

Цикличен свързан списък

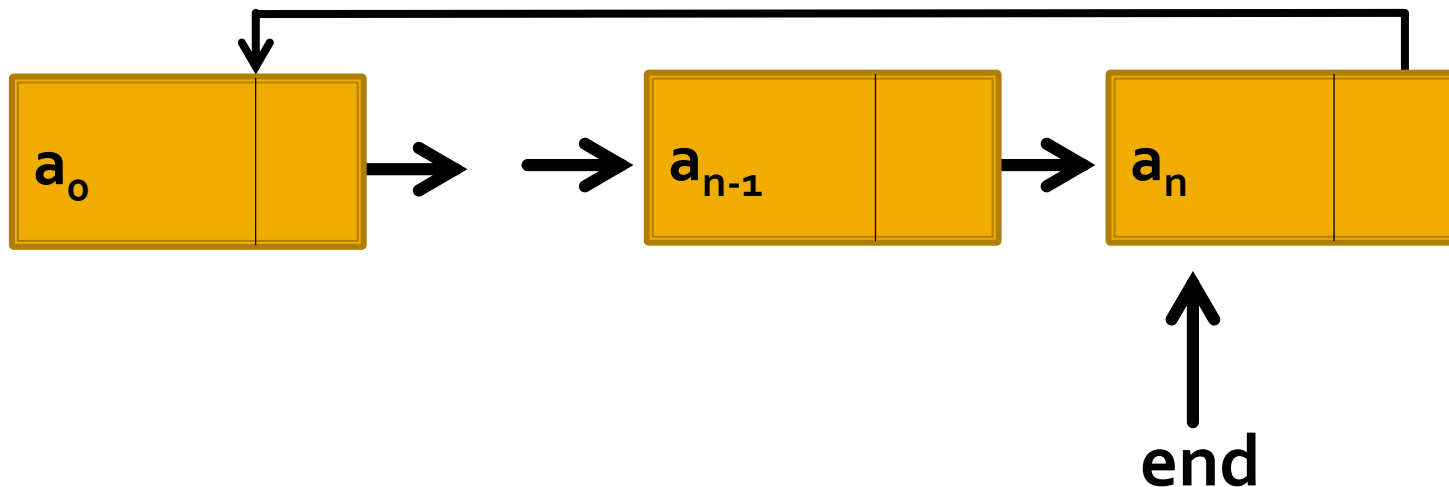
доц. д-р Нора Ангелова

Циклический связанный список

- Хомогенная линейная структура от данных
- Последний элемент на связанный список содержит ссылку на первый элемент на связанный список
- Обход элементов?
- Посещение элементов может происходить многократно

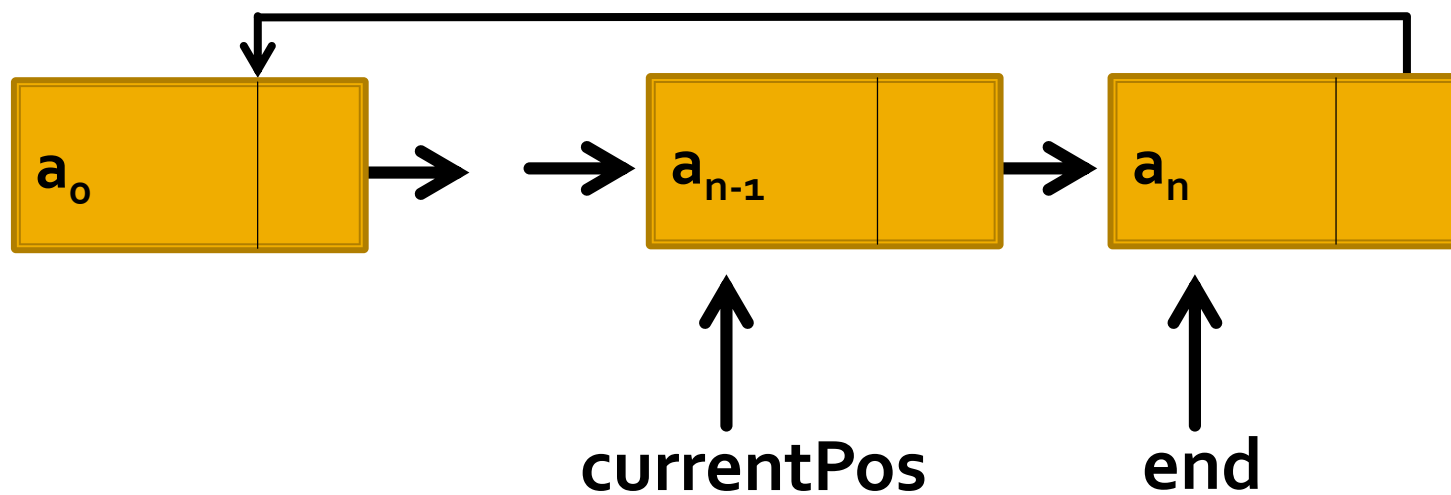
Циклический связанный список

- Графическое представление
- Возможна реализация с одним указателем
- Удобно, если указать последний элемент на связанный список



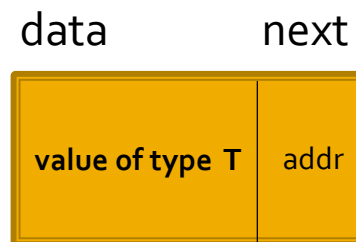
Циклический связанный список

- Обход элементов



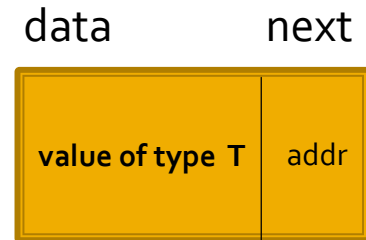
Циклический связанный список

Реализация от учебника



Циклический связанный список

```
template <typename T>  
struct CListElement {  
    T data;  
    CListElement<T> *next;  
};
```



Свързан списък с две връзки

```
template <typename T>
class CList {
private:
    CListElement<T>* end;
    CListElement<T>* currentPos; // Може да се реализира с итератор

    void deleteList();
    void copyList(CList<T> const &);
public:
    CList();
    CList(CList<T> const &);
    CList& operator= (CList<T> const &);
    ~CList();

    void iterStart(CListElement<T>* elemPtr = NULL);
    CListElement<T>* iter();

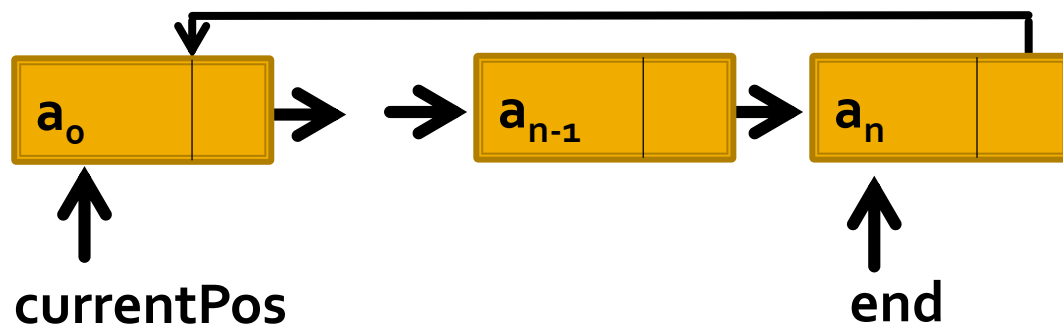
    void insertToEnd(T const &);
    void deleteElem(CListElement<T> *, T &);

    void print();
};
```

Циклический связанный список

```
template <typename T>
void CList<T>::iterStart(CListElement<T>* elemPtr) {
    if (elemPtr) {
        currentPos = elemPtr;
        return;
    }

    if (end) {
        currentPos = end->next;
    } else {
        currentPos = NULL;
    }
}
```

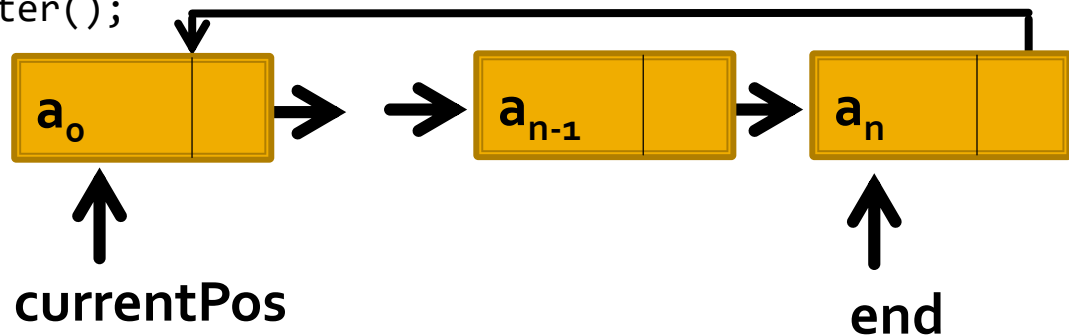


```
template <typename T>
CListElement<T>* CList<T>::iter() {
    CListElement<T>* temp = currentPos;

    if (currentPos == end) {
        currentPos = NULL; // Реализира обхождане - посещава елементите точно 1
    }
    else if (currentPos) {
        currentPos = currentPos->next;
    }
    return temp;
}
```


Циклический связанный список

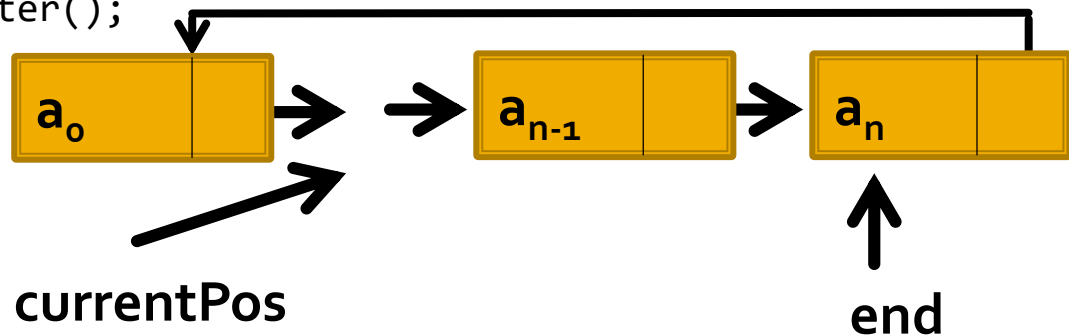
```
template <typename T>
void CList<T>::deleteList() {
    iterStart();
    CListElement<T>* elemPtr = iter();
    while (elemPtr) {
        delete elemPtr;
        elemPtr = iter();
    }
}
```



```
template <typename T>
void CList<T>::copyList(CList<T> const & list) {
    end = NULL;
    CListElement<T>* elemPtr = list.end;
    if (elemPtr) {
        elemPtr = elemPtr->next;
        while (elemPtr != list.end) {
            insertToEnd(elemPtr->data);
            elemPtr = elemPtr->next;
        }
        insertToEnd(elemPtr->data);
    }
}
```

Циклический связанный список

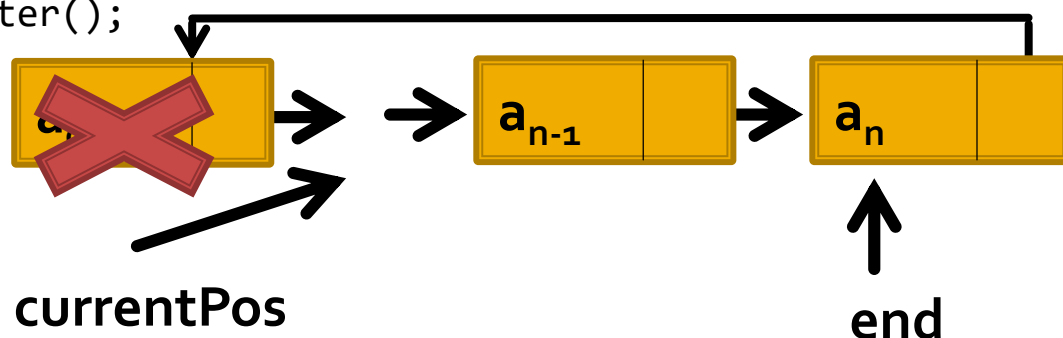
```
template <typename T>
void CList<T>::deleteList() {
    iterStart();
    CListElement<T>* elemPtr = iter();
    while (elemPtr) {
        delete elemPtr;
        elemPtr = iter();
    }
}
```



```
template <typename T>
void CList<T>::copyList(CList<T> const & list) {
    end = NULL;
    CListElement<T>* elemPtr = list.end;
    if (elemPtr) {
        elemPtr = elemPtr->next;
        while (elemPtr != list.end) {
            insertToEnd(elemPtr->data);
            elemPtr = elemPtr->next;
        }
        insertToEnd(elemPtr->data);
    }
}
```

Циклический связанный список

```
template <typename T>
void CList<T>::deleteList() {
    iterStart();
    CListElement<T>* elemPtr = iter();
    while (elemPtr) {
        delete elemPtr;
        elemPtr = iter();
    }
}
```



```
template <typename T>
void CList<T>::copyList(CList<T> const & list) {
    end = NULL;
    CListElement<T>* elemPtr = list.end; // Использва вътрешното представяне
    if (elemPtr) {
        elemPtr = elemPtr->next;
        while(elemPtr != list.end) {
            insertToEnd(elemPtr->data);
            elemPtr = elemPtr->next;
        }
        insertToEnd(elemPtr->data);
    }
}
```

Циклический связанный список

```
template <typename T>
CList<T>::CList() {
    end = NULL;
}
```

```
template <typename T>
CList<T>::~~CList() {
    deleteList();
}
```

```
template <typename T>
CList<T>::CList(CList<T> const & list) {
    copyList(list);
}
```

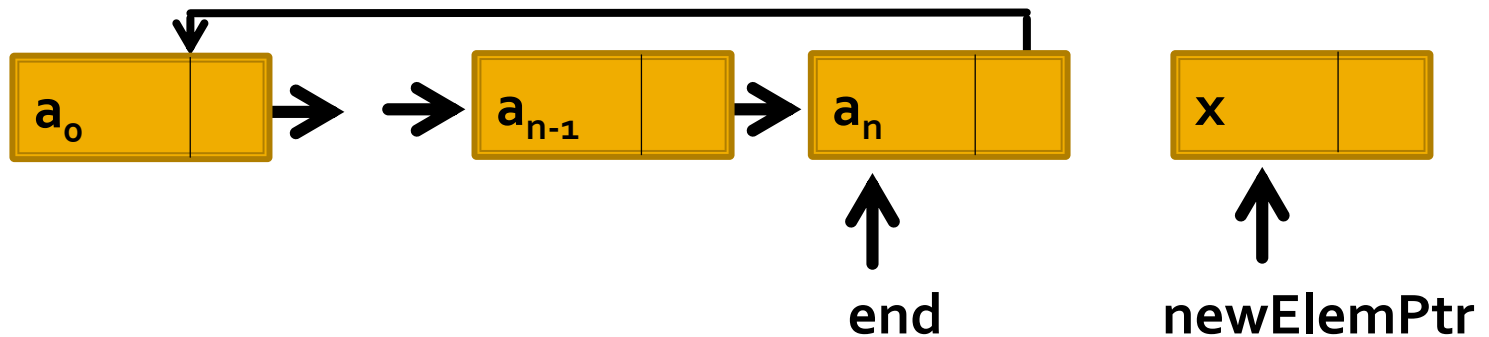
```
template <typename T>
CList<T>& CList<T>::operator=(CList<T> const & list) {
    if(this != &list) {
        deleteList();
        copyList(list);
    }
    return *this;
}
```

Циклический связанный список

```
template <typename T>
void CList<T>::insertToEnd(T const & x) {
    CListElement<T> *newElemPtr = new CListElement<T>;
    newElemPtr->data = x;

    if (end) {
        newElemPtr->next = end->next;
    }
    else {
        end = newElemPtr;
    }

    end->next = newElemPtr;
    end = end->next;
}
```

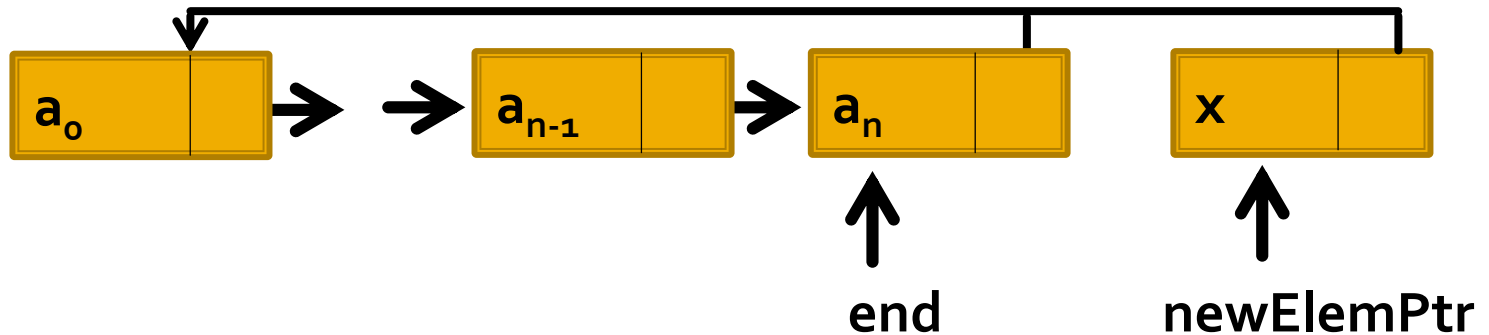


Циклический связанный список

```
template <typename T>
void CList<T>::insertToEnd(T const & x) {
    CListElement<T> *newElemPtr = new CListElement<T>;
    newElemPtr->data = x;

    if (end) {
        newElemPtr->next = end->next;
    }
    else {
        end = newElemPtr;
    }

    end->next = newElemPtr;
    end = end->next;
}
```

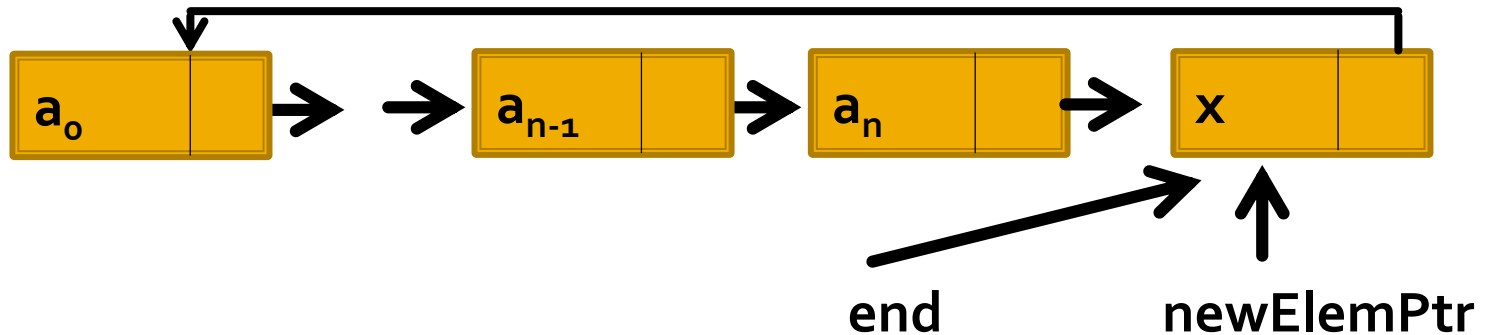


Циклический связанный список

```
template <typename T>
void CList<T>::insertToEnd(T const & x) {
    CListElement<T> *newElemPtr = new CListElement<T>;
    newElemPtr->data = x;

    if (end) {
        newElemPtr->next = end->next;
    }
    else {
        end = newElemPtr;
    }

    end->next = newElemPtr;
    end = end->next;
}
```

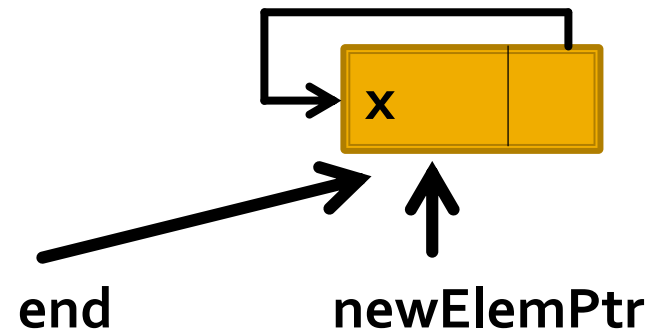


Циклический связанный список

```
template <typename T>
void CList<T>::insertToEnd(T const & x) {
    CListElement<T> *newElemPtr = new CListElement<T>;
    newElemPtr->data = x;

    if (end) {
        newElemPtr->next = end->next;
    }
    else {
        end = newElemPtr;
    }

    end->next = newElemPtr;
    end = end->next;
}
```



Циклический связанный список

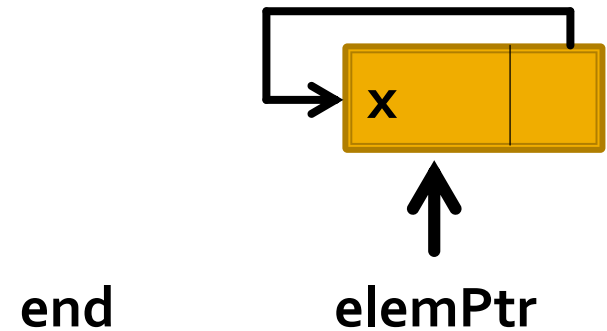
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



Циклический связанный список

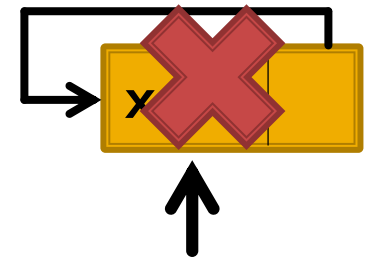
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



end

elemPtr

Циклический связанный список

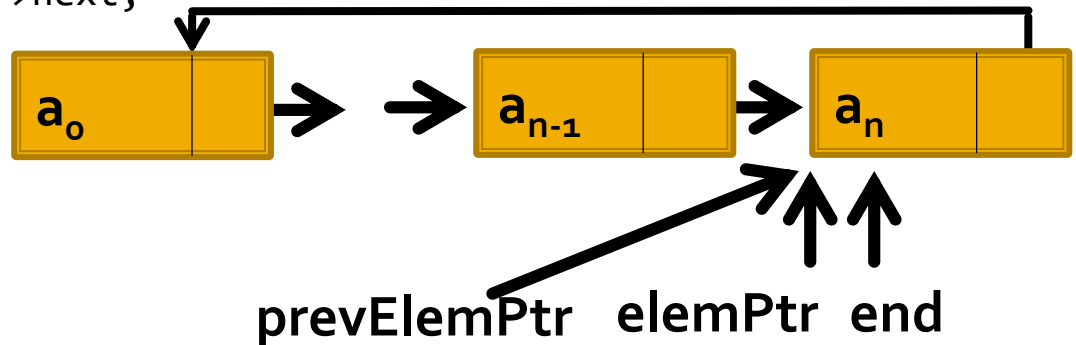
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



Циклический связанный список

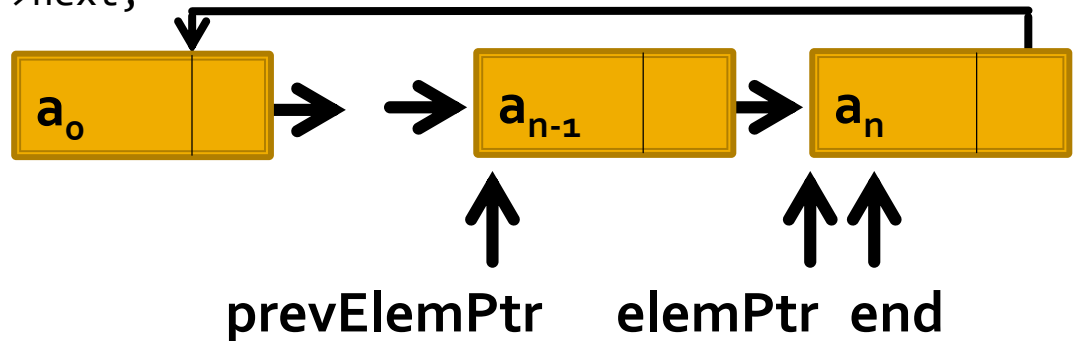
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



Циклический связанный список

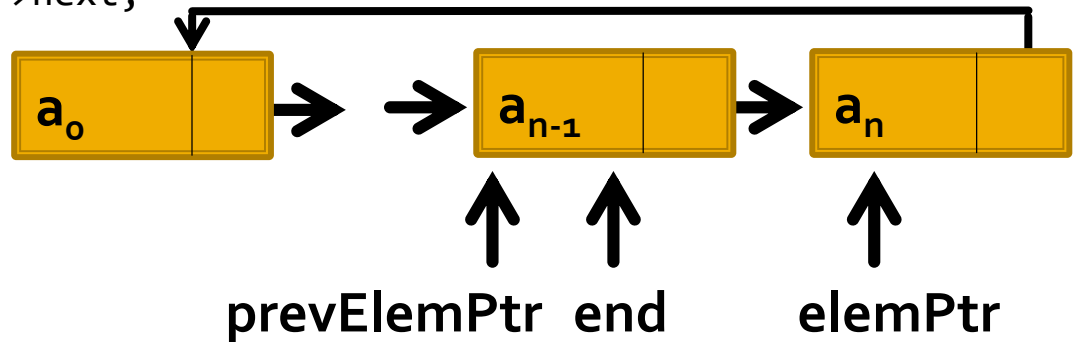
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



Циклический связанный список

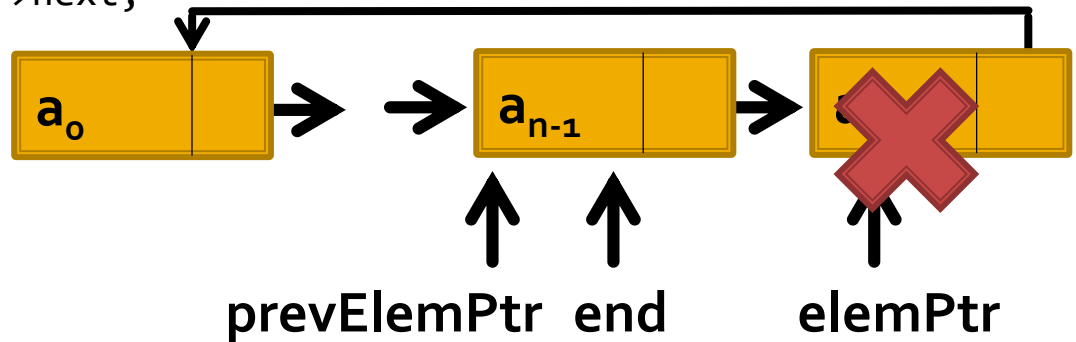
```
template <typename T>
void CList<T>::deleteElem(CListElement<T>* elemPtr, T & x) {
    x = elemPtr->data;

    if (end == end->next) {
        end = NULL;
        delete elemPtr;
        return;
    }

    CListElement<T>* prevElemPtr = end;
    while (prevElemPtr->next != elemPtr) {
        prevElemPtr = prevElemPtr->next;
    }

    prevElemPtr->next = elemPtr->next;
    if (elemPtr == end) {
        end = prevElemPtr;
    }

    delete elemPtr;
}
```



Циклический связанный список

```
template <typename T>
void CList<T>::print() {
    iterStart();
    CListElement<T>* elemPtr = iter();
    while (elemPtr) {
        std::cout << elemPtr->data << " ";
        elemPtr = iter();
    }

    std::cout << endl;
}
```

Циклический связанный список

Реализация с итератором – аналогично на связанный список с одной ссылкой



Циклически свързан списък

Задачата на Йосиф Флавий

Последният отряд от 41 сикарии защитава Галилейската крепост Масада. Сикариите не искат да се предадат на римския X Железен легион. Сикариите нямат изход в създалата се ситуация, освен да се самоубият. Тъй като според юдаизма това е особено тежък грях, трябва да измислят начин, по който да го избегнат.

За целта Йосиф Флавий предложил да се наредят един до друг в кръг и всеки трети сикарий да бъде убиван от втория, стоящ до него.

Цикличен свързан списък

Йосиф твърди, че по божията воля станало така, че той и неговия приятел Яков останали последните двама живи, комуто се паднала честта да предадат крепостта и били помилвани от римляните, но малцина са склонни да му вярват.

Задачата на Йосиф Флавий се състои в това да се намери най-лесният начин да се открият двете позиции в кръга, които ще останат непокътнати, ако е известен броят на участниците в него - 41.

Отговор: 16 и 31

Циклически свързан списък

Задача.

Дадени са естествените числа n и m . Предполага се, че m човека са наредени в кръг и всеки от тях е получил пореден номер от 1 до m (бройки в посока обратна на часовниковата стрелка). Започвайки от първия елемент в посока обратна на часовниковата стрелка, се отброява n -тият човек и се отстранява от кръга. Отново започвайки от следващия човек ($n+1$ -вия), се отброява отново n -тият човек и се отстранява. Този процес продължава до отстраняване на всички хора от кръга. Да се напише програма, която извежда номерата на отстранените хора в реда на отстраняването им.

Цикличен свързан списък

```
typedef CList<int> IntCList;  
  
void create (int m, IntCList& list) {  
    for (int element = 1; element <= m; element++) {  
        list.insertToEnd(element);  
    }  
}
```

Циклический связанный список

```
void josiffTask(int n, IntCList list) {
    list.iterStart();
    CListElement<int>* curr = list.iter();
    CListElement<int>* deleteElemPtr;
    while (curr != curr->next) {
        deleteElemPtr = curr;
        for (int index = 1; index <= n-1; index++) {
            deleteElemPtr = deleteElemPtr->next;
        }
        curr = deleteElemPtr->next;
        int x;
        list.deleteElem(deleteElemPtr, x);
        std::cout << x << " ";
    }

    std::cout << curr->data << endl;
}
```

Цикличен свързан списък

```
int main() {  
    intCList list;  
    create(41, list);  
    list.print();  
    josiffTask(3, list);  
  
    return 0;  
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41  
3 6 9 12 15 18 21 24 27 30 33 36 39 1 5 10 14 19 23 28 32 37 41 7 13 20 26 34 40 8 17 29 38 11 25 2 22 4 35 16 31  
Press any key to continue . . . _
```

Следва продължение...