Семинар №12

ФАКТ 2021

Бирюков В. А.

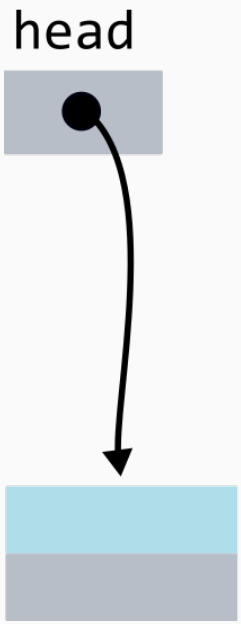
November 22, 2021

Связный список

Связный список: Структура узла

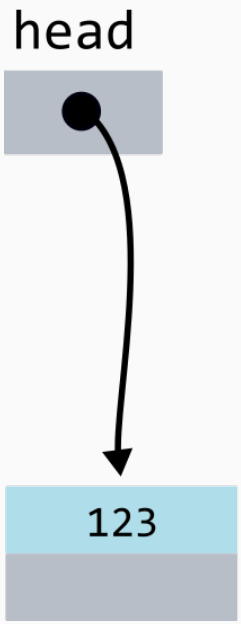

```c
struct node {
    int value;
    struct node* next;
};
typedef struct node Node;
```

Работа со связным списком



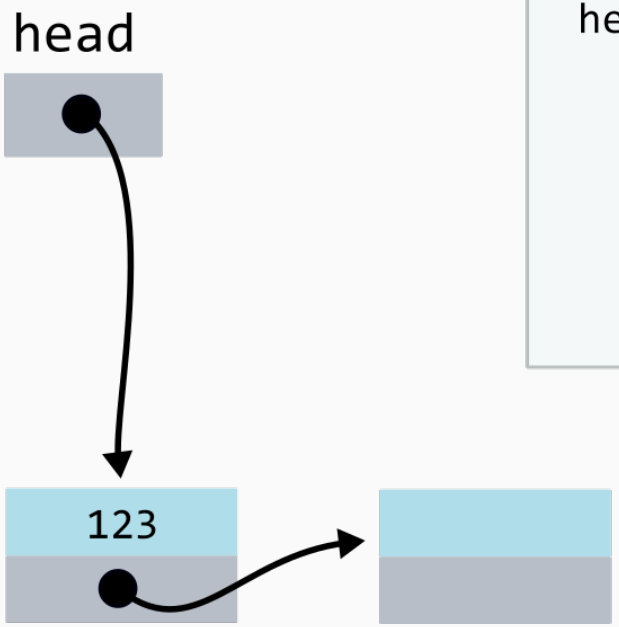

```
Node* head = malloc(sizeof(Node));
```

Работа со связным списком



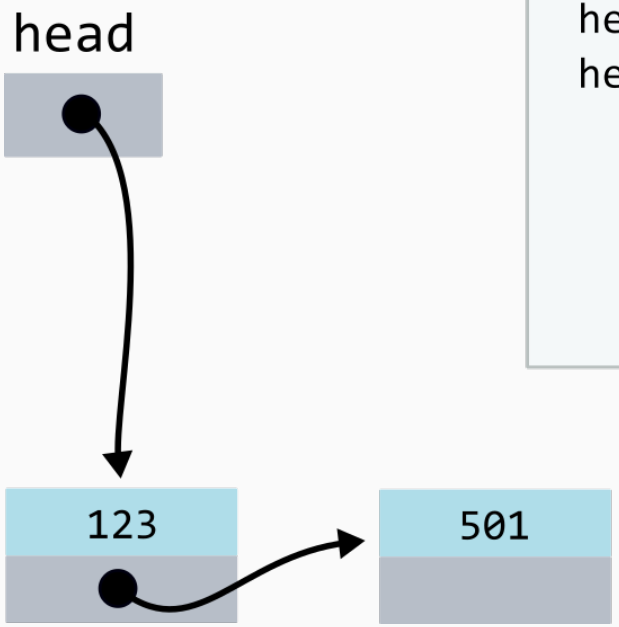

```
Node* head = malloc(sizeof(Node));
head->value = 123;
```

Работа со связным списком




```
Node* head = malloc(sizeof(Node));
head->value = 123;
head->next = malloc(sizeof(Node));
```

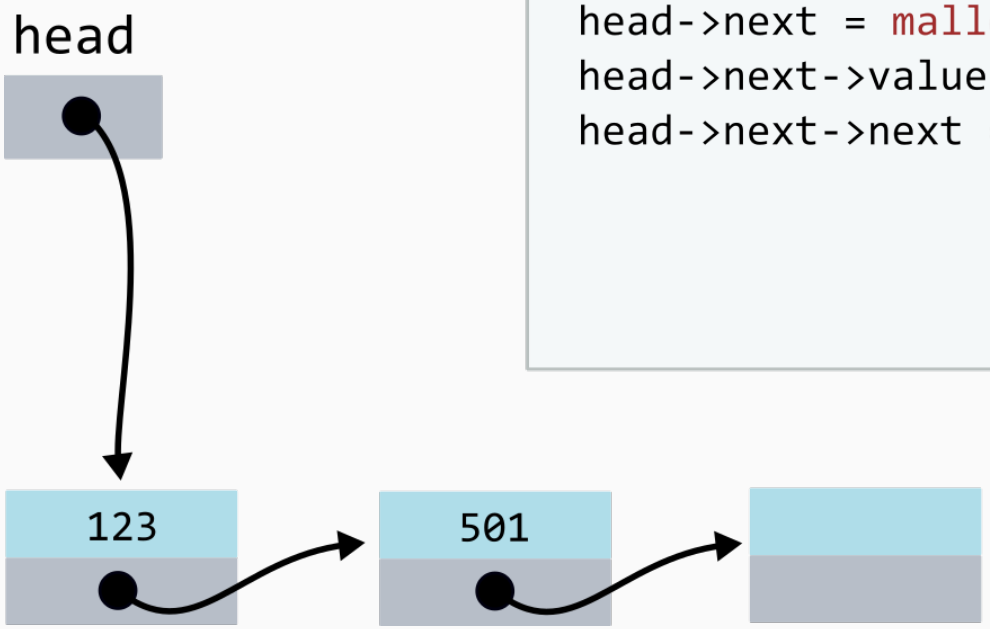
Работа со связным списком



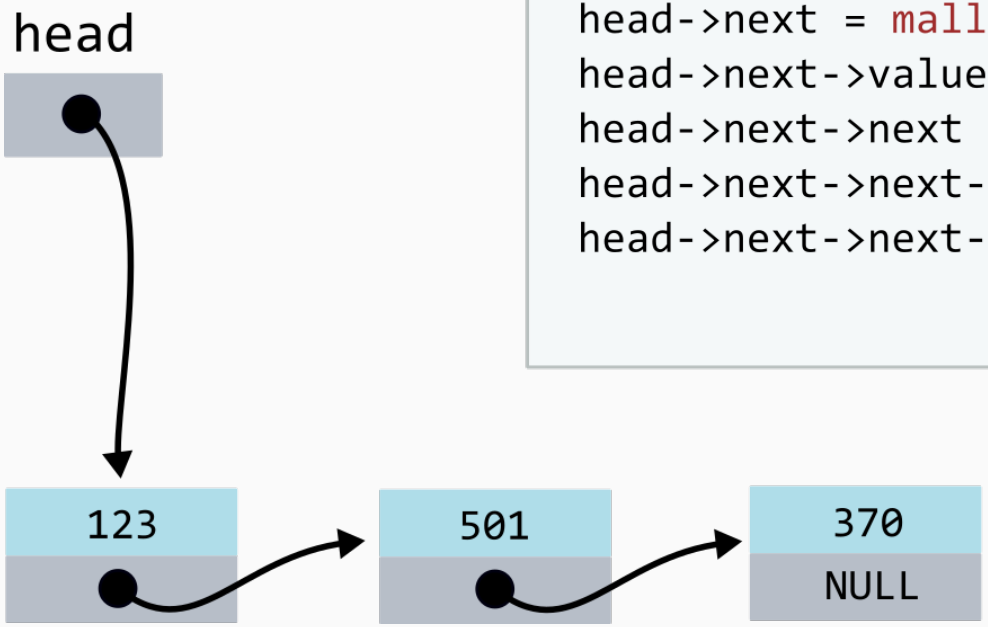

```
Node* head = malloc(sizeof(Node));
head->value = 123;
head->next = malloc(sizeof(Node));
head->next->value = 501;
```

Работа со связным списком

```
Node* head = malloc(sizeof(Node));
head->value = 123;
head->next = malloc(sizeof(Node));
head->next->value = 501;
head->next->next = malloc(sizeof(Node));
```



Работа со связным списком

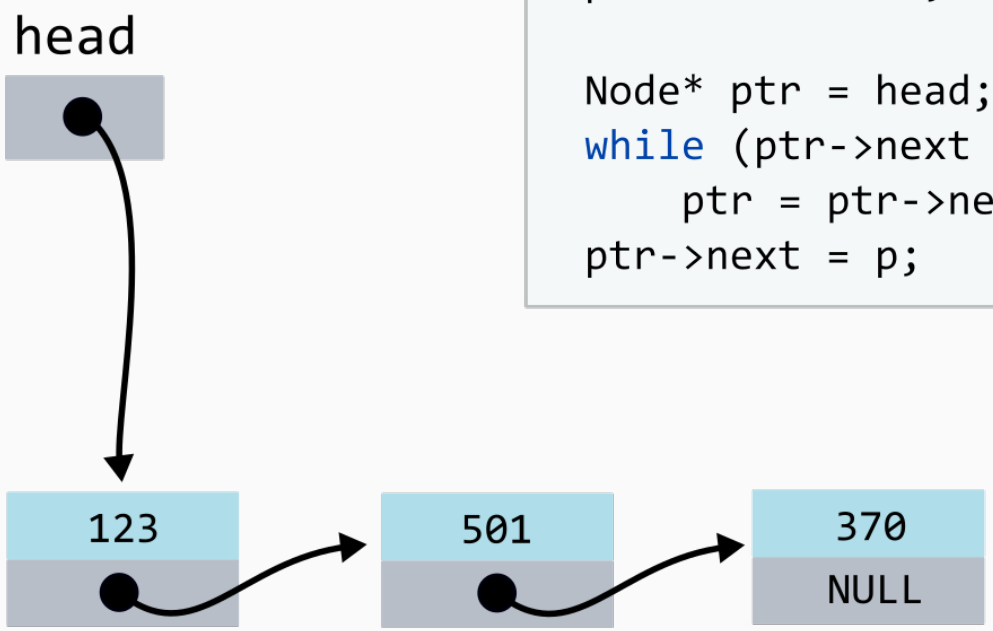



```
Node* head = malloc(sizeof(Node));
head->value = 123;
head->next = malloc(sizeof(Node));
head->next->value = 501;
head->next->next = malloc(sizeof(Node));
head->next->next->value = 370;
head->next->next->next = NULL;
```

Добавление элемента в конец связного списка

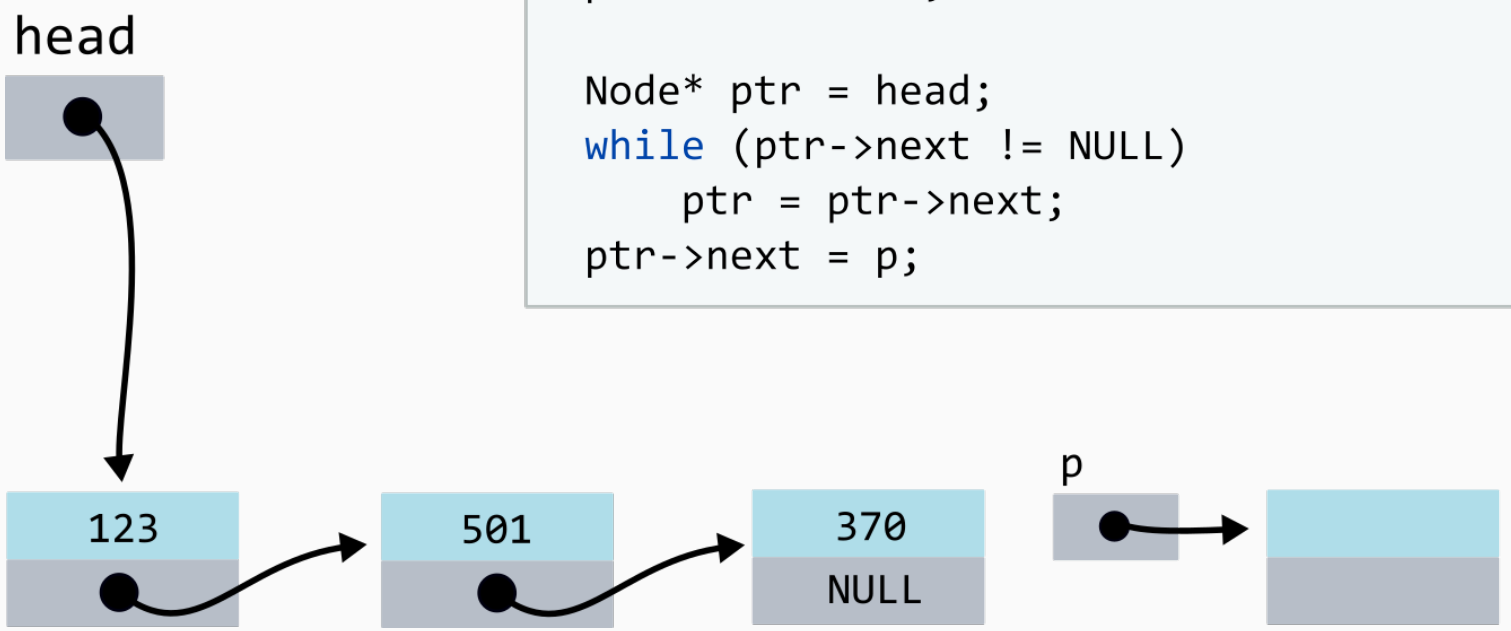
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



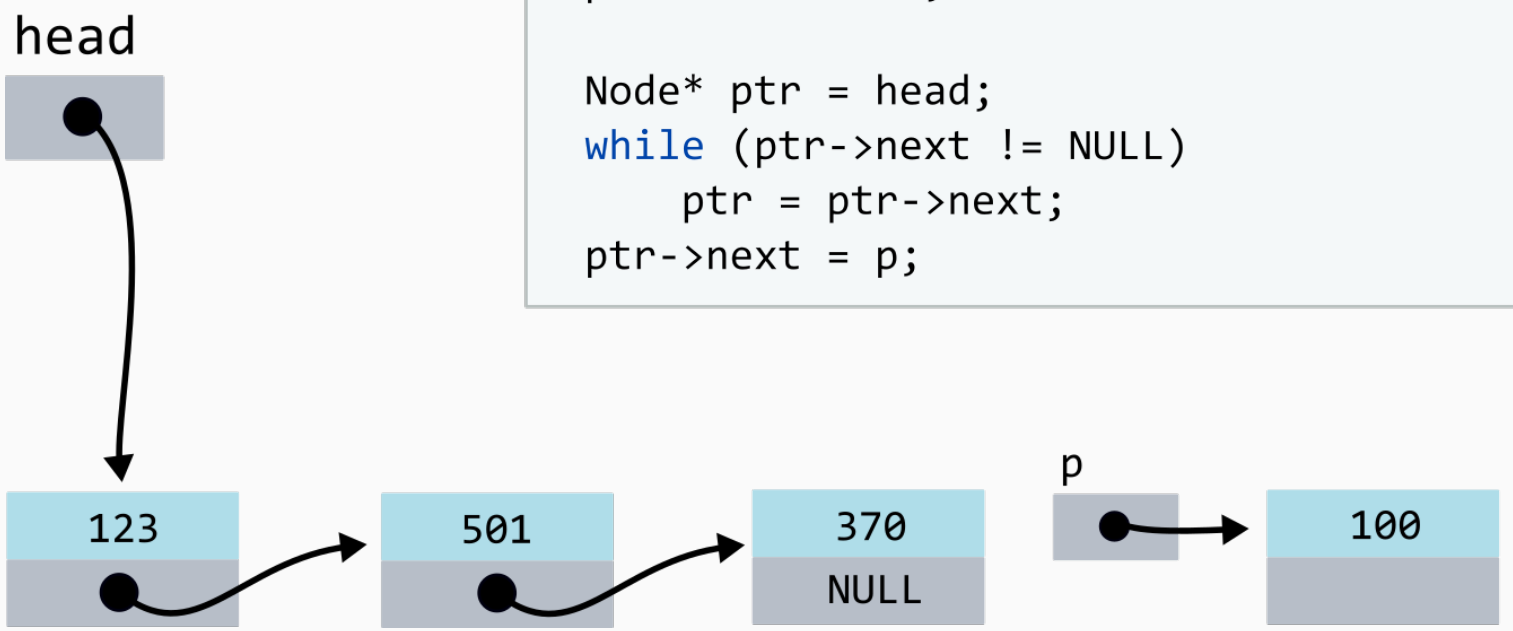
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



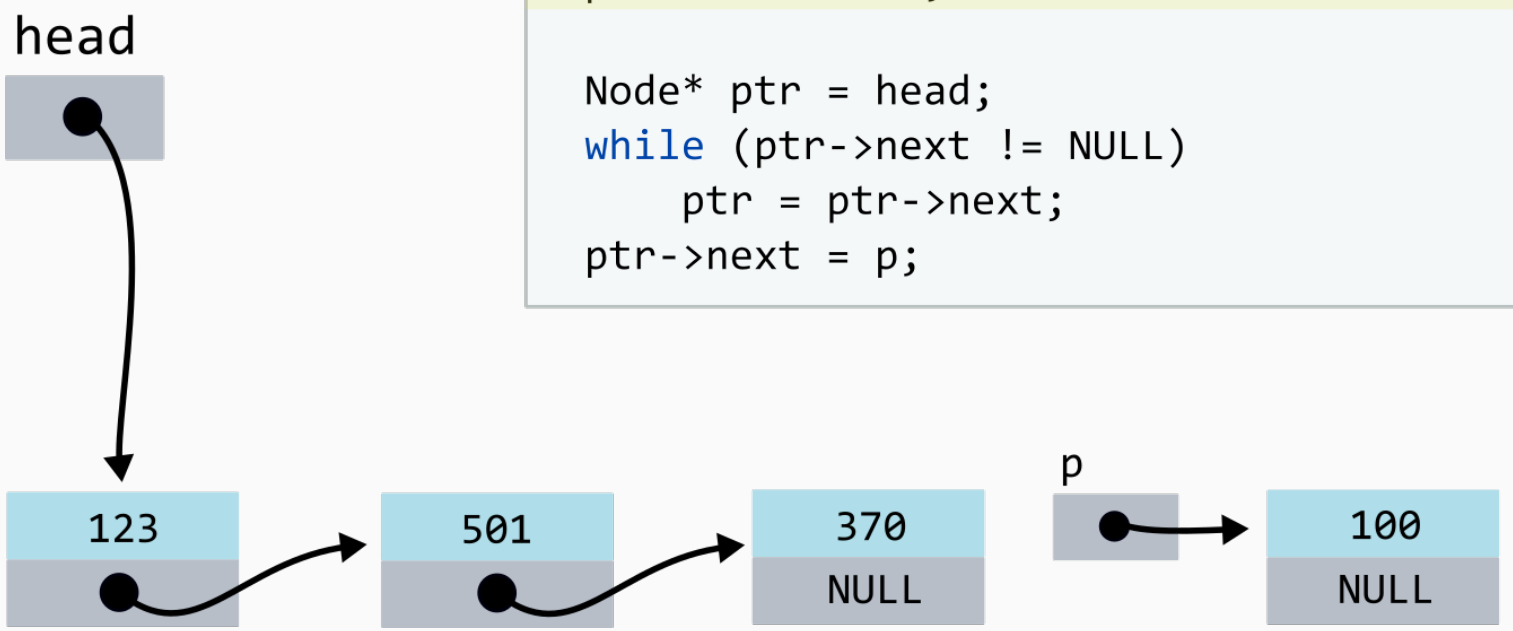
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



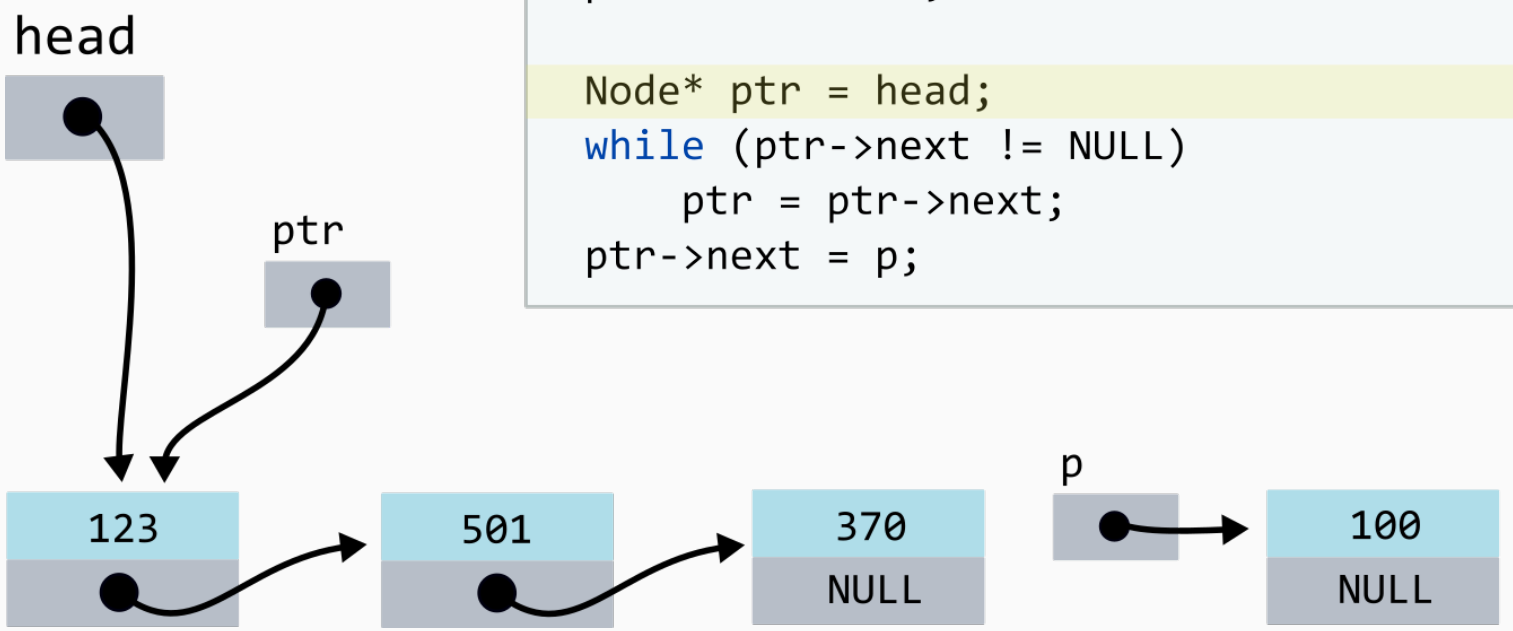
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



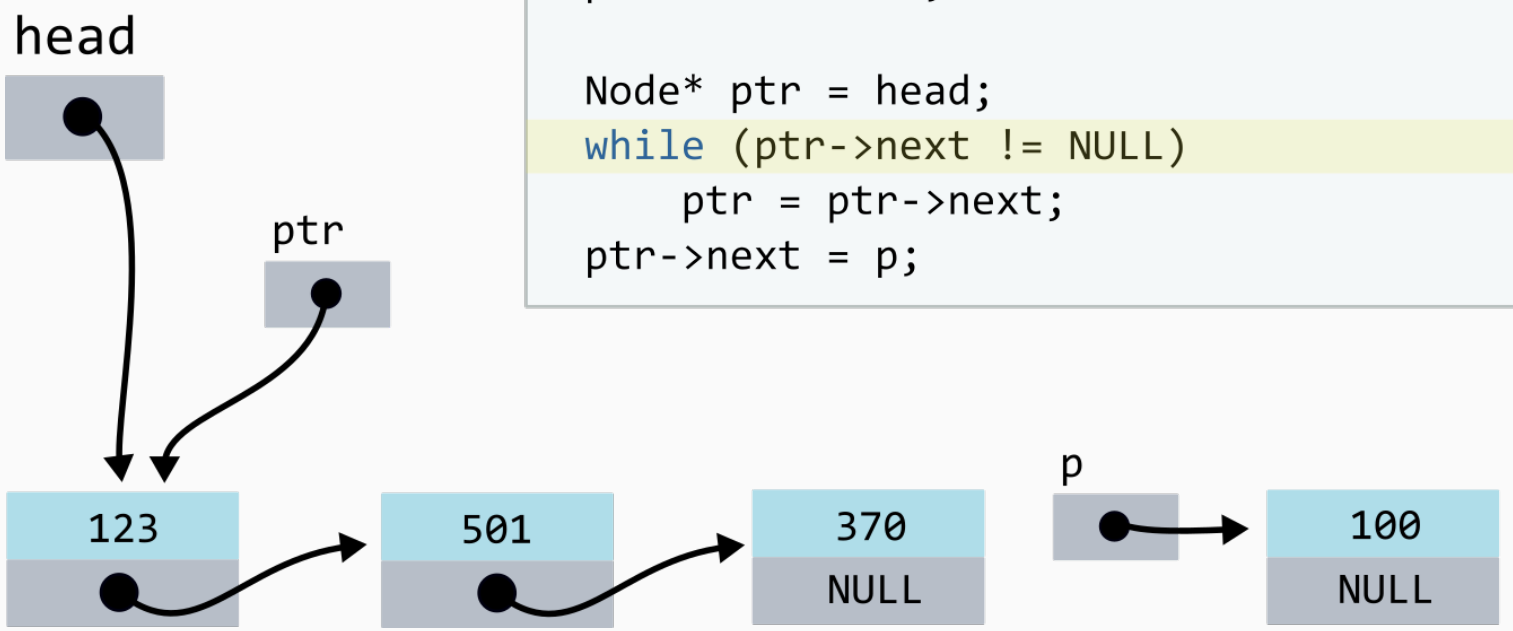
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



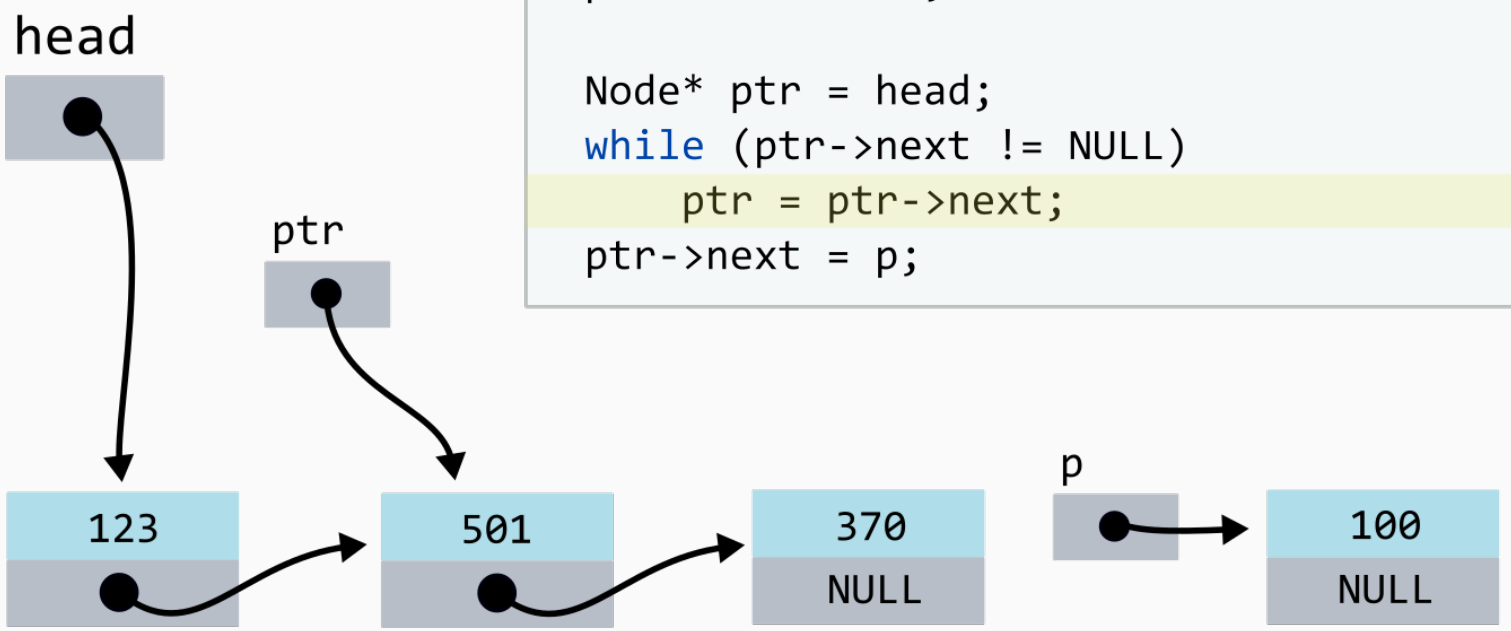
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



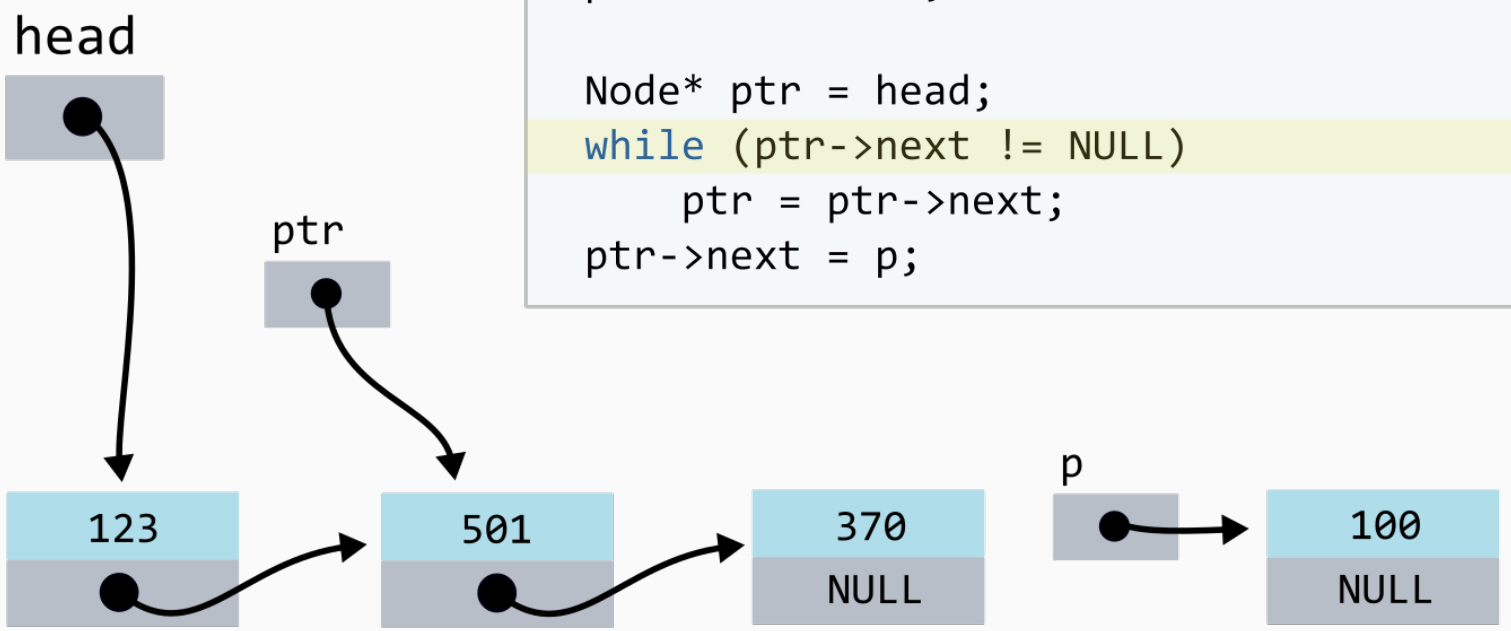
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



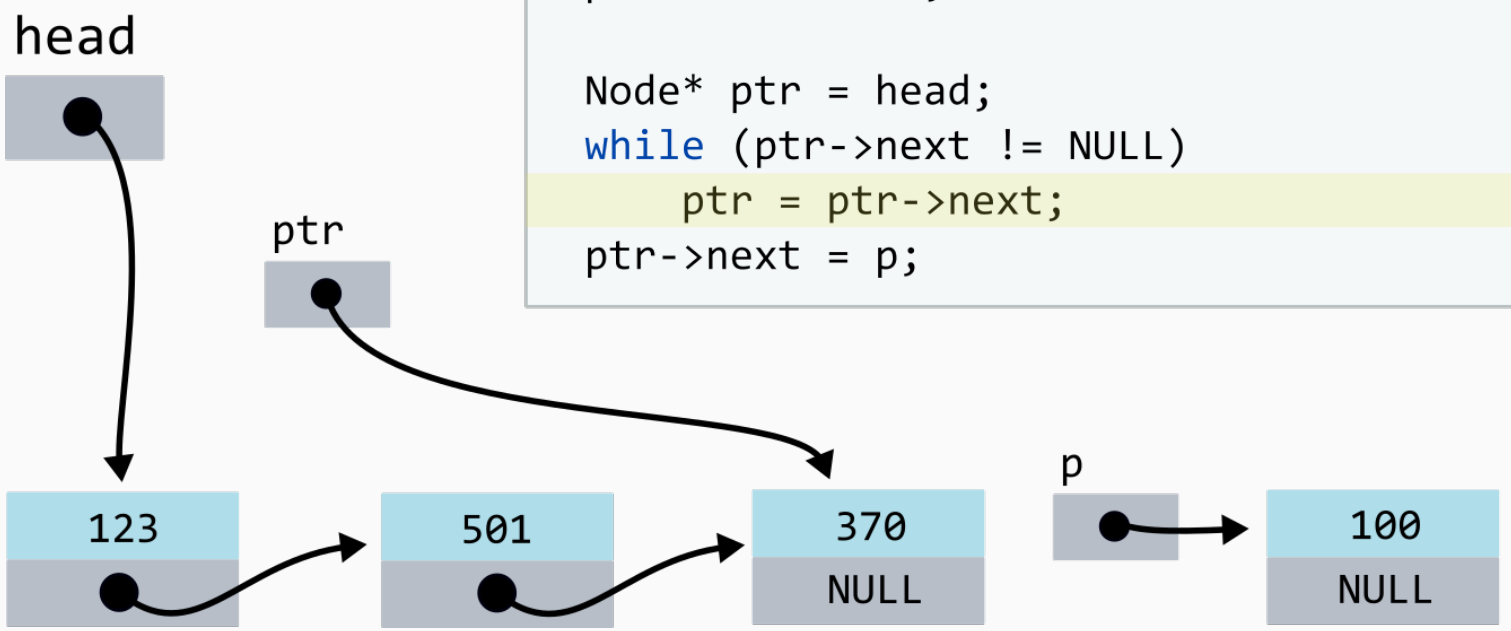
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



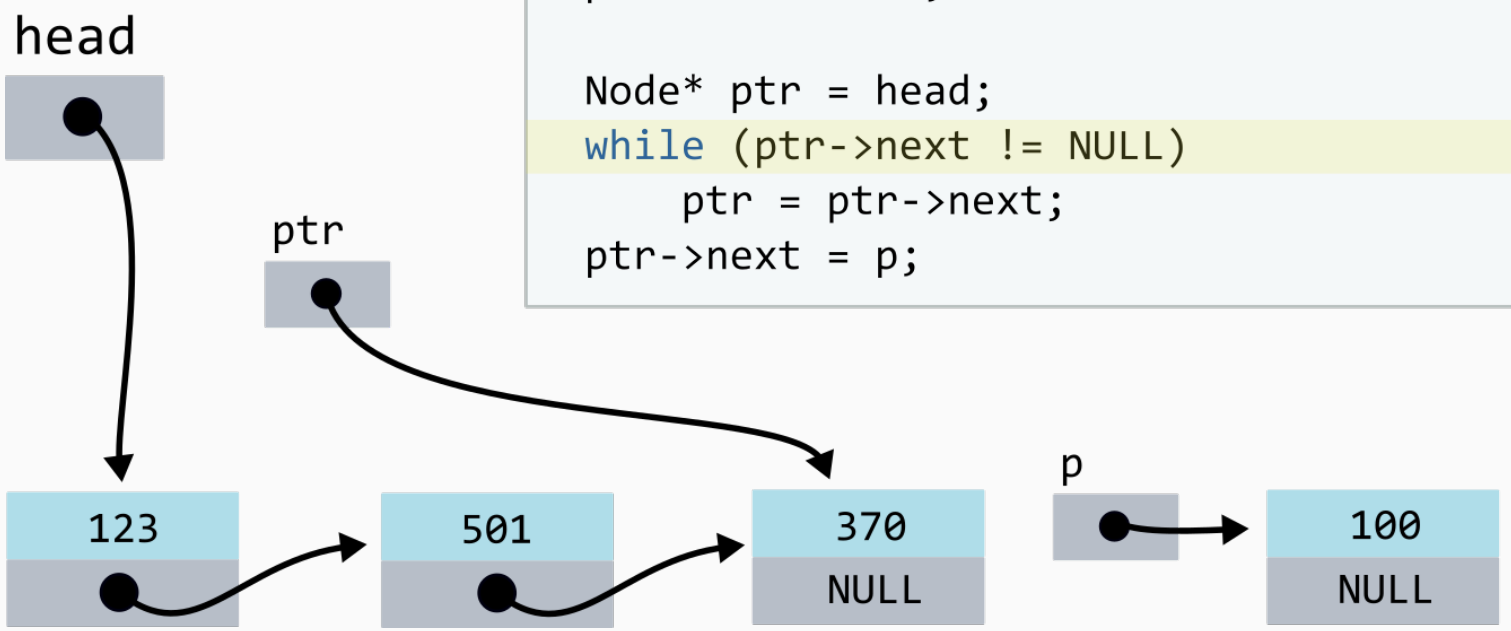
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



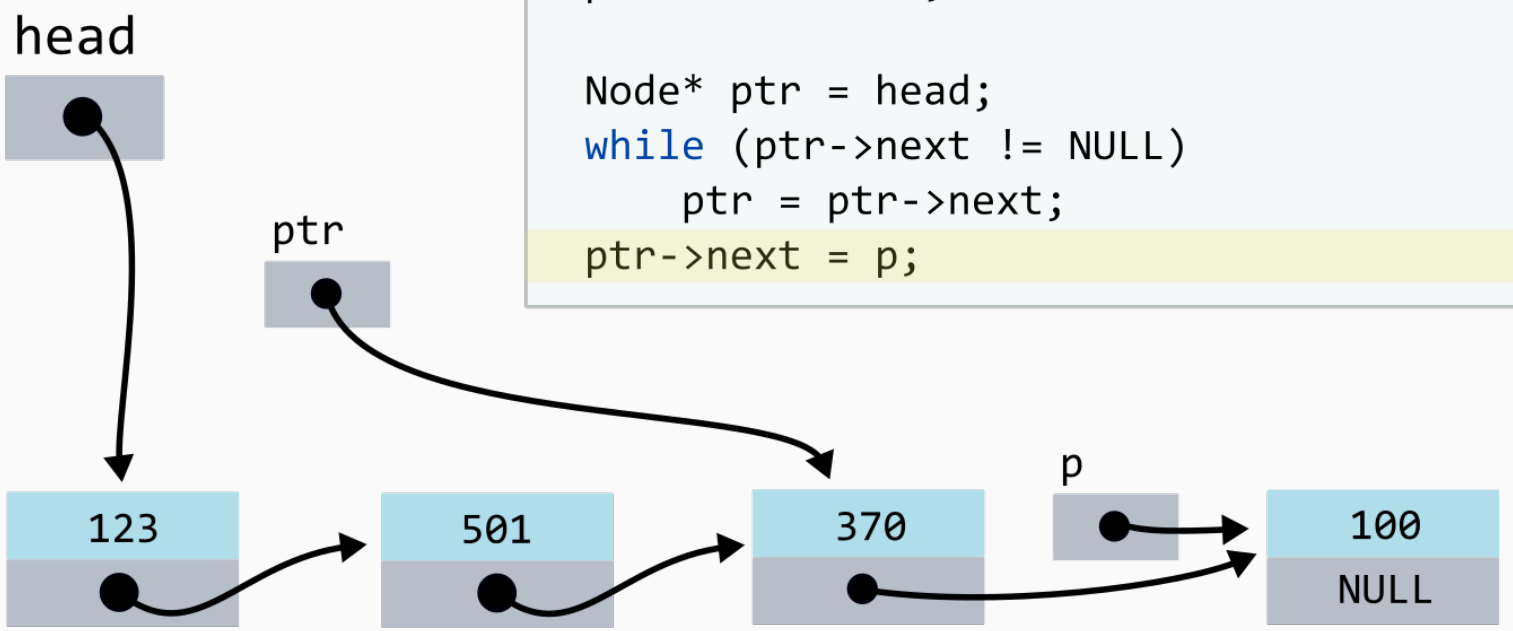
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



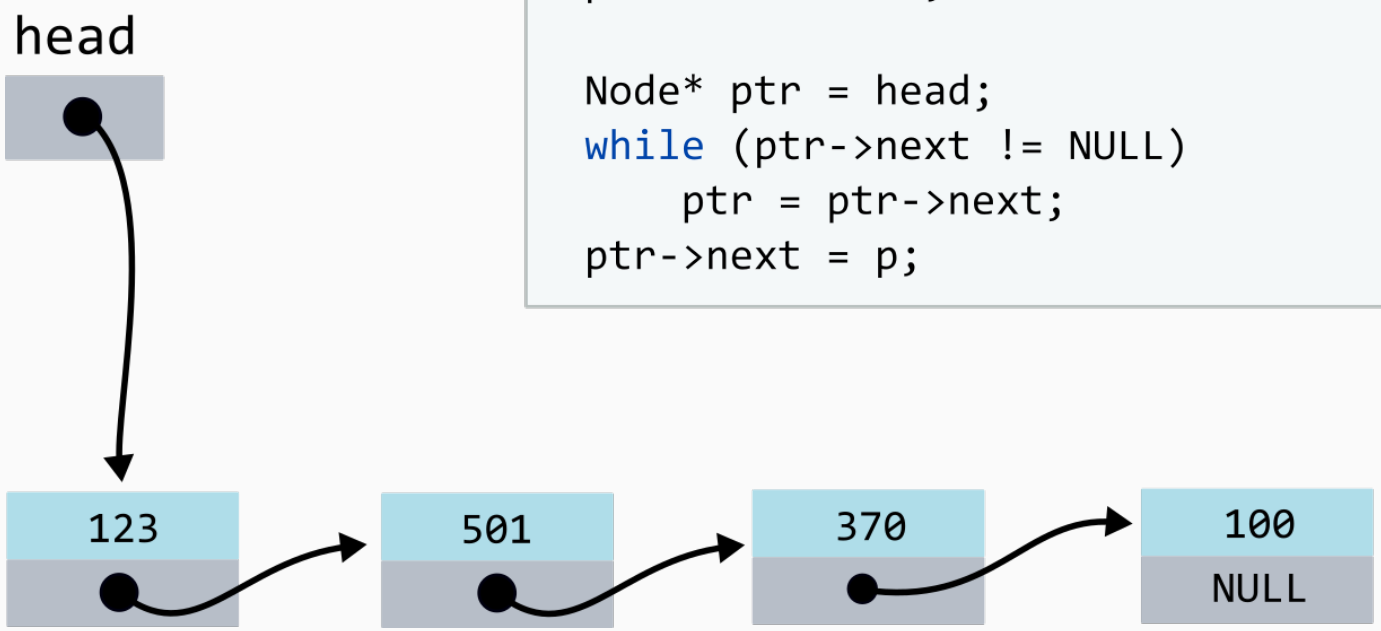
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```



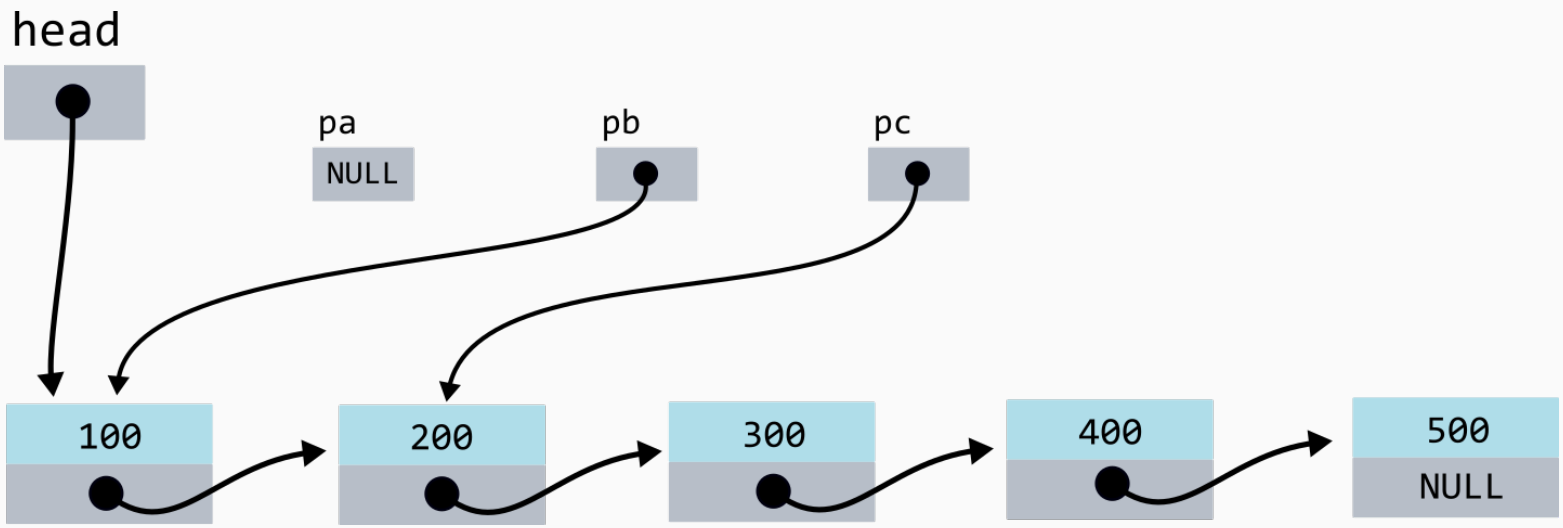
Добавление элемента в конец связного списка

```
Node* p = malloc(sizeof(Node));
p->value = 100;
p->next = NULL;

Node* ptr = head;
while (ptr->next != NULL)
    ptr = ptr->next;
ptr->next = p;
```

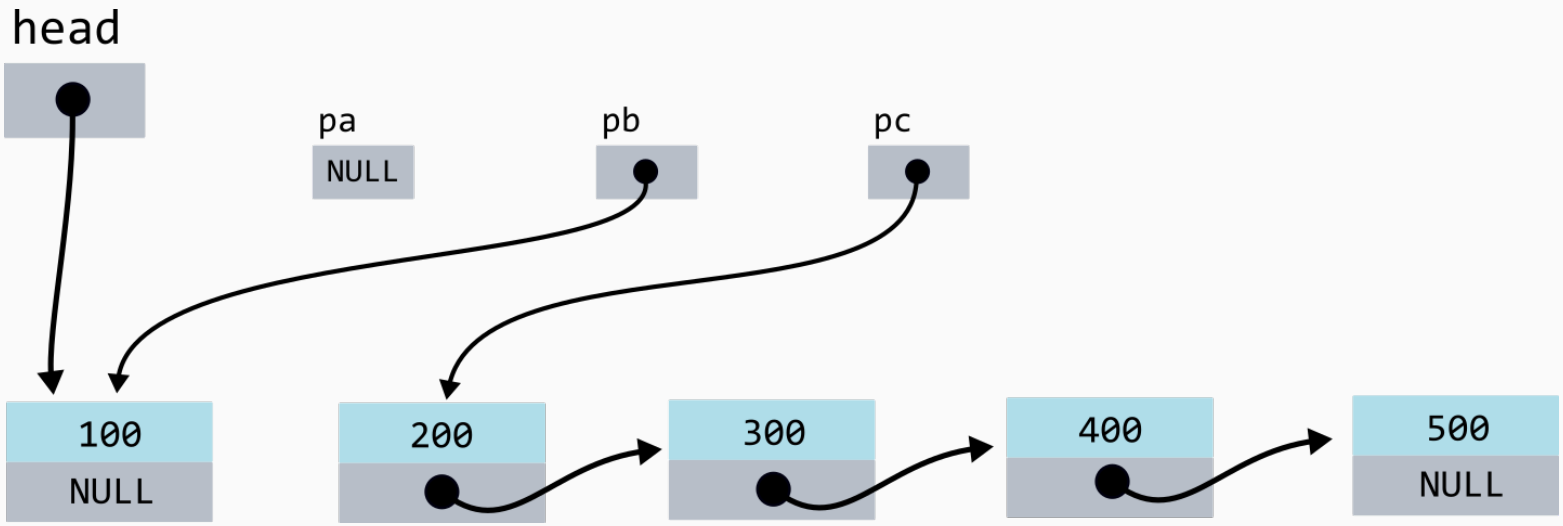


Обращение связного списка

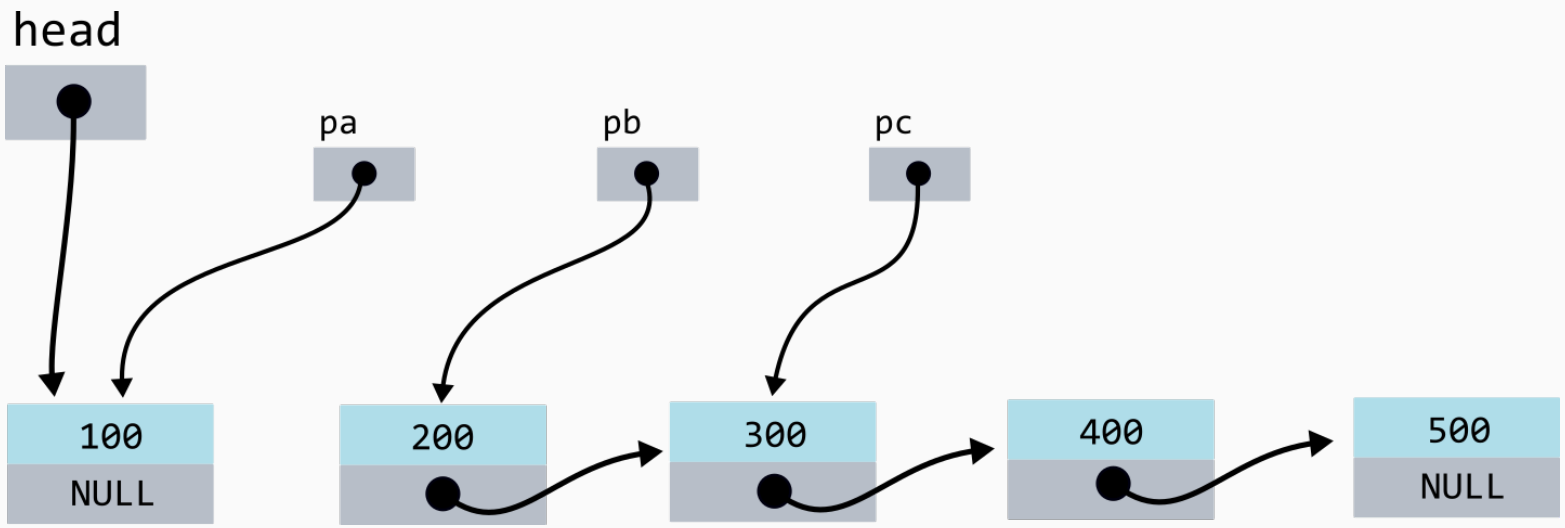
Обращение связного списка



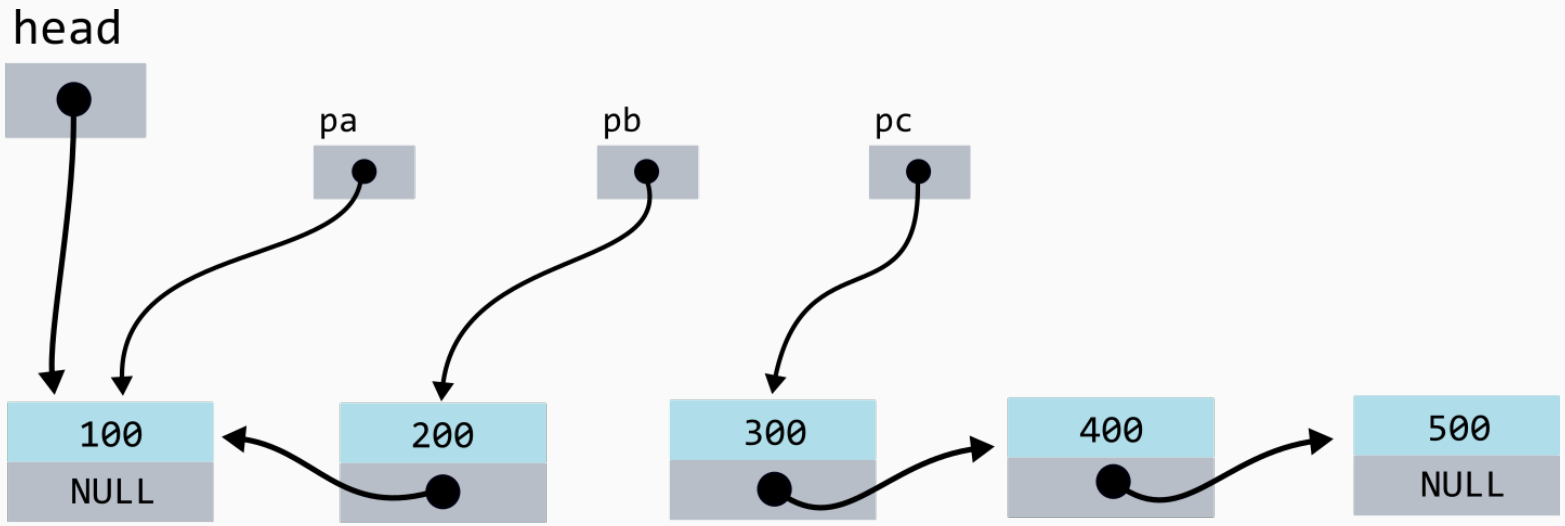
Обращение связного списка



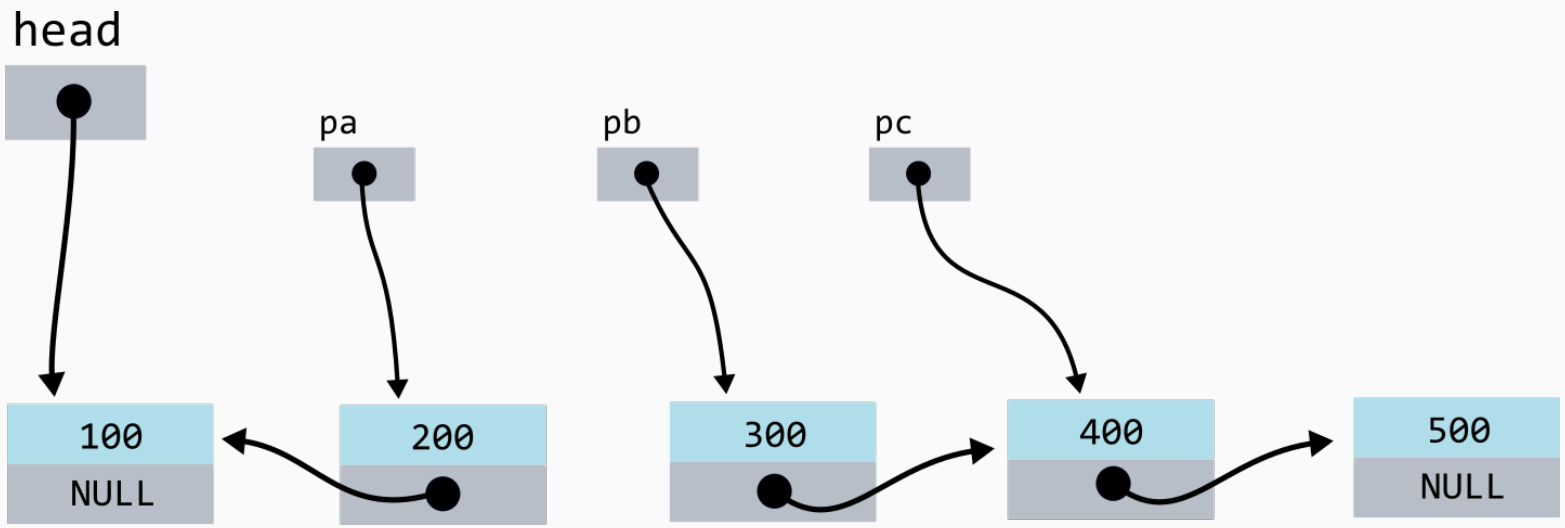
Обращение связного списка



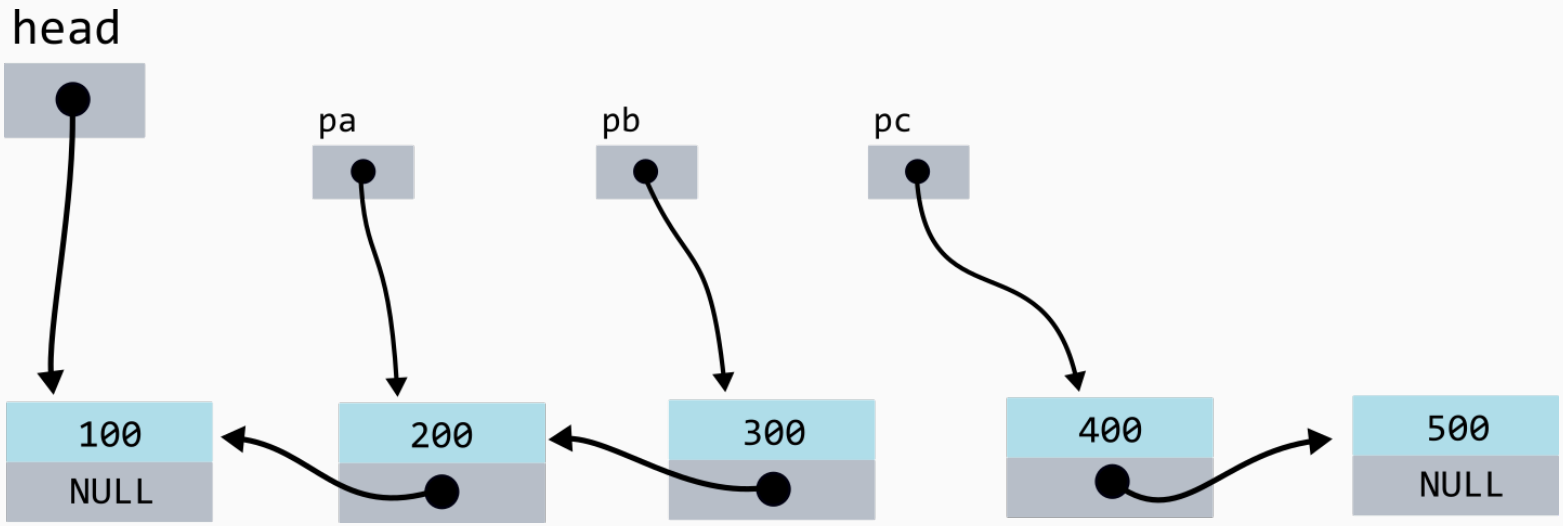
Обращение связного списка



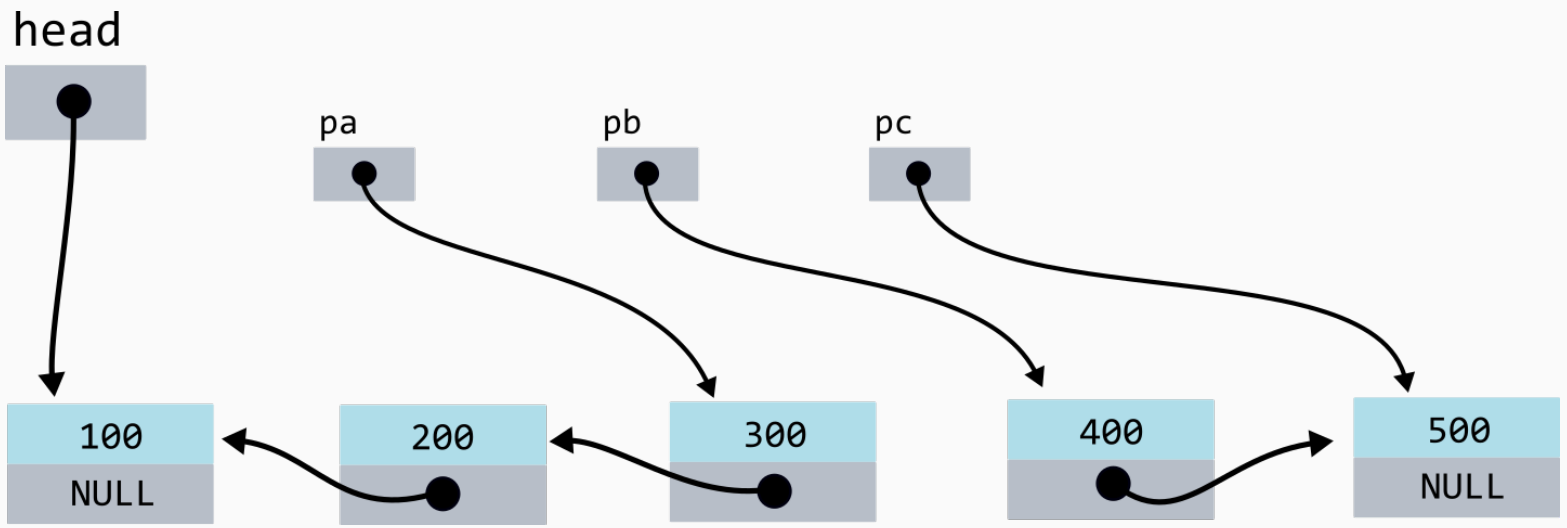
Обращение связного списка



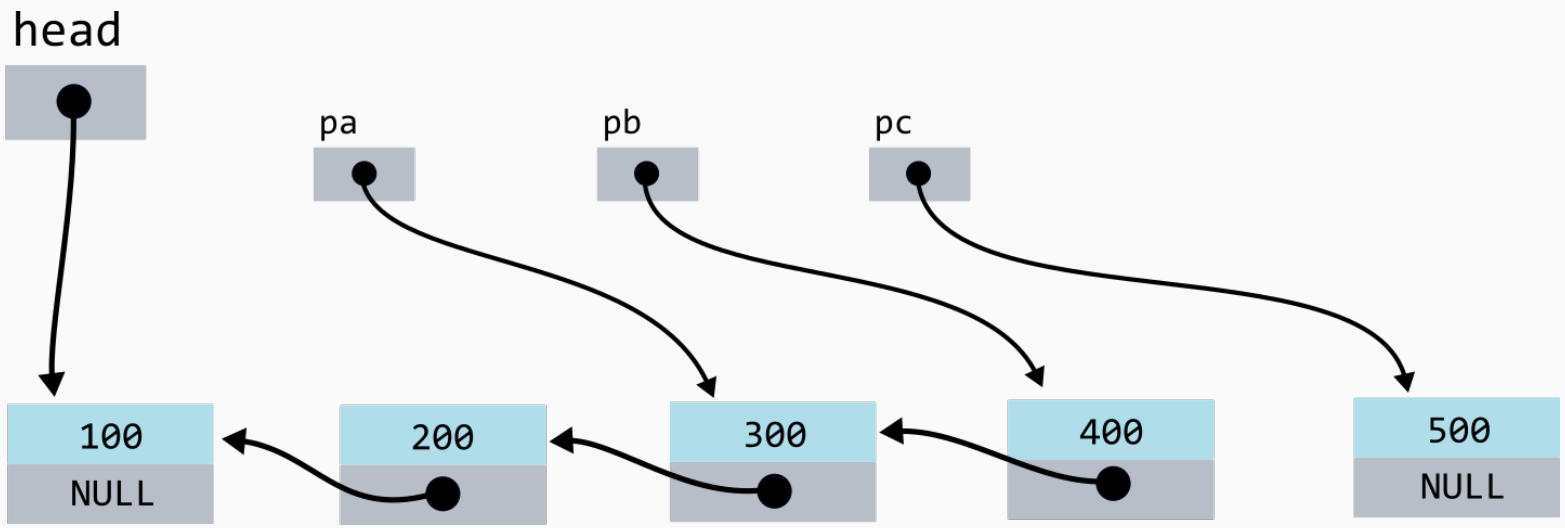
Обращение связного списка



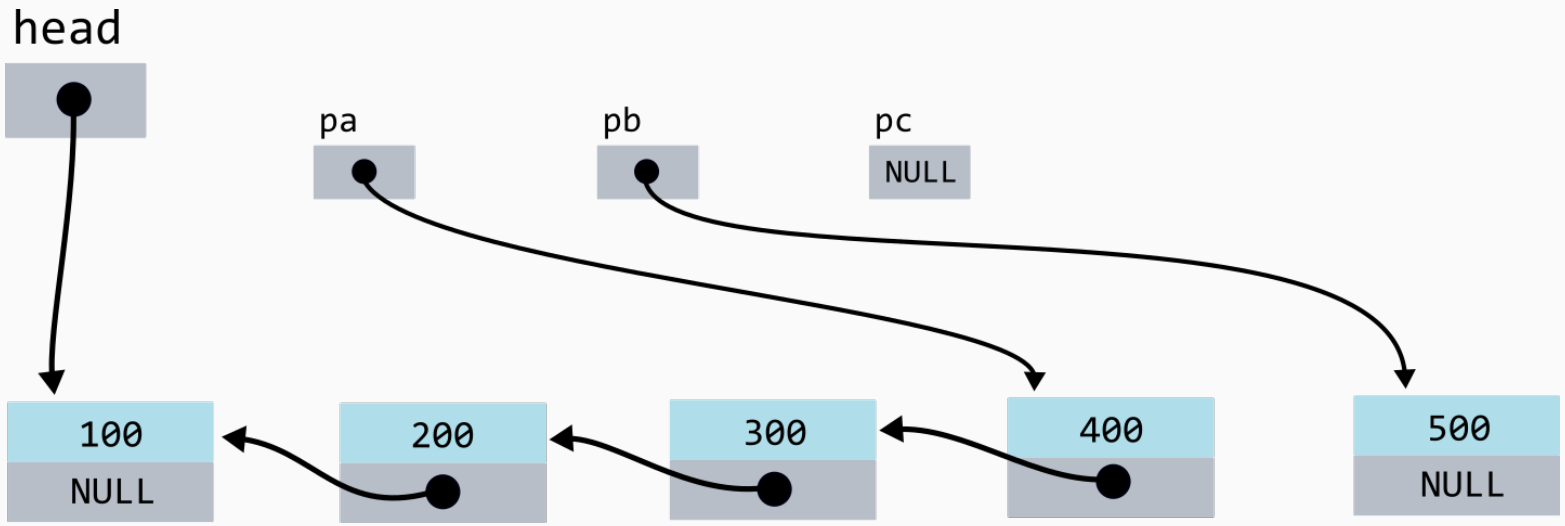
Обращение связного списка



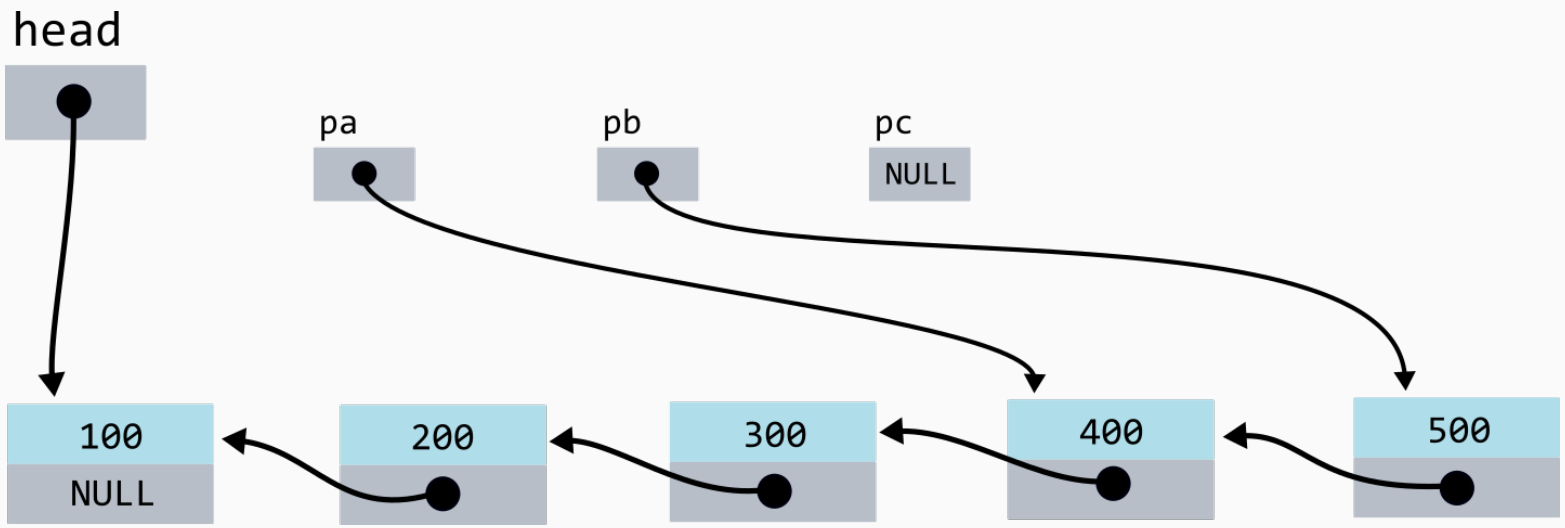
Обращение связного списка



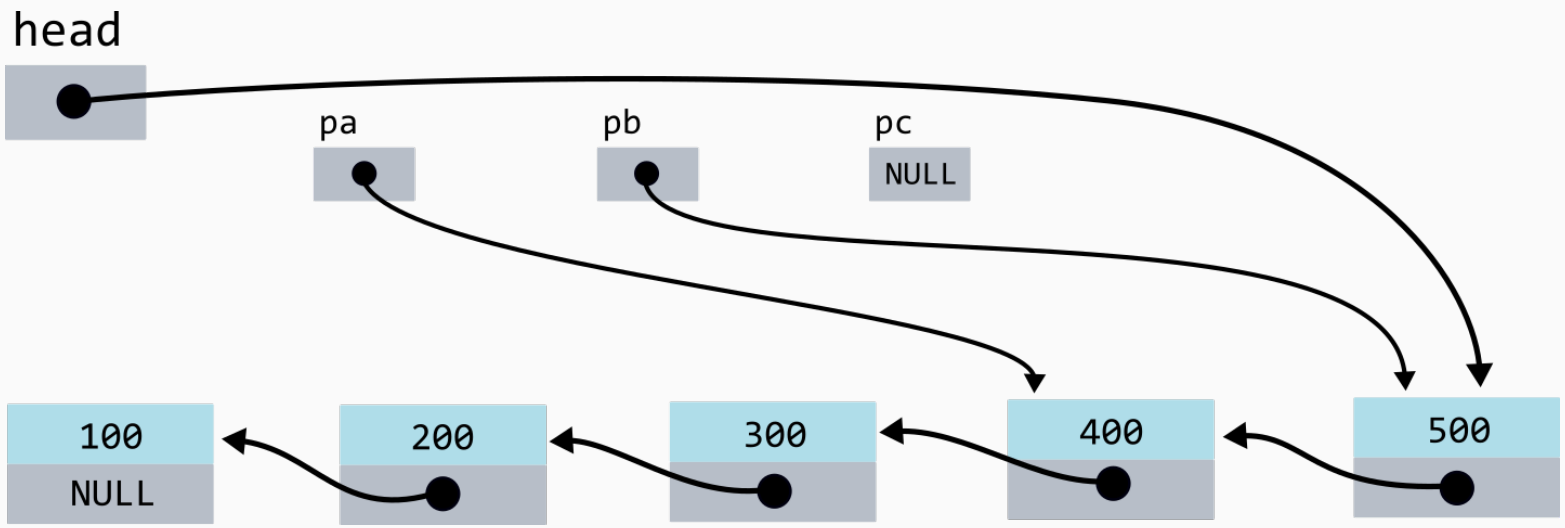
Обращение связного списка



Обращение связного списка



Обращение связного списка



Вычислительная сложность операций со связным списком

Вычислительная сложность операций со связным списком

Операция	Массив	Односвязный список
Доступ по номеру	$O(1)$	$O(N)$
Поиск	$O(N)$	$O(N)$
Вставка в начало	$O(N)$	$O(1)$
Вставка в конец	$O(1)$	$O(N)$
Вставка в конец если известен указатель на последний элемент	$O(1)$	$O(1)$
Вставка в середину	$O(N)$	$O(N)$
Вставка в середину если известен указатель на предыдущий элемент	$O(N)$	$O(1)$

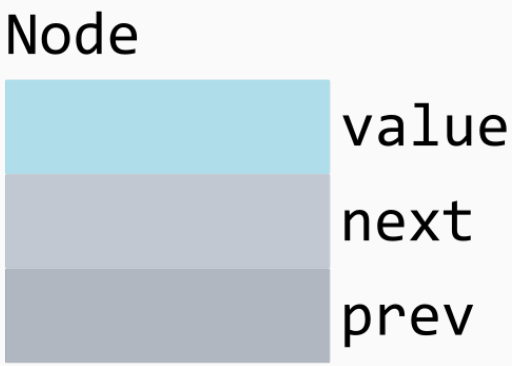
Недостатки связного списка

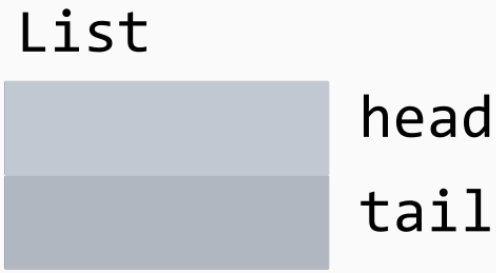
- В отличии от массива долгий доступ по индексу ($O(N)$)
- Чтобы вставить в конец списка, нужно пробежать до конца списка, а это долго ($O(N)$)
- С удаление из конца списка то же самое ($O(N)$)
- Если известен указатель на элемент списка, то можно быстро вставить элемент после него, но нельзя быстро вставить элемент до него (только за $O(N)$).
- Если известен указатель на элемент списка, то нельзя быстро удалить его, так как нужно знать указатель на предыдущий элемент.

Двусвязный список решает эти недостатки (кроме долгого доступа по индексу)

Двусвязный список

Двусвязный список: Определение

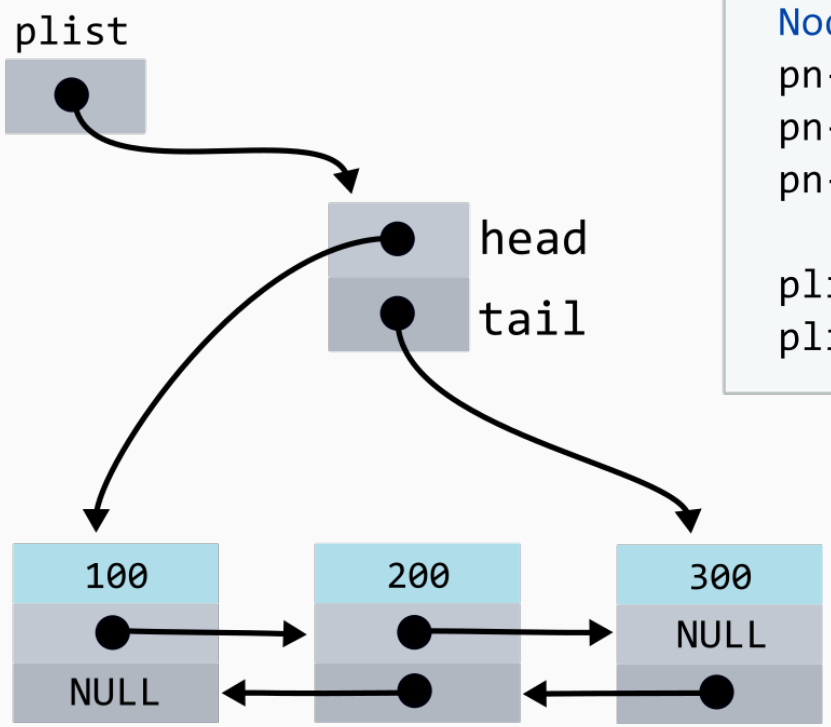





```c
struct node {
    int value;
    struct node* next;
    struct node* prev;
};
typedef struct node Node;

struct list {
    Node* head;
    Node* tail;
};
typedef struct list List;
```

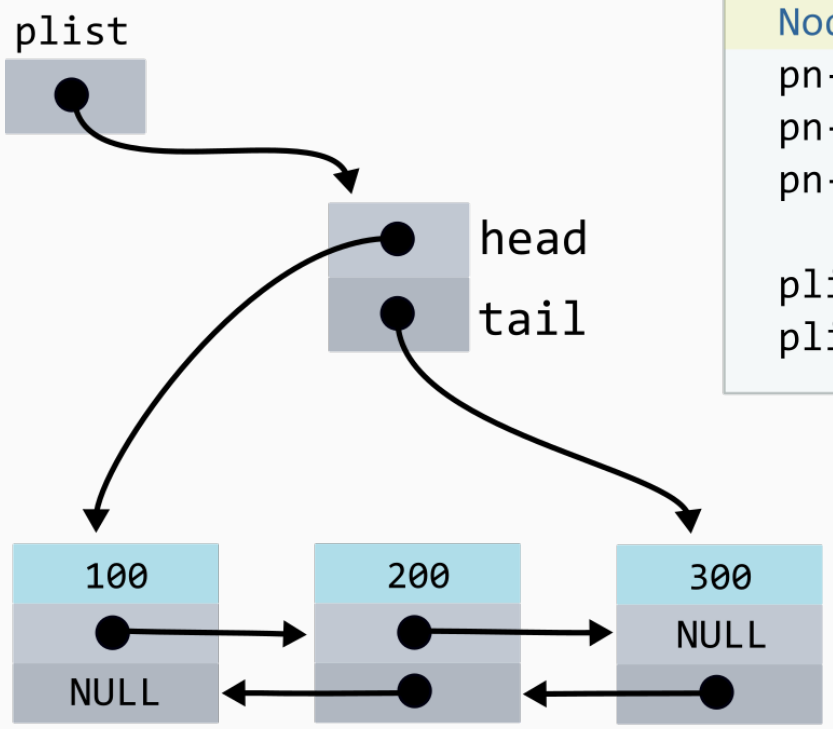
Добавление элемент в конец двусвязного списка

Добавление элемент в конец двусвязного списка

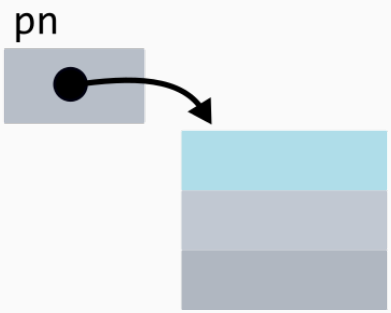



```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

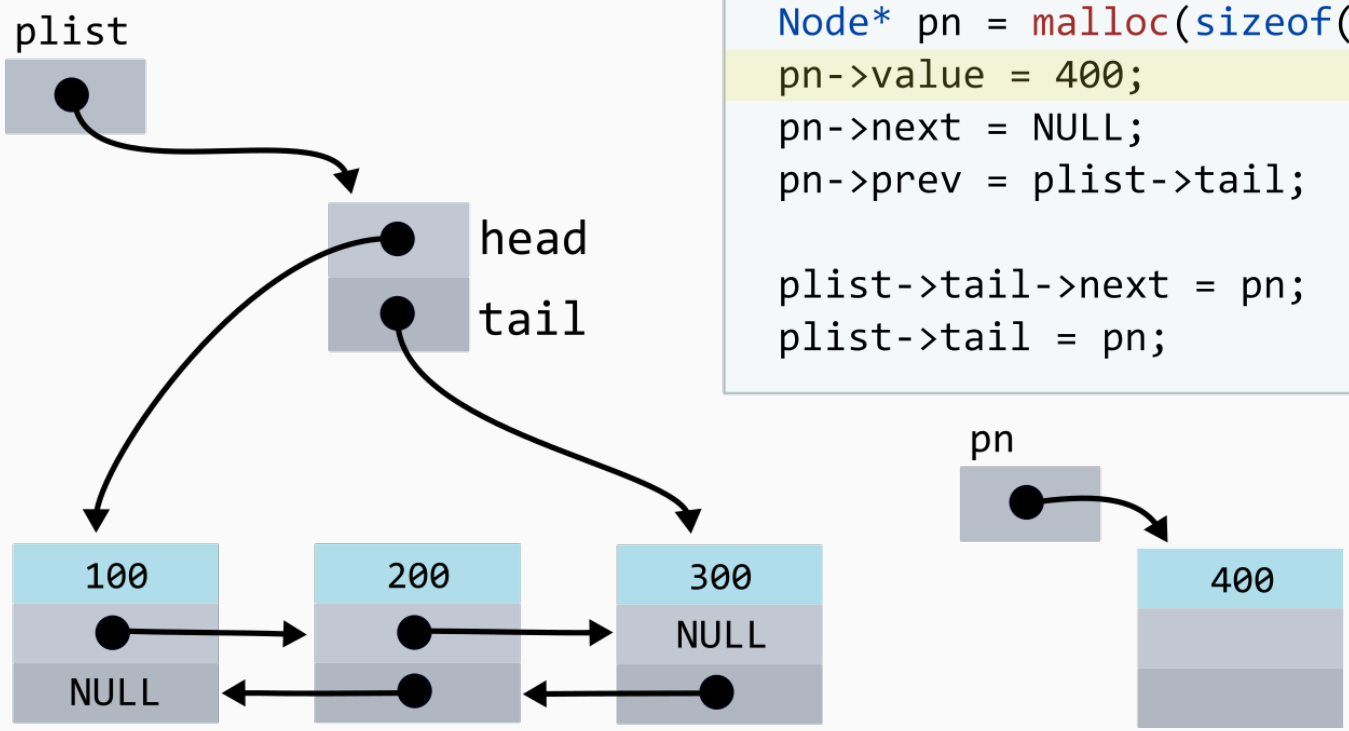
Добавление элемент в конец двусвязного списка




```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

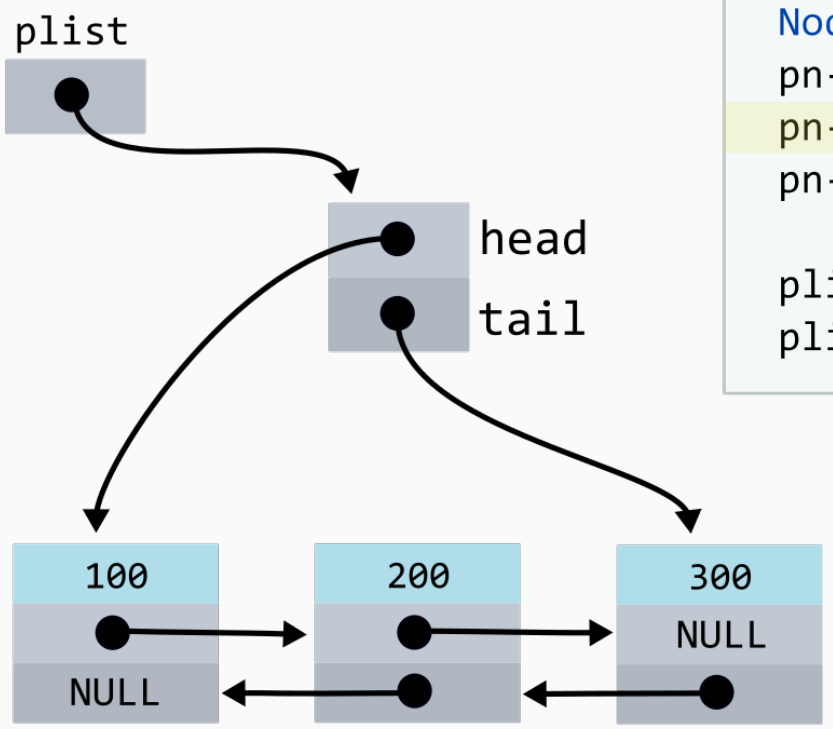


Добавление элемент в конец двусвязного списка

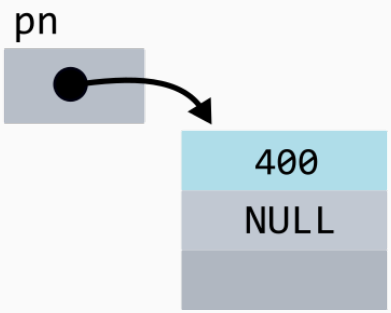



```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

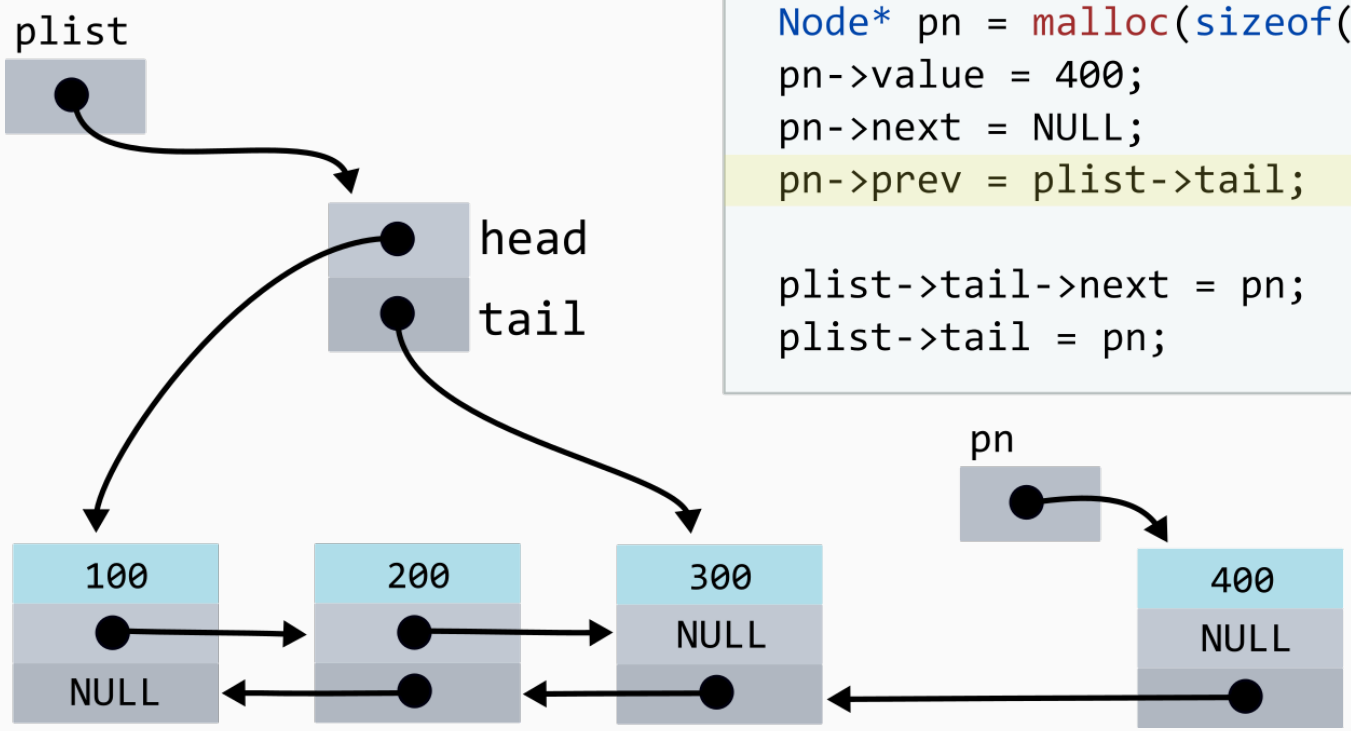
Добавление элемент в конец двусвязного списка




```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

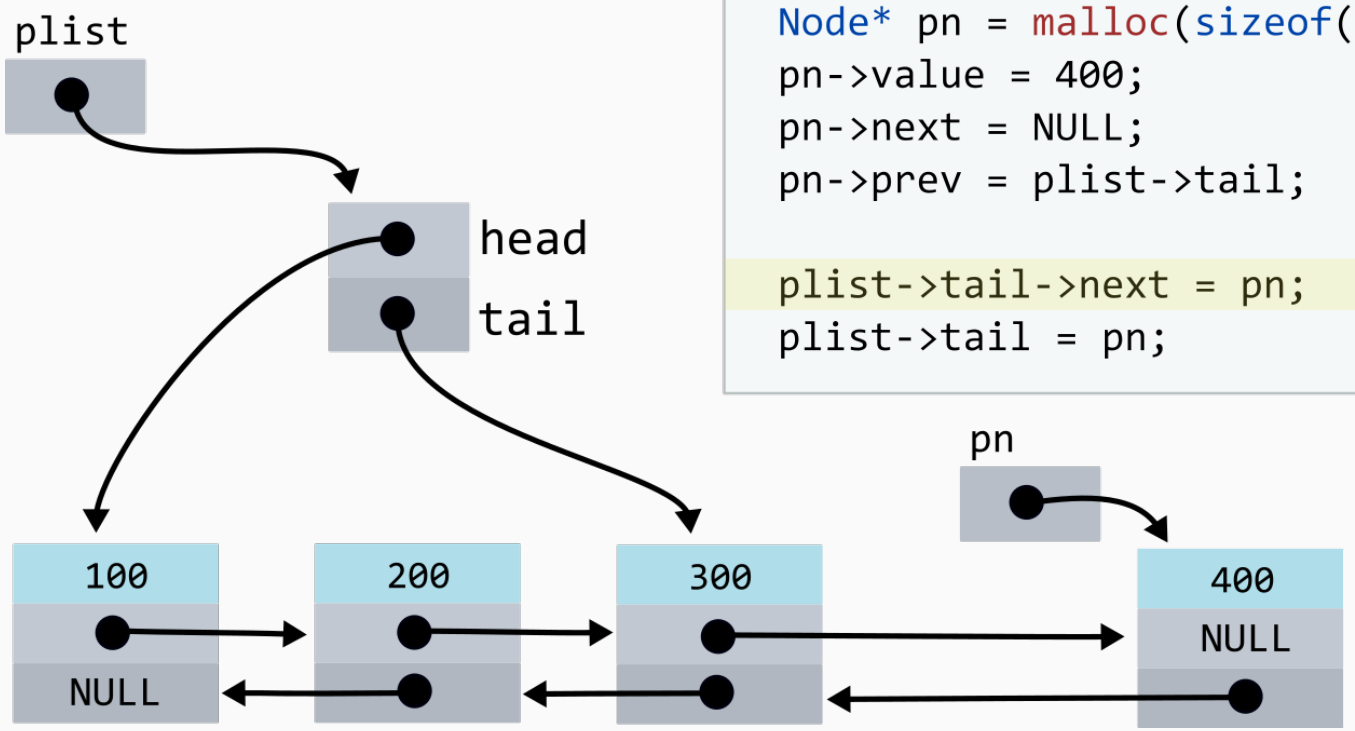


Добавление элемент в конец двусвязного списка



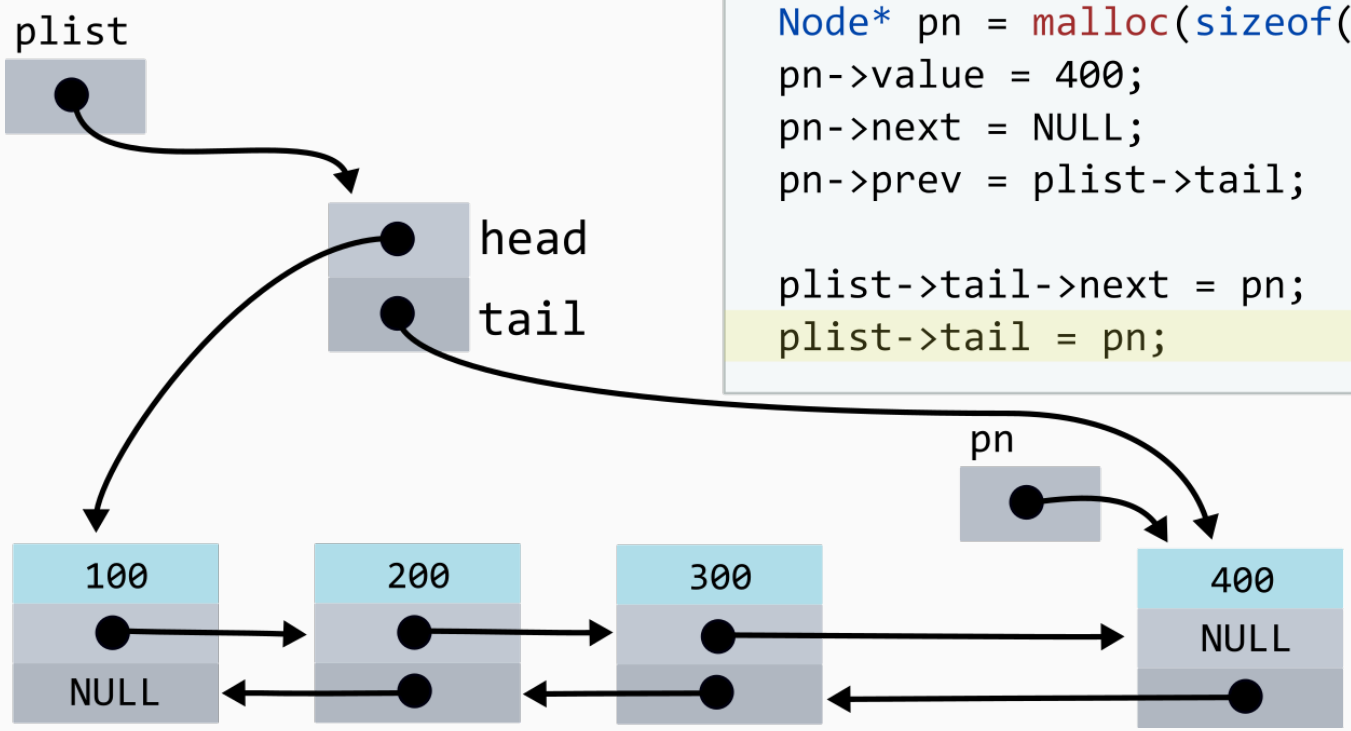

```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

Добавление элемент в конец двусвязного списка



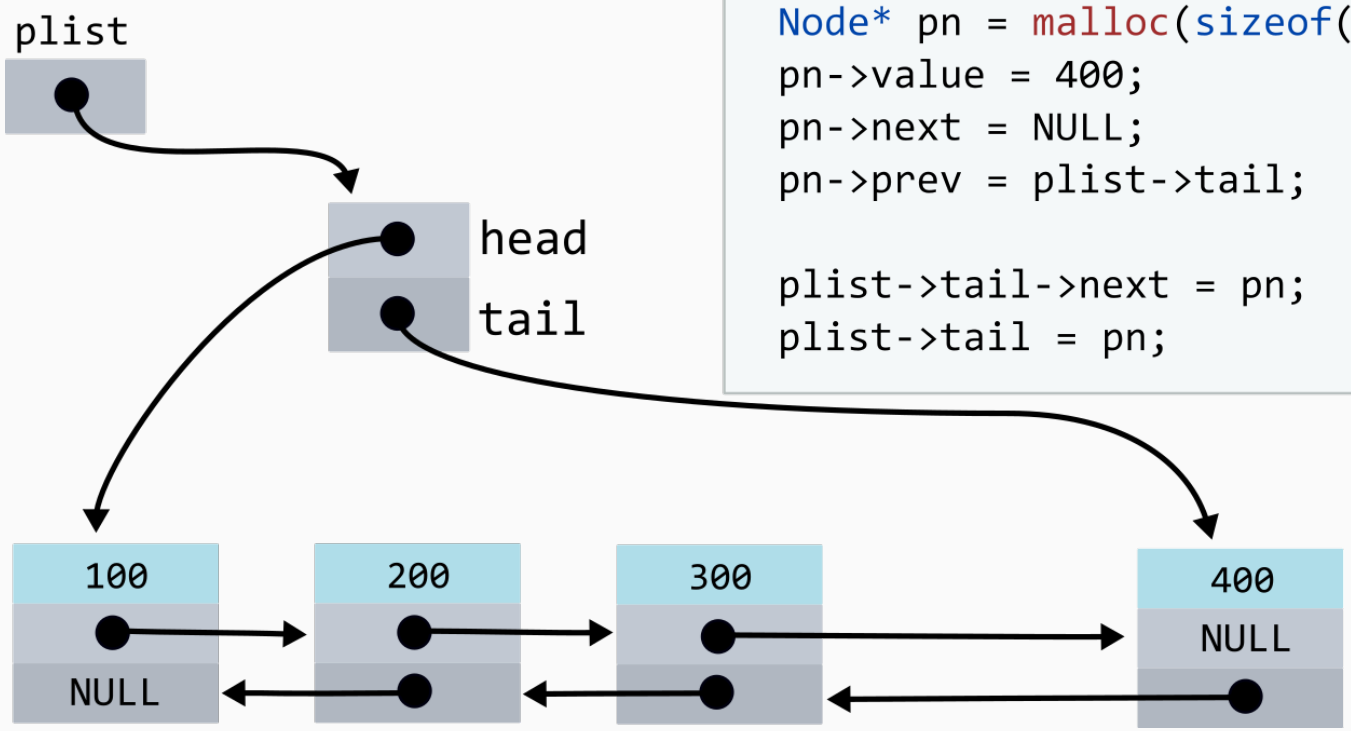

```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

Добавление элемент в конец двусвязного списка




```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

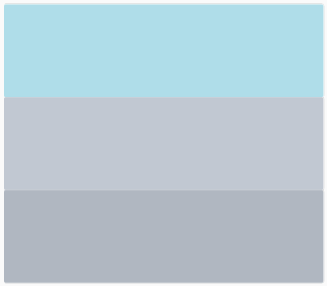
Добавление элемент в конец двусвязного списка




```
Node* pn = malloc(sizeof(Node));
pn->value = 400;
pn->next = NULL;
pn->prev = plist->tail;

plist->tail->next = pn;
plist->tail = pn;
```

Циклический двусвязный список

Циклический двусвязный список: Определение




```c
struct node {
    int value;
    struct node* next;
    struct node* prev;
};
typedef struct node Node;
```

Циклический двусвязный список