Линейни структури от данни. Двойно свързан списък.

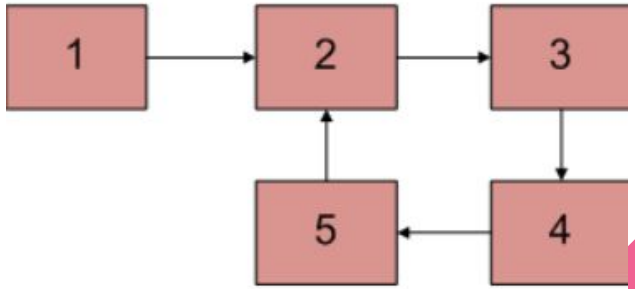
Структури от данни - семинар 2022/2023

Какво бяха индуктивните структури от данни

```
struct Person {
    char name[100];
    int age;
    Person* parent;
};
```

Вложени еднакви обекти един в друг, което създава проблем при референциите, затова се използват указатели, които водят до nullptr

Изключение: Цикличните списъци
Пример:



Възможност за разширение на ИСД:

Можем освен указател към родителя на Person, да добавим такъв и към детето му (към момента считаме че всеки Person има само 1 дете)

```
struct Person {
    char name[100];
    int age;
    Person* parent;
    Person* child;
};
```

Създаване на йерархия

C++ 11 constructors

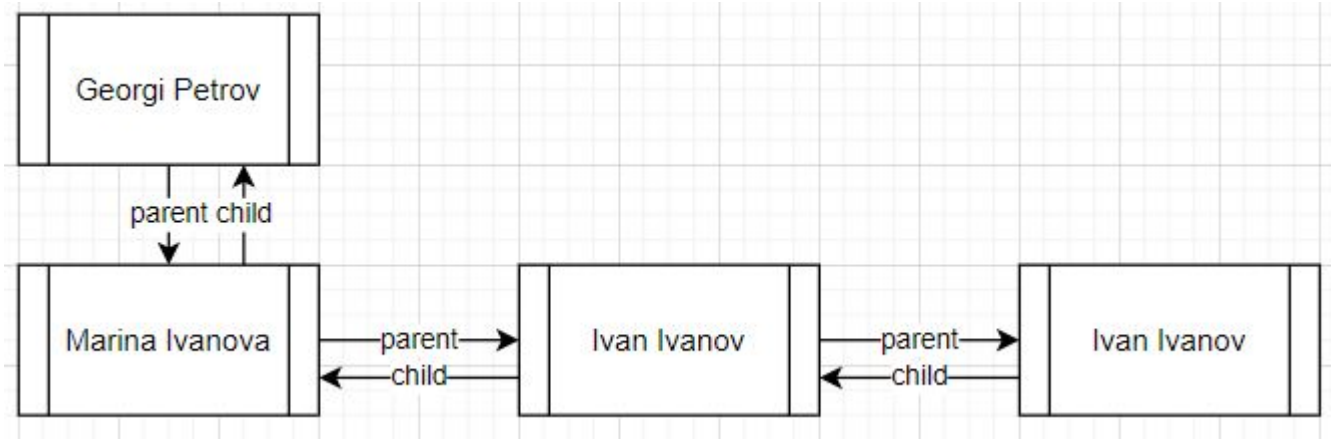
```
Person* firstPerson = new Person{ .name: "Ivan Ivanov",  .age: 25,  .parent: nullptr,  .child: nullptr};
Person* firstPersonsParent = new Person{ .name: "Petar Ivanov",  .age: 48,  .parent: nullptr,  .child: firstPerson};
Person* firstPersonsChild = new Person{ .name: "Marina Ivanova",  .age: 1,  .parent: firstPerson,  .child: nullptr};
firstPerson->parent = firstPersonsParent;
firstPerson->child = firstPersonsChild;
```

Визуална репрезентация

Добавяне на нов елемент в началото

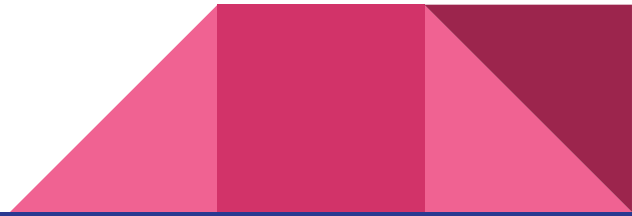
Person* marinaChild= new Person("Georgi Petrov", 0, firstPersonsChild, nullptr);

firstPersonChild->child= marinaChild;



Welcome to the nice place where it’s not needed to use save pointer

But we will use it :)



Линеен двусвързан списък

Box/Node

- Имплементация с шаблон (template)

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

- Създаване на 1 елемент

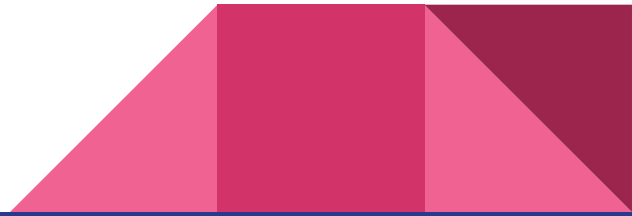
```
Node<int>* first = new Node<int>{ .data: 1,  .previous: nullptr,  .next: nullptr};
```

Основни операции

- Добавяне на елемент в началото
- Добавяне на елемент на произволно място

- Обхождане

- Изтриване на първия елемент
- Изтриване на произволен елемент



Обхождане

За разлика от едносвързаната версия използването на first = first->next не е проблем, тъй като можем да се връщаме към предишните елементи с first = first->prev

- **Обхождане от първия към последния елемент**

```
while(first != nullptr && first->next != nullptr) {
        first = first->next;
}
```

- **Обхождане от последния към първия елемент**

```
while(first != nullptr && first->prev != nullptr) {
        first = first->prev;
}
```

Добавяне на елемент в началото

first - linked list

- **With initialization**

  Node* newElement = new Node(data, nullptr, first);

  first->prev = newElement;

  first = newElement;

Добавяне на елемент на произволно място

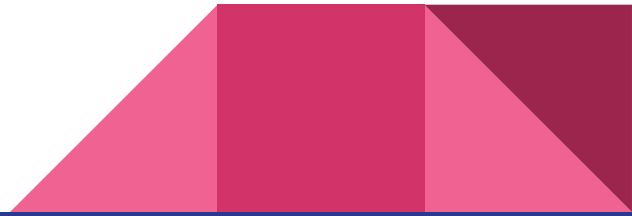
Създаваме новия елемент без prev и next, обхождаме с помощен указател и пренареждаме указателите

```
Node<T> *_new = new Node<T>(new_data,nullptr,nullptr);
Node<T> * cur = this->first;

while(cur!= nullptr && i>0)
{
    i--;
    cur=cur->next;
}
if(cur==nullptr && i)
{
    throw std::out_of_range("Index out of bounds");
}

_new->prev = cur;
cur->next->prev = _new;
_new->next = cur->next;
cur->next = _new;
```

Изтриване на първия елемент

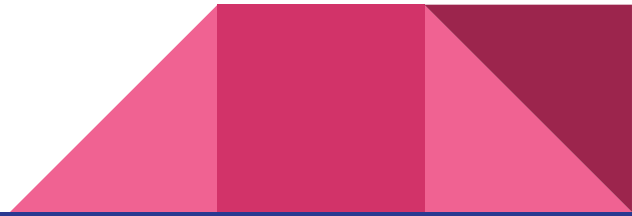
Местим указателя напред към втория елемент, изтриваме първия елемент и пренасочваме

```
first = first->next;

delete first->prev;

first->prev = nullptr;
```



Изтриване на произволен елемент

Пазим с помощен указател елемента, който искаме да изтрием пренареждаме указателите на предишния и следващия елемент и изтриваме



Шаблон за имплементация

```
template <class T>
class DLL
{
    private:
    //data members/fields

    Node<T> *first,*last;

    void copy(const DLL<T>&);
    void destroy();
```

```
public:
//constructors
DLL();
DLL(const DLL<T>&);
DLL<T>& operator=(const DLL<T>&);
~DLL();

void print();
//add/remove element
DLL<T>& push(const T&);
DLL<T>& push_back(const T&);
DLL<T>& pop();
DLL<T>& pop_back();
DLL<T>& clear();
DLL<T>& insertAfter(size_t,const T&);
DLL<T>& deleteAt(size_t);
//access
size_t size() const;
T& front() const;
T& back() const;
bool empty();
```

Анализ на сложността по време

Operation	Singly Linked List	Doubly Linked List
insert at head	O(1)	O(1)
insert at tail	O(1)	O(1)
remove at head	O(1)	O(1)
remove at tail	O(n)	O(1)