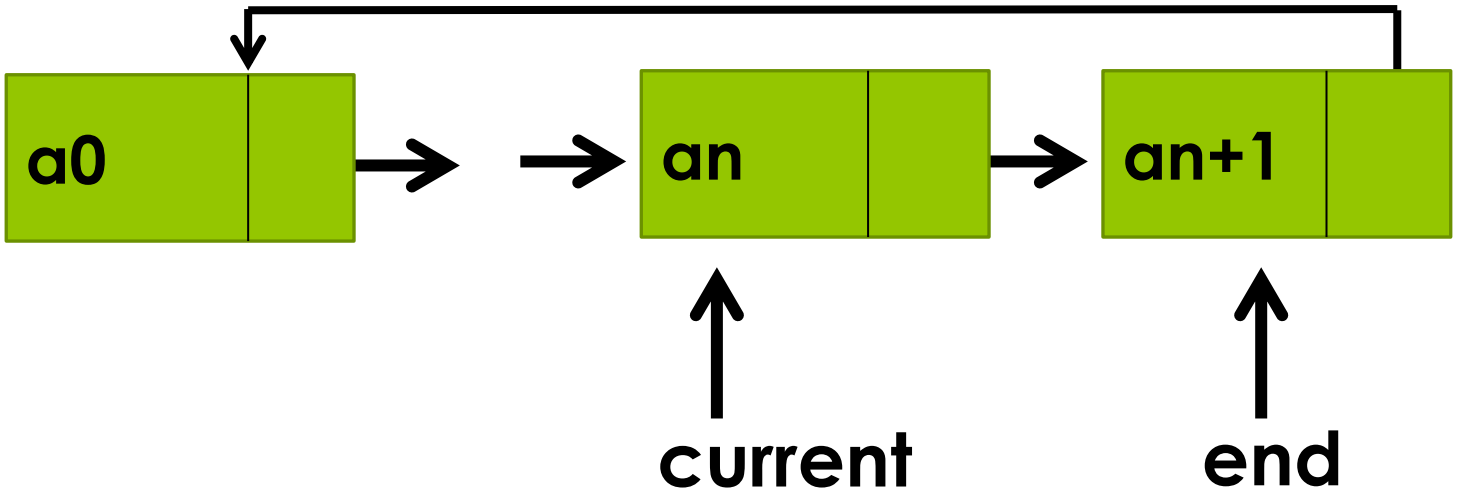
Цикличен свързан списък

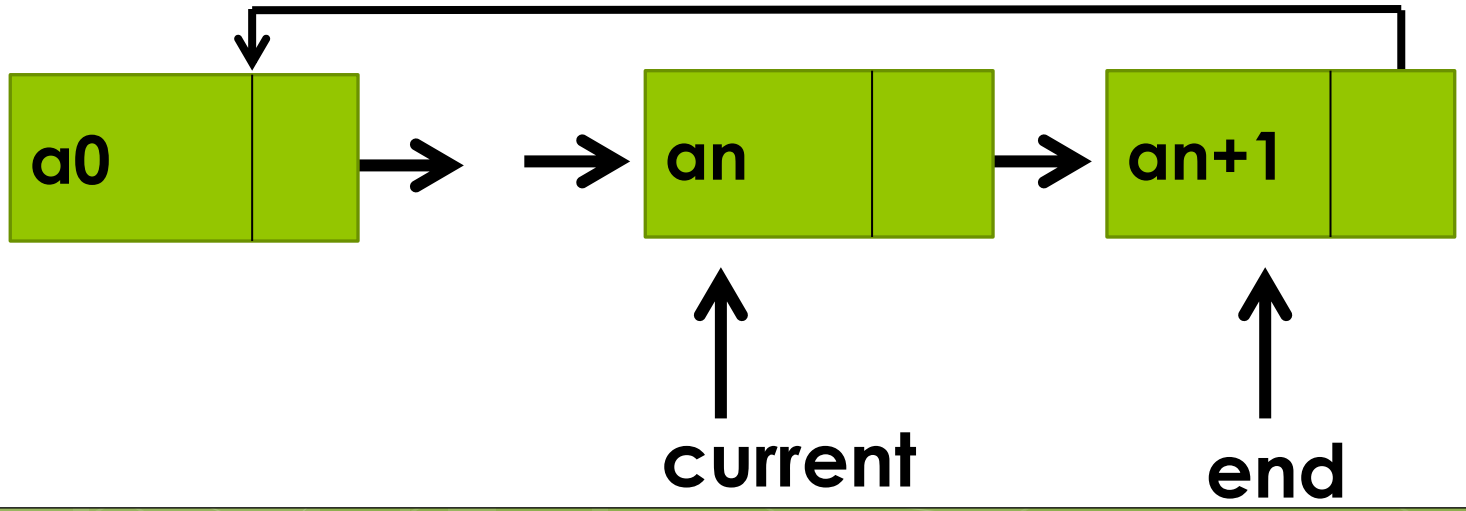
Изготвил:
гл.ас. д-р Нора Ангелова

Цикличен свързан списък



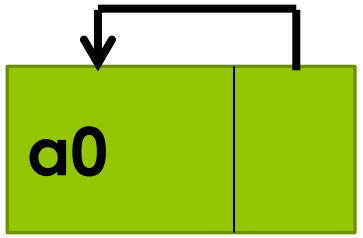
Цикличен свързан списък

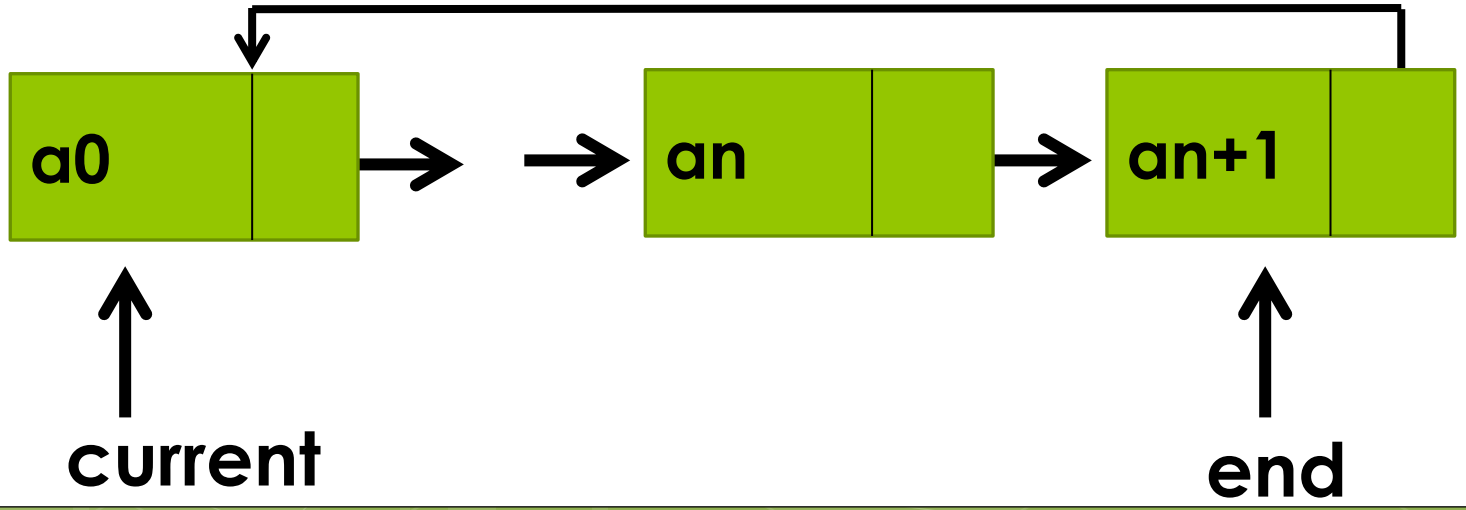
- Ако операцията включване и изключване се осъществява само в края или началото
- Използва се един указател end, указващ последния елемент на списъка
- За улеснение ще използваме и допълнителен указател current.



Реализация

- По подразбиране да създаваме празен списък
- 1 елемент - да сочи сам себе си
- Да можем лесно да обходим всички елементи веднъж. (от първи до последен)





Цикличен свързан списък

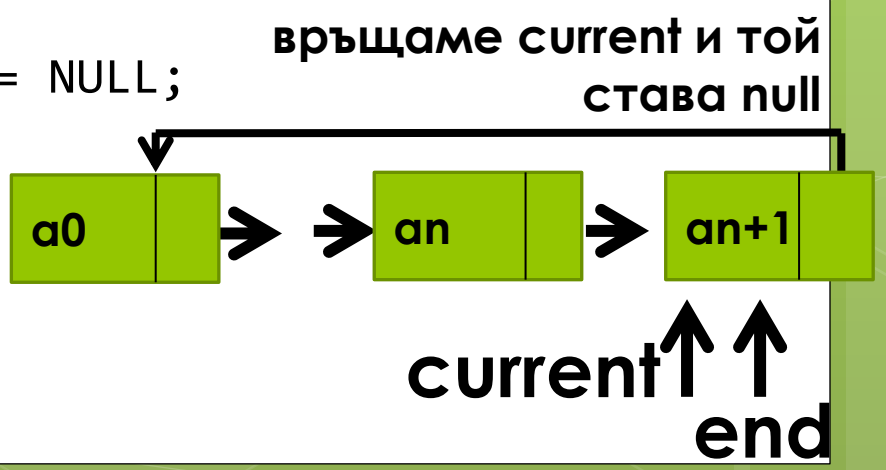
```
template <class T>
struct elem_cir {
  T inf;
  elem_cir<T> *link;
};
```

```
template <class T>
class CirList
{
private:
  elem_cir<T> *end;
  elem_cir<T> *current;

  void DeleteList();
  void CopyList(CirList<T> const &);
```

```
public:
  CirList();
  CirList(CirList<T> const &);
  CirList& operator= (CirList<T> const &);
  ~CirList();

  void IterStart(elem_cir<T> *p = NULL);
  elem_cir<T>* Iter();

  void ToEnd(T const &);
  void DeleteElem(elem_cir<T>*, T &);

  void print();
};
```

```
template <class T>
void CirList<T>::IterStart(elem_cir<T> *p)
{
  if (p) current = p;
  else
    if (!end) current = NULL;
    else current = end->link;
}
```

```
template <class T>
elem_cir<T>* CirList<T>::Iter()
{
  elem_cir<T> *p = current;
  if (current == end) current = NULL;
  else if (current)
       current = current->link;

  return p;
}
```



```
///
/// Removes all elements of a list
///
template <class T>
void CirList<T>::DeleteList()
{
  IterStart();
  elem_cir<T> *p = Iter();
  while(p) {
    delete p;
    p = Iter();
  }
}
```

```
template <class T>
void CirList<T>::CopyList(CirList<T> const & list)
{
  end = NULL;
  elem_cir<T> *p = list.end;
  if (p) {
    p = p->link;
    while(p != list.end) {
      ToEnd(p->inf);
      p = p->link;
    }
    ToEnd(p->inf);
  }
}
```

```
///
/// Creates an empty list
///
template <class T>
CirList<T>::CirList()
{
  end = NULL;
}
```

end →

```
///
/// Destroys a list
///
template <class T>
CirList<T>::~CirList()
{
  DeleteList();
}
```

```
///
/// Copy constructor
///
template <class T>
CirList<T>::CirList(CirList<T> const & list)
{
  CopyList(list);
}
```

```
///
/// Copies the contents of one list to another
///
template <class T>
CirList<T>& CirList<T>::operator=(CirList<T> const & list)
{
  if(this != &list)
  {
    DeleteList();
    CopyList(list);
  }

  return *this;
}
```

```
template <class T>
void CirList<T>::ToEnd(T const & x)
{
  elem_cir<T> *p = new elem_cir<T>;
  p->inf = x;

  if (end) p->link = end->link;
  else end = p;

  end->link = p;
  end = end->link;
}
```

```
template <class T>
void CirList<T>::DeleteElem(elem_cir<T> *p, T & x)
{
  x = p->inf;
  if (end != end->link) {
    elem_cir<T> *q = end;
    while(q->link != p) q=q->link;
    q->link = p->link;
    if (p == end) end = q;
    delete p;
  }
  else {
    end = NULL;
    delete p;
  }
}
```

```
template <class T>
void CirList<T>::print()
{
  IterStart();
  elem_cir<T>* p = Iter();
  while(p) {
    cout << p->inf << " ";
    p = Iter();
  }

  cout << endl;
}
```

```
CirList<int> list; int x;
list.ToEnd(1);
list.ToEnd(2);
list.ToEnd(3);
list.ToEnd(4);

list.print(); // 1 2 3 4

//---------------------------------

list.IterStart();
elem_cir<int> *p = list.Iter();
list.DeleteElem(p, x);
list.print(); // 2 3 4
```

Кой е по-по-най

Даден е свързан списък от цели числа. Да се напише функция, която изтрива първото срещане на дадено число.

```
typedef LList<int> IntList;
void deleteElem(int a, IntList& list) {
  int x;
  list.IterStart();
  elem_link<int> *p = list.Iter();
  while(p && p->inf != a) p = list.Iter();
  if (p && p->inf == a) list.DeleteElem(p,x);
}
```

Кой е по-по-най

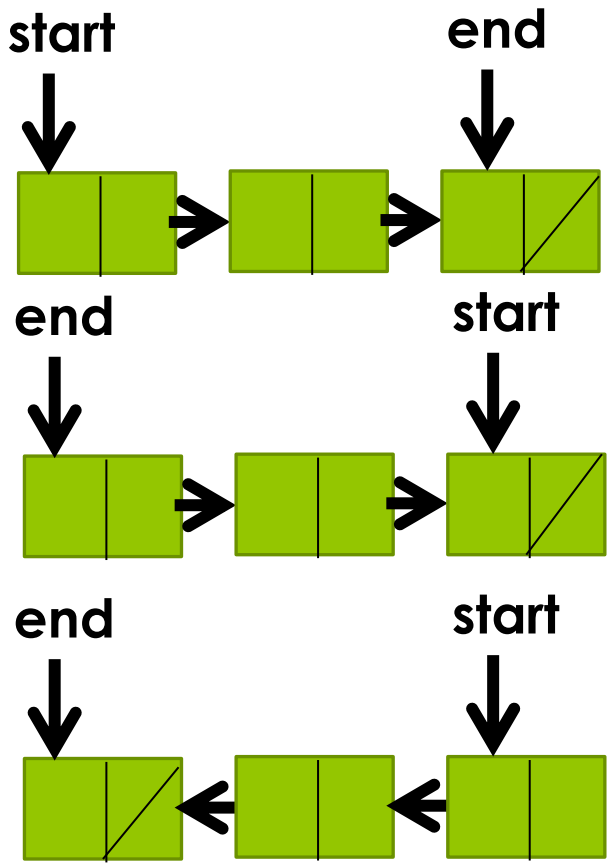
Даден е свързан списък от цели числа. Да се напише функция, която изтрива всяко срещане на дадено число.

```
typedef LList<int> IntList;
void deleteElem(int a, IntList& list) {
  int x;
  list.IterStart();
  elem_link<int> *p = list.Iter();
  while(p) {
    if (p->inf == a) {
      list.DeleteElem(p,x);
    }
    p = list.Iter();
  }
}
```

Кой е по-по-най

Да се напише член-функция на класа LList – reverse(), която обръща елементите на свързан списък. Обръщането да се извършва без създаване на ново копие в паметта.

```
template <class T>
void LList<T>::reverse() {
  elem_link<T> *cur, *prev, *temp;
  cur = start;
  if (cur) {
    prev = NULL;
    temp = start;
    start = end;
    end = temp;
    while(cur != start) {
      temp = cur->link;
      cur->link = prev;
      prev = cur;
      cur = temp;
    }
    cur->link = prev;
  }
}
```



Кой е по-по-най

Да се напише рекурсивна функция, която извежда в обратен ред елементите на свързан списък от цели числа.

```
typedef LList<int> IntList;
void print_reverse(IntList &list, elem_link<int> *p) {
  if(!p) return;
  print_reverse(list, p->link);
  cout << p->inf;
}
...
IntList list;
list.ToEnd(1);
list.ToEnd(2);
list.ToEnd(3);
list.IterStart();
print_reverse(list, list.Iter());
```

Кой е по-по-най

Да се напише рекурсивна функция, която извежда в обратен ред елементите на свързан списък от цели числа.

```
void print_reverse(IntList list) {
  int x;
  list.IterStart();
  elem_link<int> *p = list.Iter();
  if (p) {
    list.DeleteElem(p, x);
    print_reverse(list);
    cout << x << " ";
  }
}
```

```
cout << “КРАЙ”;
```