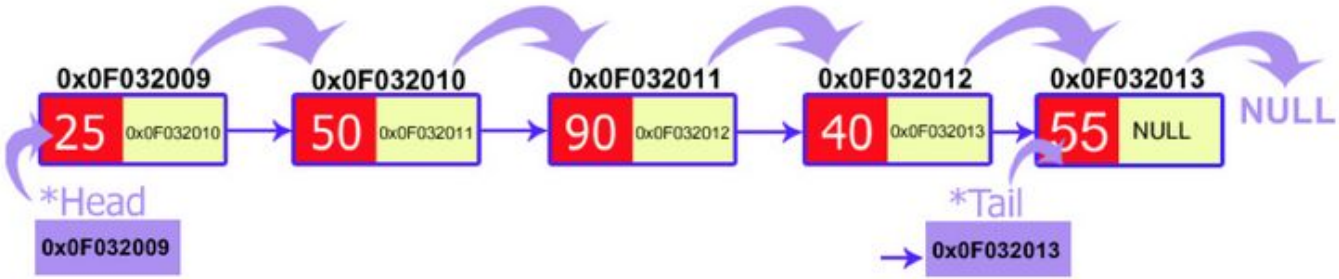


```
void insertAtStart(int value){
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        if(head==NULL){
                head=temp;
                tail=temp;
        }
        else{
                temp->next = head;
                head = temp;
        }
}
```

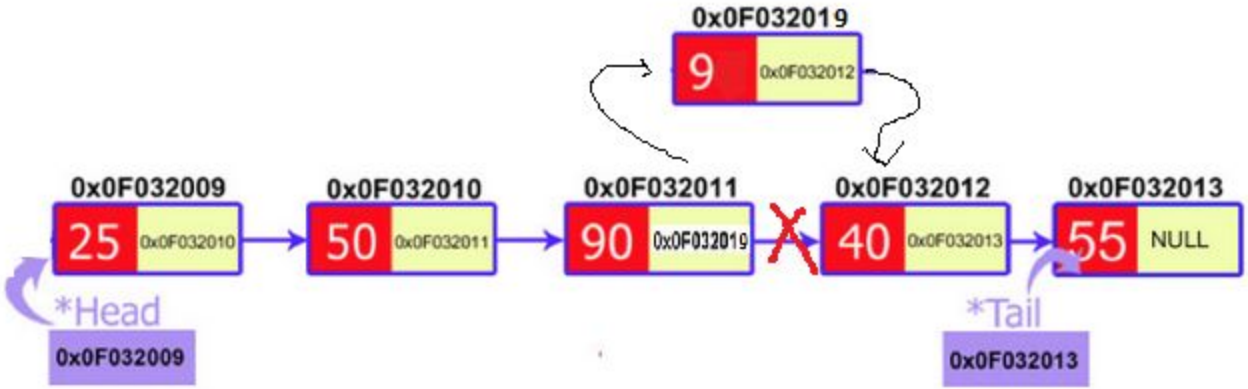




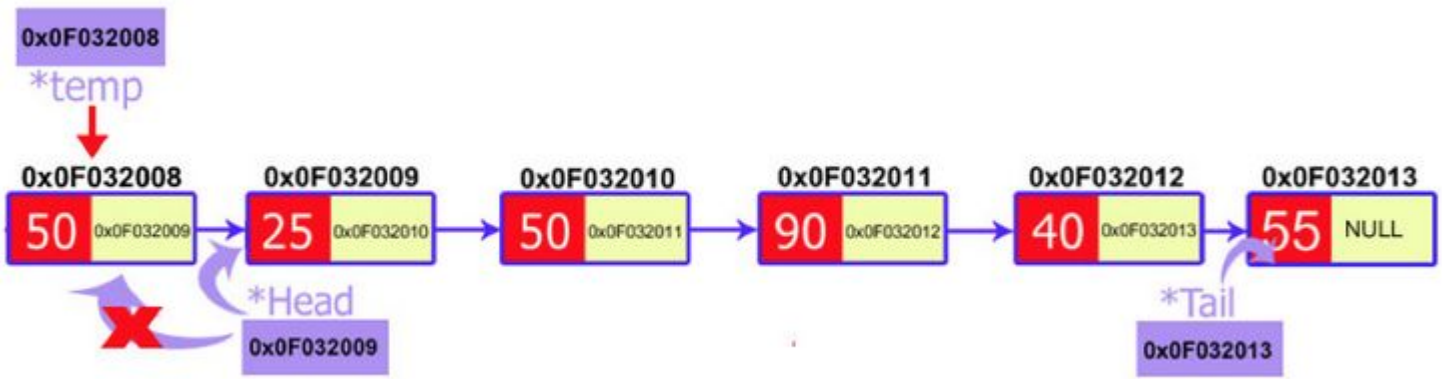
```
void print()
{
    node *temp = head;
    while(temp!=NULL)
    {
        cout<<temp->data<<"\t";
        temp=temp->next;
    }
}
```



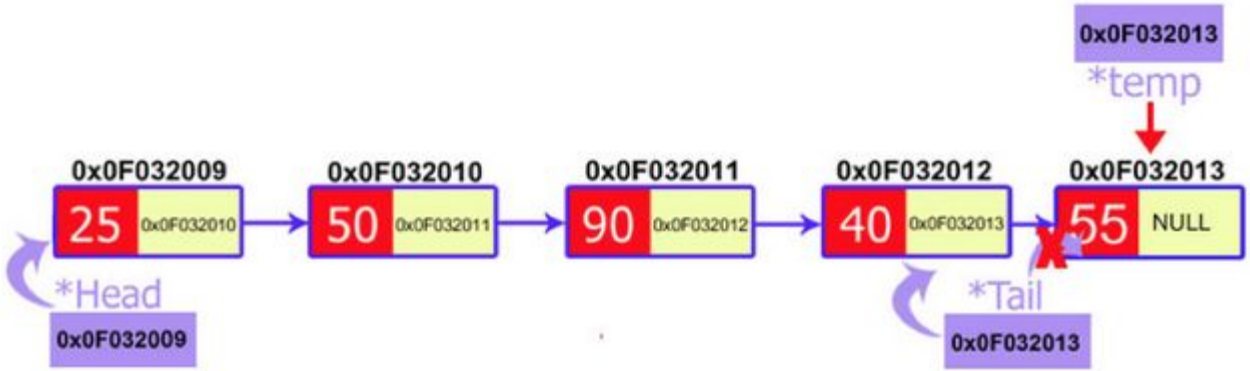
```
void insert_position(int pos, int value){
    node *pre;
    node *cur;
    node *temp=new node;
    temp->data=value;
    cur=head;
    for(int i=1;i<pos;i++) {
        pre=cur;
        cur=cur->next;
    }

    pre->next=temp;
    temp->next=cur;
  }
```



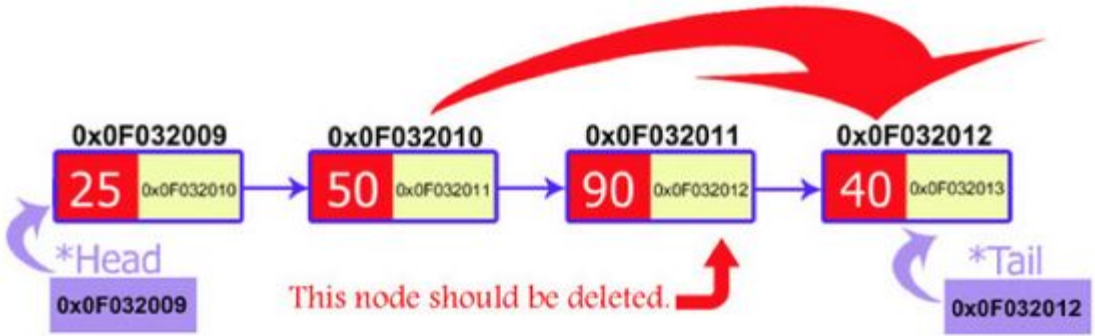
```
void delete_first()
 {
   node *temp;
   temp=head;

  if(head!=null){
   head=head->ne
   delete temp;

   }
 }
```



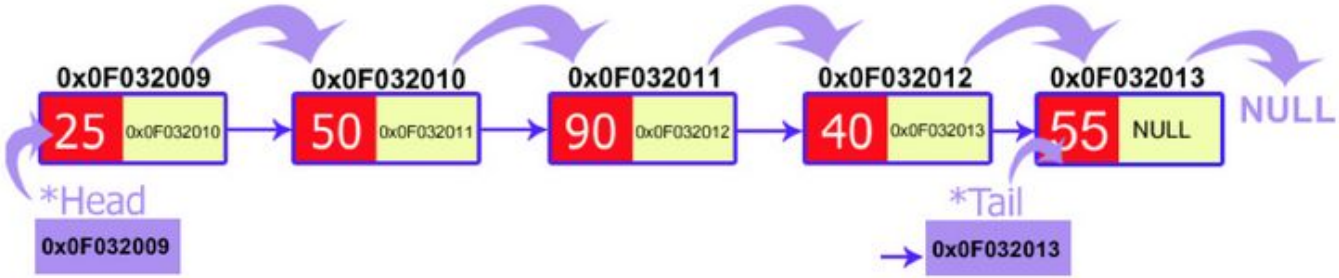
```
void delete_last()
 {
   node *current;
   node *previous;
   current=head;
   while(current->next!=NULL)
   {
     previous=current;
     current=current->next;
   }
   tail=previous;
   previous->next=NULL;
   delete current;
 }
```



```
void delete_position(int pos)
 {
   node *current;
   node *previous;
   current=head;
   for(int i=1;i<pos;i++)
   {
     previous=current;
     current=current->next;
   }
   previous->next=current->next;
 }
```



```
boolean search(int x){
        node* iter = head;
        while(iter!=NULL){
                if(iter->data==x){
                        return true;
                }
                Iter = iter->next;
        }
        return false;
}
```



Анимации на основните операции.

https://visualgo.net/en/list

Задачи

Задача 1:Да се залепи на края на даден списък втори даден списък

Решение 1 O(n)
Добавяме елементите на втория списък на края на първия.

Решение 2 O(1)
Закачаме началото на втория списък за края на първия.

Задача 2: Да се обърне реда на елементите в даден списък.

Сортиране чрез сливане

Задача: Да се раздели даден списък на два други с приблизително равна дължина

Задача: Да се слеят два възходящо подредени списъка в един

Реализации на LinkedList в Github

https://github.com/triffon/sdp-2017-18/blob/master/lists/llist.cpp

std::list

Реализацията на `std::list` в STL е двусвързана.

- `front()`, `back()` — първи и последен елемент
- `begin()`, `end()` — итератори към началото и края
- `rbegin()`, `rend()` — итератори за обратно обхождане
- `push_front()`, `push_back()` — вмъкване в началото/края
- `pop_front()`, `pop_back()` — изтриване от началото/края
- `insert()`, `erase()` — вмъкване/изтриване на позиция
- `splice()` — прехвърляне на елементи от един списък в друг
- `remove()`, `remove_if()` — филтриране по стойност/условие
- `merge()` — сливане на подредени списъци
- `sort()` — сортиране на списък (на място)
- `reverse()` — обръщане на списък
- `==,!=,<,>,<=,>=` — лексикорафско сравнение на два списъка