

Абстрактный тип данных СПИСОК

Операции над абстрактным списком:

- Создать пустой список
- Уничтожить список
- Определить, пуст ли список
- Определить количество элементов в списке
- Вставить элемент в указанную позицию
- Удалить элемент из указанной позиции
- Посмотреть (извлечь) элемент из заданной позиции

Диаграмма абстрактного списка



Операции над абстрактным Списком

- ***createList()*** - создает пустой список
- ***destroy()*** – уничтожает список
- ***isEmpty()*** – определяет пуст ли список
- ***insert(index, NewElement)*** - вставляет новый элемент *NewElement* в список на позицию *index*
- ***remove(index)*** – удаляет элемент списка, находящийся в позиции *index*

Операции над абстрактным Списком

- ***retrieve(index)*** – возвращает элемент, находящийся в списке на позиции *index*
- ***getlength()*** – возвращает количество элементов в списке
- ***Pos find(Element)***– возвращает позицию элемента *Element*
(*Pos* может быть как номером элемента, так и указателем на некоторый элемент)

Реализация списков

- Необходимо определить тип элементов и понятия «позиция» элемента:
- ***typedef int TypeItem*** – тип элемента может быть как простым, так и сложным
- ***typedef int Pos*** – в данном случае позицией элемента будет его номер в списке

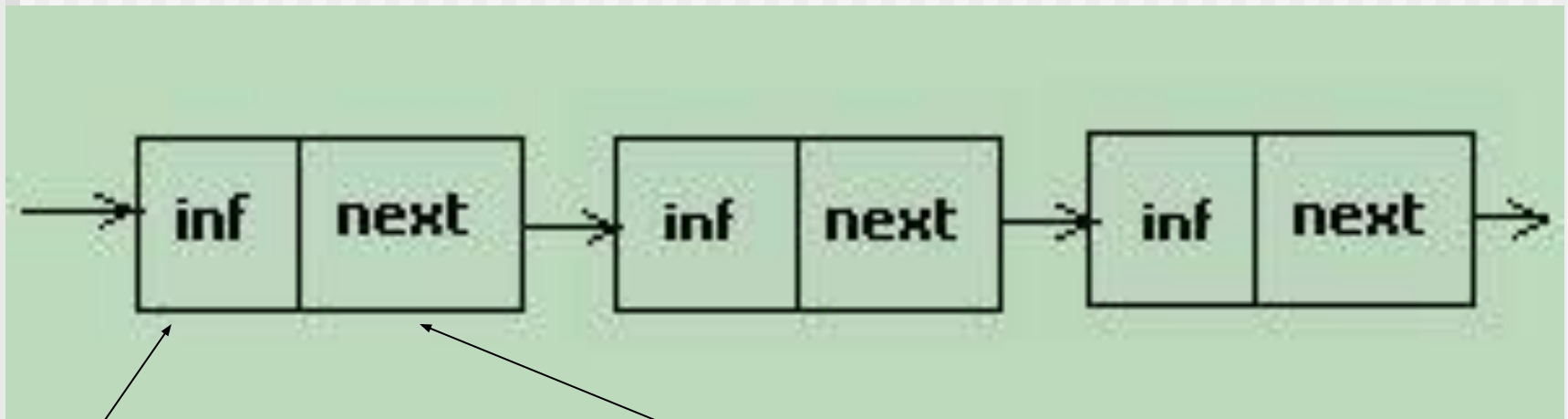
Реализация списков посредством массивов

- При реализации с помощью массивов все элементы списка располагаются в смежных ячейках, причем у каждого элемента определен номер.
- Это позволяет легко просматривать список, вставлять и удалять элементы в начало и в конец списка.
- Однако, вставка элемента в середину списка потребует от нас сдвинуть все остальные элементы, также как удаление

Реализация списков с помощью указателей

- В данном случае элементы списка не обязательно расположены в смежных ячейках, для связывания элементов используются указатели.
- Эта реализация освобождает нас с одной стороны от использования непрерывной области памяти
- Нет необходимости перемещения элементов при вставке или удалении элемента в список.
- Необходима дополнительная память для хранения указателей.

Реализация связанных списков с помощью указателей



информационная
часть

ссылочная часть –
указатель на
следующий элемент

Определение структуры List:

```
struct Node
{
    TypeItem Item; // элемент списка
    Node *Next; // указатель на
    следующий элемент
}
```

Определение структуры List:

```
struct List
```

```
{ int size ; //кол-во элементов списка  
  ListNode *head; //указатель на связный список  
  ListNode *find(int index)  
  const; //возвращает указатель на узел с номером index  
  void createList();  
  void destroyList();
```

Определение структуры List:

// Операции над списком:

```
int isEmpty() const;
int getLength() const;
void insert(int index, Typeltem newItem);
void remove(int index);
void retrieve(int index, Typeltem& dataItem)
const;
void show() const;
}; // Конец описания списка
```

Описания необходимых типов и переменных

- `typedef int Pos;` // позицией элемента будет его номер в списке
- `typedef Node *Pos;` // позицией элемента будет указатель на этот элемент