Динамические типы данных. Списки

1. Назначение и виды списков.
2. Односвязные линейные списки.
3. Двусвязные линейные списки.
4. Циклические списки.
5. Мультисписки.
6. Нелинейные списки.
7. Язык программирования ЛИСП.

1. Назначение и виды списков

Списком называется упорядоченное, возможно пустое, множество (набор) элементов (**узлов**), которые состоят из ***данных*** и ***полей связи*** между узлами***.*** К спискам применимы ряд операций, например, **включения, исключения, копирования, поиска, сортировки**.

Списки – наиболее широко применяемая в программировании динамическая структура данных:

- при решении прикладных задач программирования:

• при организации списков объектов (пользователей, задач, документов, рассылки, и т.п.);

- в системном императивном программировании:

• при реализации ядра ОС;

• при реализации СУБД;

• при реализации пользовательского интерфейса;

• при работе с файлами;

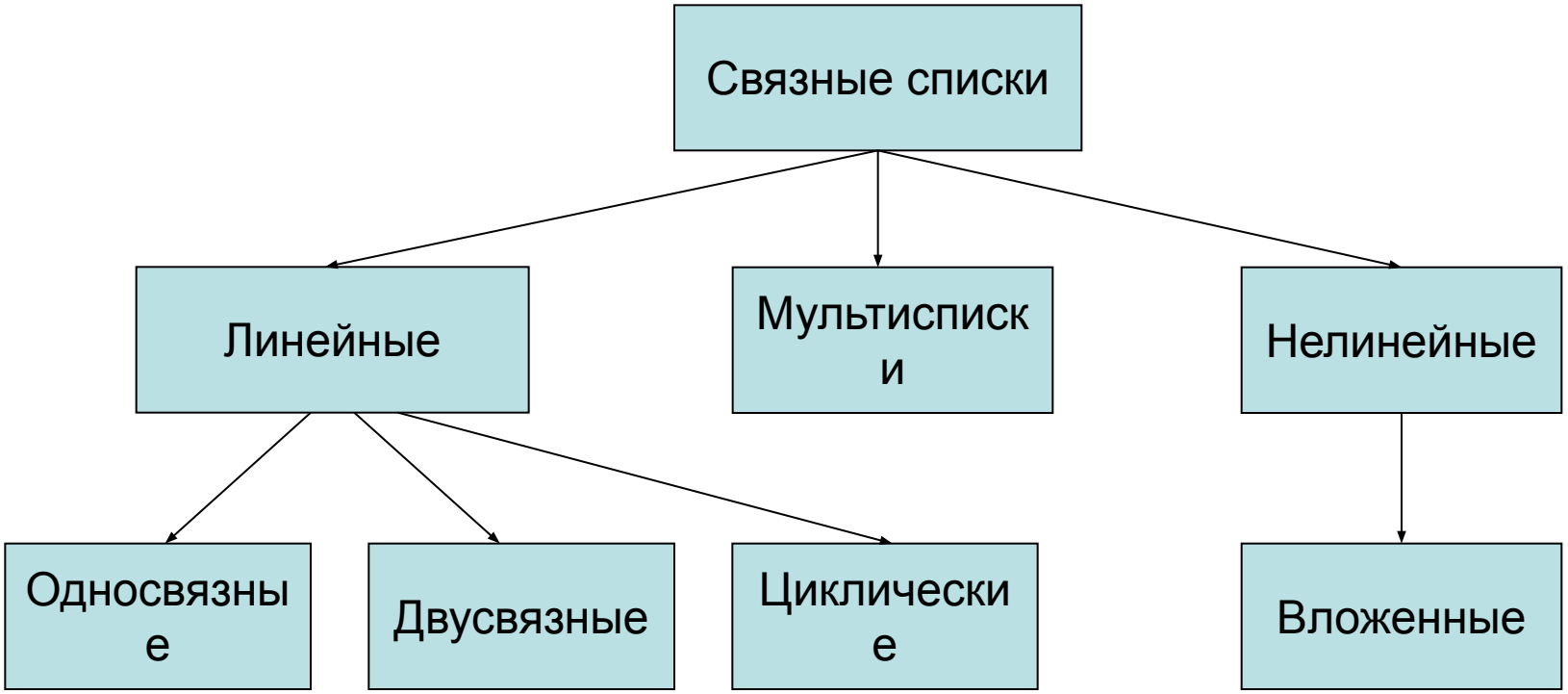
• при построении других динамических структур данных: стеки, очереди, деревья, сети и графы, хеш-таблицы;

• при построении трансляторов;

в функциональном программировании:

• язык ЛИСП и его диалекты

Виды связных списков



Линейные – все узлы расположены на одном уровне (в линию).
Мультисписки – узлы могут быть элементами нескольких списков.
Нелинейные – узлы расположены на разных уровнях.
Вложенные – узлами могут быть другие списки (подсписки).

Особенности списков

1. Последовательный доступ (вместо прямого, как у массивов).
2. Произвольное размещение в динамической памяти.
3. Используются указатели для реализации полей связи.

Достоинства:

•лёгкость добавления и удаления элементов;

•размер ограничен только объёмом памяти компьютера и разрядностью указателей;

Недостатки:

•на поля связи (указатели на следующий и предыдущий элемент) расходуется дополнительная пам’ять;

•работа со списком медленнее, чем с массивами, так как к любому элементу списка можно обратиться, только пройдя все предшествующие ему элементы;

•элементы списка могут быть расположены в памяти разреженно, что окажет негативный эффект на кэширование процессора;

2. Линейные односвязные списки

1. Односвязный список состоит из узлов (элементов), которые состоят из поля данных и указателя на следующий узел.
2. Начальный узел называется головой списка.
3. Значение указателя связи последнего узла равно **NULL**.
4. Полей данных может быть несколько.

Сравнение массивов и списков

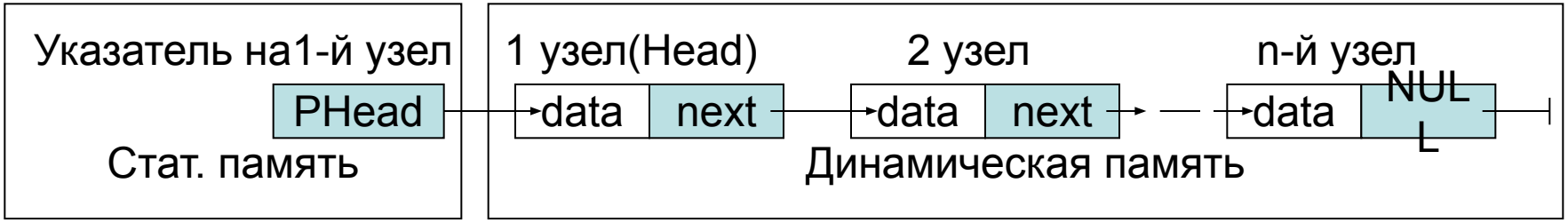
Последовательное распределение

Адрес	Содержимое
$L_0 + c$:	Элемент 1
$L_0 + 2c$:	Элемент 2
$L_0 + 3c$:	Элемент 3
$L_0 + 4c$:	Элемент 4
$L_0 + 5c$:	Элемент 5

Связанное распределение

Адрес	Содержимое	
A:	Элемент 1	B
B:	Элемент 2	C
C:	Элемент 3	D
D:	Элемент 4	E
E:	Элемент 5	Λ

Операции над списками



- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке
- удаление списка.

Описание узла списка

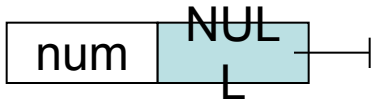
```
struct Node
{
        int  data; // поле данных
        Node *next; // указатель на след.узел
};
```

Создание списка

```
typedef Node *PNode; // тип данных: указатель на узел
PNode PHead; // указатель на голову списка
PHead = NULL; // список пуст
```

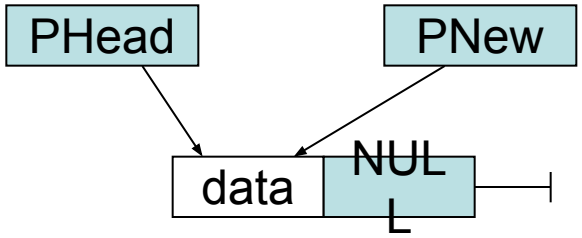
Создание узла

```
PNode createNode (int num)
{
        PNode PNew = new Node; // указатель на новый узел
        PNew->data = num; // поле данных – номер узла
        PNew->next = NULL; // следующего узла нет
        return PNew; // результат функции – адрес узла
}
```



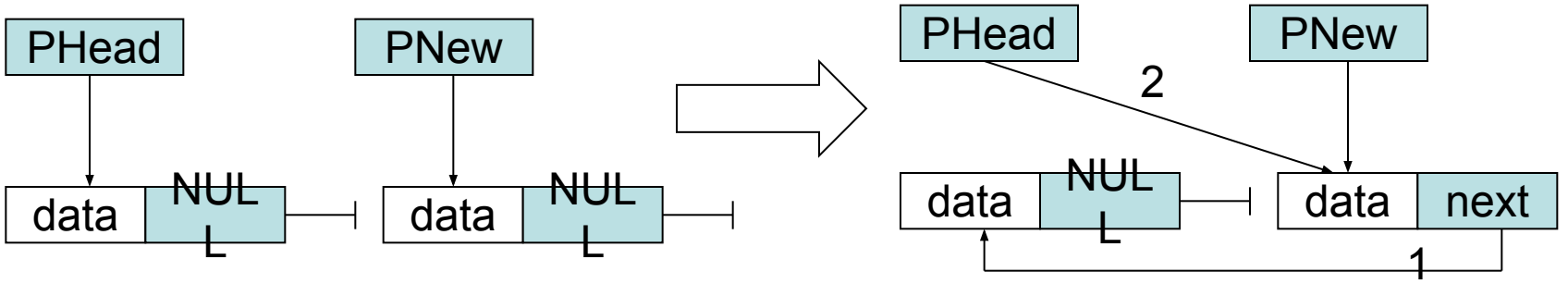
Добавление узла в список

в пустой список:




```
PNew = CreateNode (1);
PHead = PNew;
```

в начало списка:

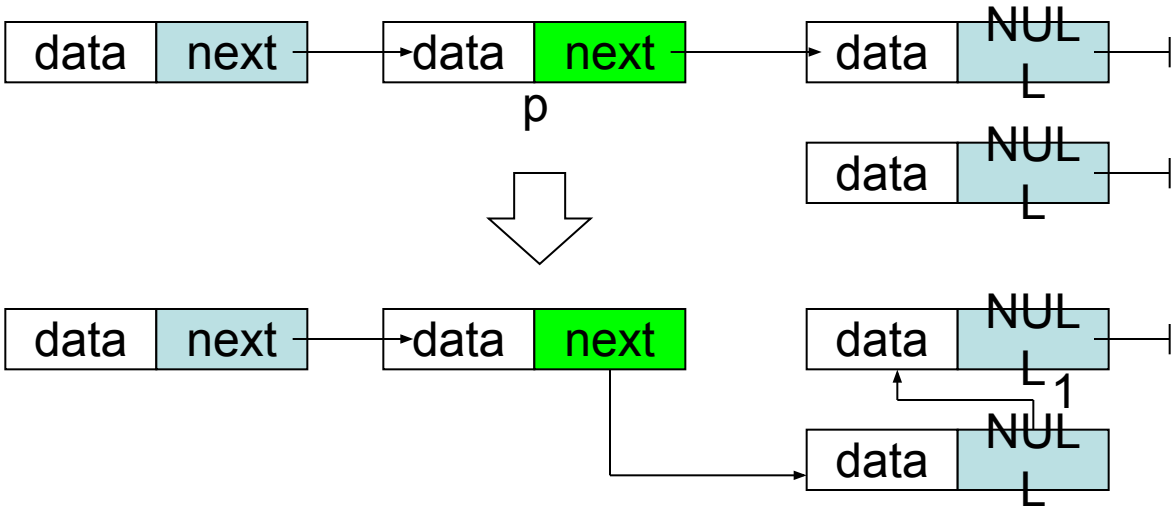



```
void addFirst (PNode &PHead, PNode PNew)
{
// установить next нового узла на голову списка
      PNew->next = PHead;
//установить голову списка на новый узел
      PHead = PNew;
}
```

адрес головы списка передается *по ссылке*

Добавление узла в список

После заданного узла:




```
void addAfter (PNode p, PNode PNew)
{
//установить next нового узла на узел, следующий за заданным
PNew->next = p->next;// p – указатель на заданный узел
//установить next заданного узла на новый узел
      p->next = PNew;
}
```

Последовательность операций менять нельзя, т.к. будет потерян адрес узла, следующего за заданным.

Добавление узла в конец списка

```
void addLast(PNode PHead, PNode PNew)
{
	PNode q = PHead;
	if (Head == NULL) { // если список пуст,
		AddFirst(Head, NewNode); // вставляем первый элемент
		return;
		}
	while (q->next) q = q->next; // ищем последний элемент
	AddAfter(q, PNew);
}
```

Добавление узла перед заданным

```
void addBefore(PNode PHead, PNode p, PNode PNew)
{
	PNode q = PHead;
	if (Head == p) {
		AddFirst(Head, NewNode); // вставка перед первым узлом
		return;
		}
	while (q && q->next!=p) // ищем узел, за которым следует p
		q = q->next;
	if ( q ) // если нашли такой узел,
	AddAfter(q, NewNode); // добавить новый после него
}
```

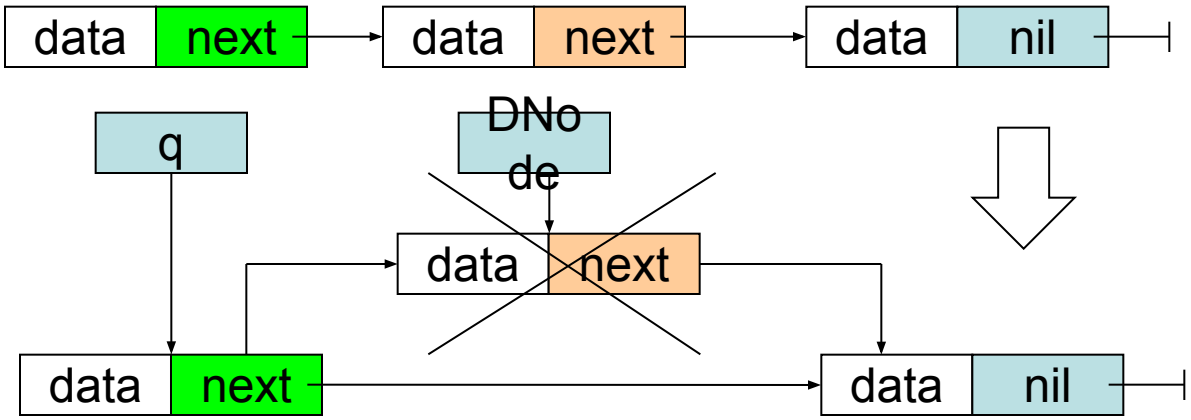
Проход по списку

```
void print_list(PNode PHead)
{
	PNode p = PHead;
	printf("[ ");
	while(p != NULL)
	{
		printf("%d ", p->data);
		p = p->next;
	}
	printf("]\n");
}
```

Поиск элемента в списке

```
PNode findNode (PNode PHead, int num)
{
	PNode q = PHead;
// пока есть узел проверить нужное условие
	while (q && ((q->data) != num))
		q = q->next;
	return q;
}
```

Удаление узла после заданного




```
void deleteNode(PNode PHead, PNode DNode)
{
	PNode q = PHead;
	if (Head == DNode)
		Head = DNode->next; // удаляем первый элемент
	else {
		while (q && q->next != DNode) // ищем элемент
			q = q->next;
	if ( q == NULL ) return; // если не нашли, выход
	q->next = DNode->next;
		}
	delete DNode; // освобождаем память
}
```

Удаление списка

```
void delete_list(PNode &PHead){
        if (PHead != NULL){
          delete_List(PHead->next);
          delete PHead;
                    }
}
```

Задание:

1. Удалить все элементы.
2. Найти средину списка.
3. Найти средину списка за один проход.

Пример программы

```
#include <stdio.h>
int main()
{
Node* list = NULL;
print_list(list); // выводит: [ ]
pNode = createNode(1); //создание узла
print_list(list); // печать списка: [ 1 ]
addFirst(list, 10); //вставка в начало списка
addFirst(list, 8); //вставка в начало списка
print_list(list); // печать списка: [ 8 10 1 ]
Node* node = findNode(list, 10);// поиск элемента
Node *pNew = createNode(3); //создание узла
addAfter(node, pNew);//вставка после
print_list(list); // печать списка: [ 8 10 3 1 ]
deleteNode(list, node);// удаление узла
print_list(list); // печать списка: [ 8 3 1 ]
pNode = delete_list(list); // удаление списка
print_list(list); // печать списка: [ ]
}
```