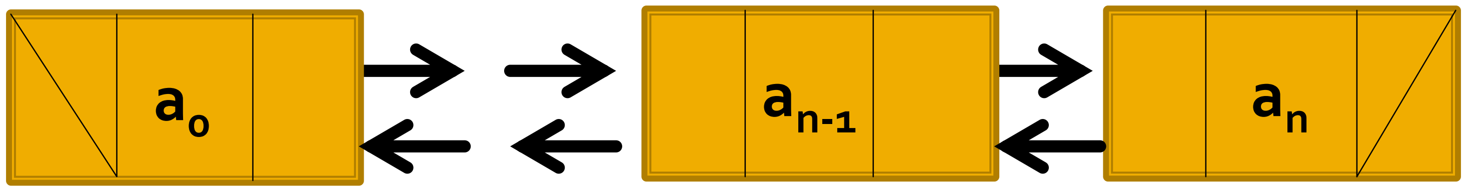
Свързан списък с две връзки

доц. д-р. Нора Ангелова

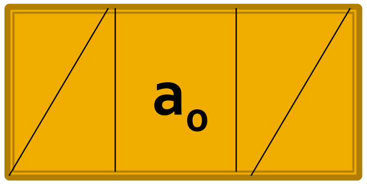
Свързан списък с две връзки

- Физическо представяне
- Логическо представяне



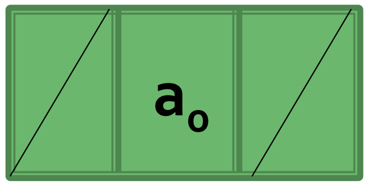
Свързан списък с две връзки

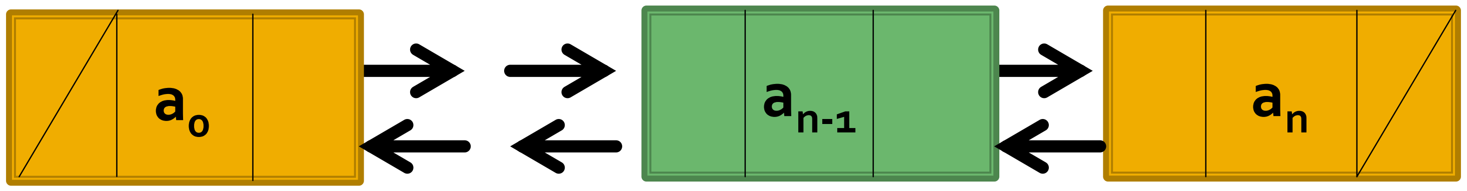
- Добавяне на елемент



Свързан списък с две връзки

- Добавяне на елемент

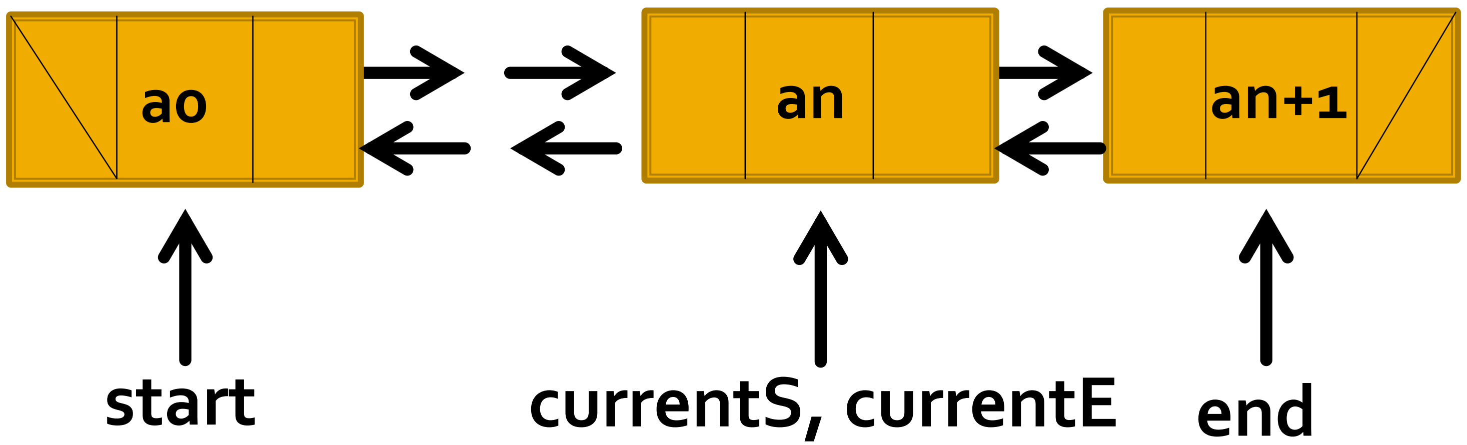




Свързан списък с две връзки

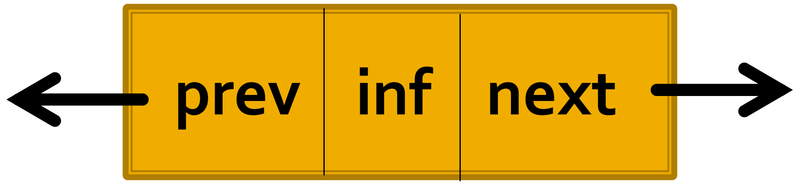
Реализация с указатели

- Елементи с три полета



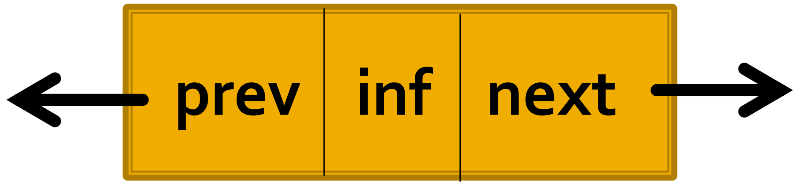
Свързан списък с две връзки

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



Свързан списък с две връзки

Реализация от учебника

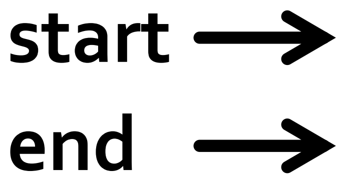


Свързан списък с две връзки

```
template <typename T>
class DList {
private:
  DListElement<T> *start;
  DListElement<T> *end;
  DListElement<T> *currentS;
  DListElement<T> *currentE;

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

  void iterStart(DListElement<T> *elemPtr = NULL);
  DListElement<T>* iterNext();

  void iterEnd(DListElement<T> *elemPtr = NULL);
  DListElement<T>* iterPrev();

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

  void print();
  void print_reverse();
  int length();      //може да бъде const (разглеждаме дословно реализацията)
};
```

Свързан списък с две връзки

```
// Конструктор по подразбиране
template <typename T>
DList<T>::DList() {
  start = NULL;
  end = NULL;
}
```



Свързан списък с две връзки

```
// Деструктор
template <typename T>
DList<T>::~DList() {
  deleteList();
}
```

Свързан списък с две връзки

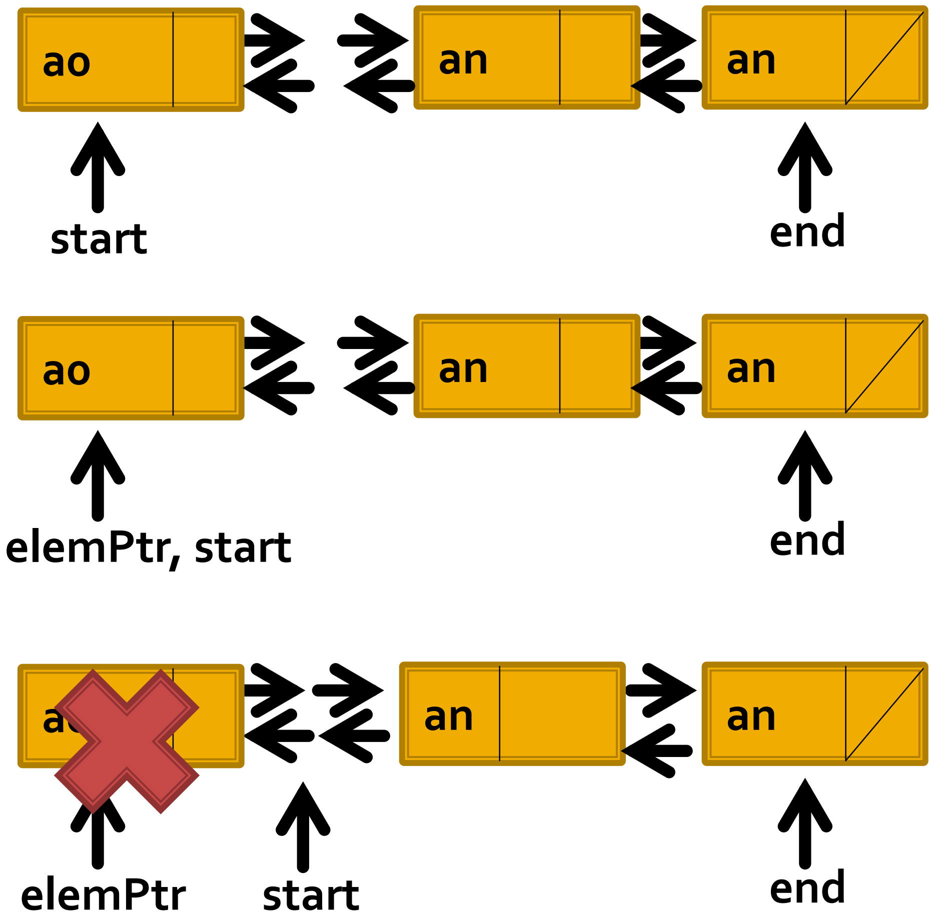
```
// Конструктор за копиране
template <typename T>
DList<T>::DList(DList<T> const & list) {
  copyList(list);
}
```

Свързан списък с две връзки

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

Свързан списък с две връзки

```
// Изтриване на списък
template <typename T>
void DList<T>::deleteList() {
  DListElement<T> *elemPtr;
  while (start) {
    elemPtr = start;
    start = start->next;
    delete elemPtr;
  }

  end = NULL;
}
```



Свързан списък с две връзки

```
// Копиране на елементите на списък
template <typename T>
void DList<T>::copyList(DList<T> const & list) {
  start = end = NULL;
  if (list.start) {
    DListElement<T> *elemPtr = list.start;
    while (elemPtr) {
      insertToEnd(elemPtr->data);
      elemPtr = elemPtr->next;
    }
  }
}
```

Свързан списък с две връзки

```
template <typename T>
void DList<T>::iterStart(DListElement<T> *elemPtr) {
  if (elemPtr) {
    currentS = elemPtr;
  } else {
    currentS = start;
  }
}


template <typename T>
DListElement<T>* DList<T>::iterNext() {
  DListElement<T> *temp = currentS;
  if (currentS) {
    currentS = currentS->next;
  }
  return temp;
}
```

Свързан списък с две връзки

```
template <typename T>
void DList<T>::iterEnd(DListElement<T> *elemPtr) {
  if (elemPtr) {
    currentE = elemPtr;
  } else {
    currentE = end;
  }
}


template <typename T>
DListElement<T>* DList<T>::iterPrev() {
  DListElement<T> *temp = currentE;
  if (currentE) {
    currentE = currentE->prev;
  }
  return temp;
}
```

Свързан списък с две връзки

```
template <typename T>
void DList<T>::insertToEnd(T const & x) {
  DListElement<T> *newElemPtr = new DListElement<T>;
  newElemPtr->data = x;
  newElemPtr->next = NULL;

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

Свързан списък с две връзки

```
template <typename T>
void DList<T>::deleteElem(DListElement<T> *delElemPtr, T & x) {
  x = delElemPtr->data;
  if (start == end) {
    start = NULL;
    end = NULL;
  }
  else if (delElemPtr == start) {
    start = start->next;
    start->prev = NULL;
  }
  else if (delElemPtr == end) {
    end = delElemPtr->prev;
    end->next = NULL;
  }
  else {
    delElemPtr->prev->next = delElemPtr->next;
    delElemPtr->next->prev = delElemPtr->prev;
  }
  delete delElemPtr;
}
```

Свързан списък с две връзки

```
template <typename T>
void DList<T>::print() {
  DListElement<T>* elemPtr = start;
  while (elemPtr) {
    cout << elemPtr->data << " ";
    elemPtr = elemPtr->next;
  }

  cout << endl;
}
```

Свързан списък с две връзки

```
template <typename T>
void DList<T>::print_reverse() {
  DListElement<T>* elemPtr = end;
  while (elemPtr) {
    cout << elemPtr->data << " ";
    elemPtr = elemPtr->prev;
  }

  cout << endl;
}
```

Свързан списък с две връзки

```
template <typename T>
int DList<T>::length() {
  int n = 0;
  DListElement<T>* elemPtr = start;
  while (elemPtr) {
    n++;
    elemPtr = elemPtr->next;
  }

  return n;
}
```

Свързан списък с две връзки

```
int main() {
  DList<int> list; int x;
  list.insertToEnd(1);
  list.insertToEnd(2);
  list.insertToEnd(3);
  list.insertToEnd(4);

  list.iterStart();
  DListElement<int>* temp = list.iterNext();
  list.deleteElem(temp,x);

  list.iterEnd();
  temp = list.iterPrev();
  list.deleteElem(temp,x);

  list.print();                // 2 3
  list.print_reverse();        // 3 2
  cout << list.length();       // 2

  system("pause");
  return 0;
}
```

Реализация с итератор

https://github.com/noraAngelova/sdp-kn-2020-2021/blob/main/linked_list/double_linked_list.cpp

Следва продължение...