Лекция 5
Связные списки

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Структуры и алгоритмы обработки данных»
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)
Весенний семестр, 2016

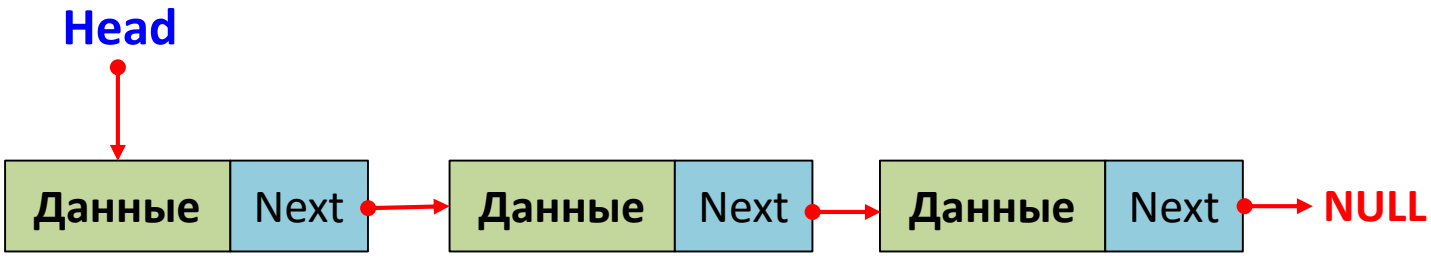
Связные списки (Linked lists)

- **Связный список (linked list) –** динамическая структура данных для хранения информации, в которой каждый элемент хранит указатели на один или несколько других элементов

Операция	Описание	Вычислительная сложность	Сложность по памяти
AddFront(*L*, *x*)	Добавляет элемент *x* в начало списка *L*	*O*(1)	*O*(1)
AddEnd(*L*, *x*)	Добавляет элемент *x* в конец списка *L*	*O*(*n*)	*O*(1)
Lookup(*L*, *x*)	Отыскивает элемент *x* в списке *L*	*O*(*n*)	*O*(1)
Size(*L*)	Возвращает количество элементов в списке *L*	*O*(1) или *O*(*n*)	*O*(1)

Односвязный список (Singly linked list)

- Размер списка заранее не известен – элементы добавляются во время работы программы (динамически)
- Память под элементы выделяется динамически (функции: malloc, calloc, free)



Односвязный список (Singly linked list)

```c
#include <stdio.h>
#include <stdlib.h>

struct listnode {
    char *data;                 /* Data */
    int value;                  /* Data */
    struct listnode *next;      /* Next node */
};
```

Создание элемента (выделение памяти)

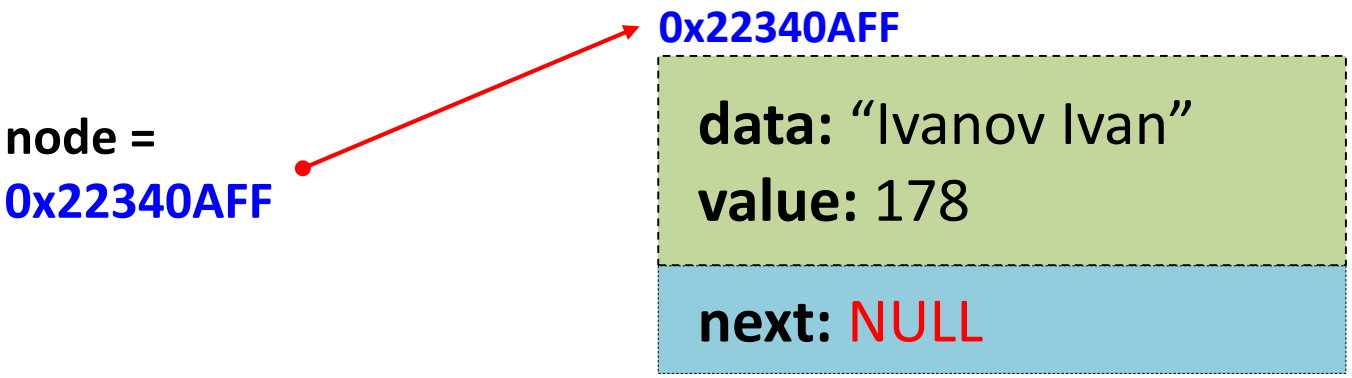
```
struct listnode *list_createnode(char *data,
                                 int value)
{
    struct listnode *p;

    p = malloc(sizeof(*p));    // Выделяем память
    if (p != NULL) {
        p->data = data;
        p->value = value;
        p->next = NULL;
    }
    return p;
}
```

Сложность создания элемента

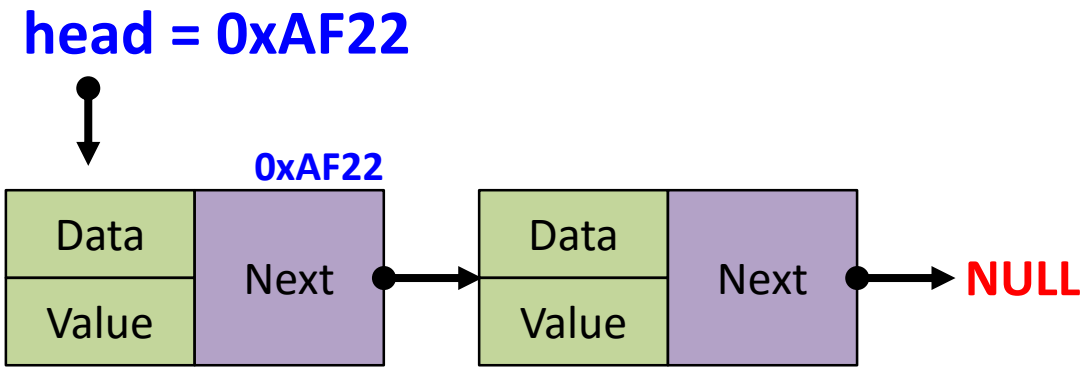
$T_{CreateNode} = O(1)$

Создание элемента (выделение памяти)

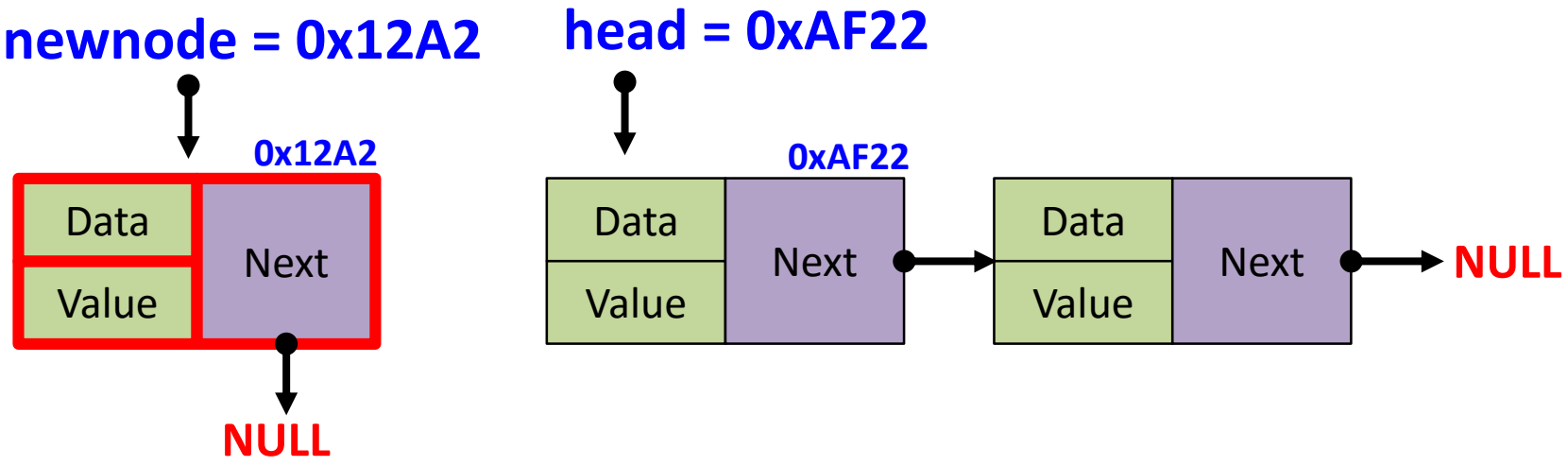
```
int main()
{
    struct listnode *node;

    /* Список из одного элемента */
    node = list_createnode("Ivanov Ivan", 178);

    return 0;
}
```



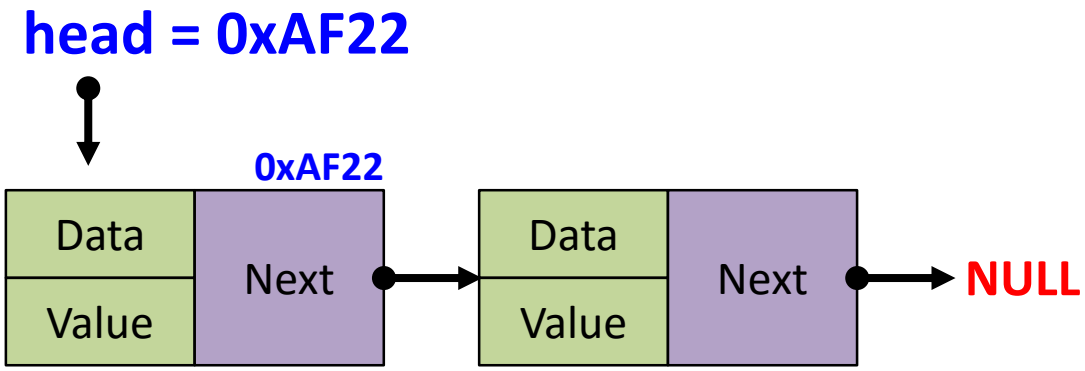
Добавление элемента в начало списка



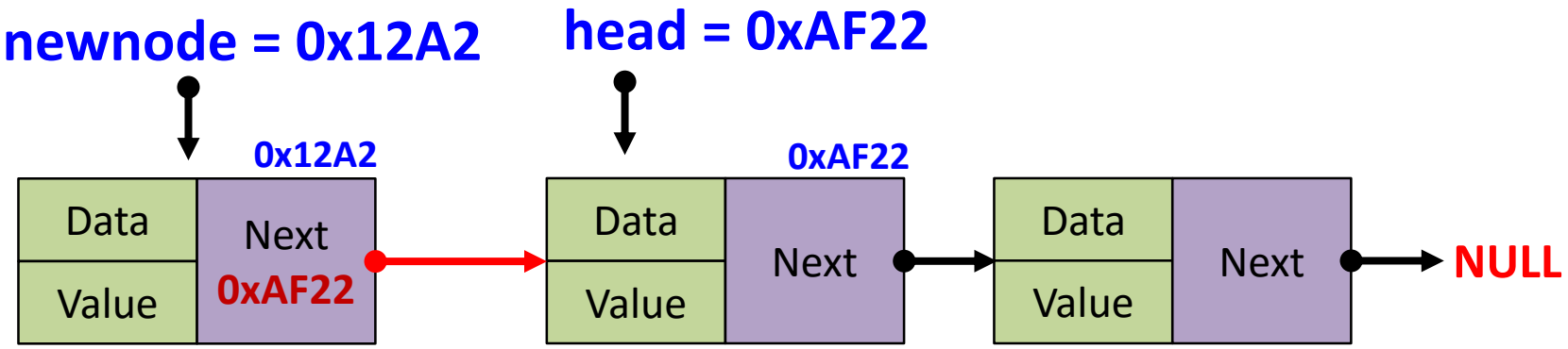
1. Создаем новый узел **newnode** в памяти



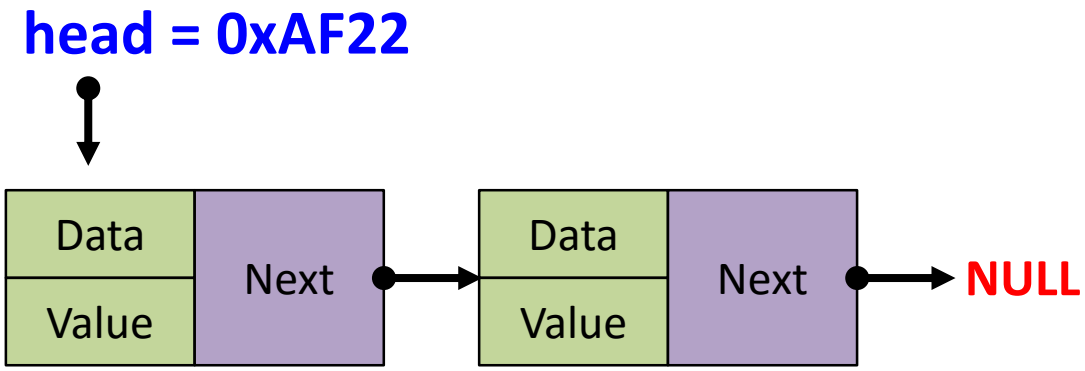
Добавление элемента в начало списка



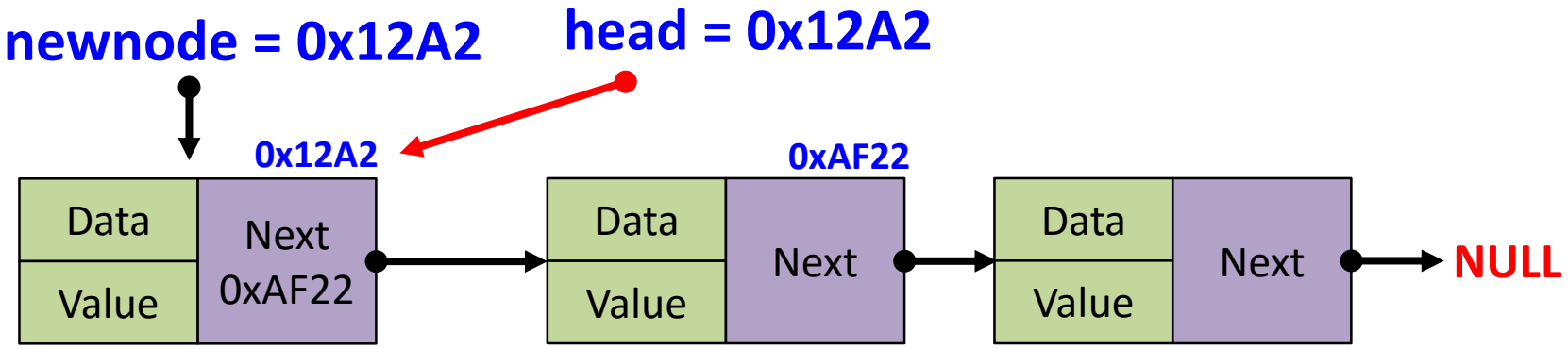
2. Устанавливаем указатель **next** узла **newnode** на **head**



Добавление элемента в начало списка



3. Делаем головой списка узел **newnode**



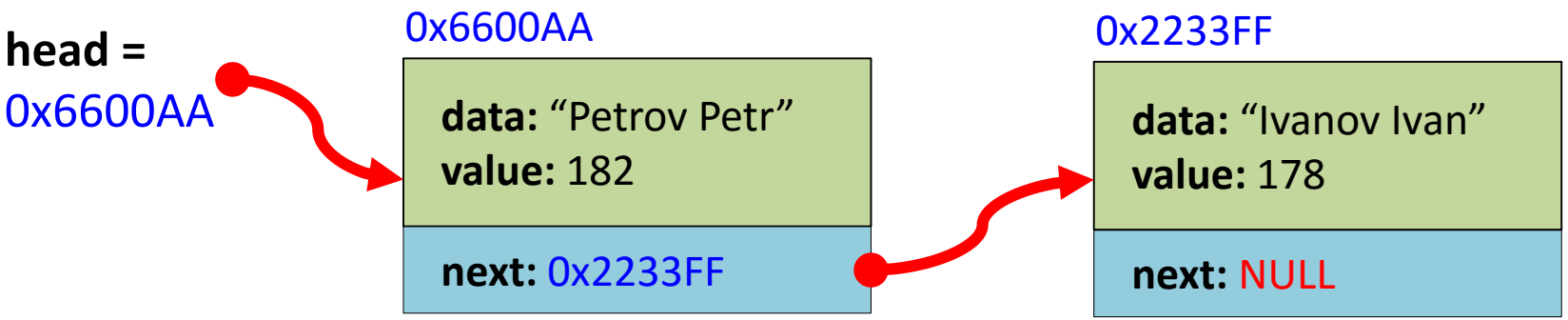
Добавление элемента в начало списка

```
struct listnode *list_addfront(
    struct listnode *list,
    char *data, int value)
{
    struct listnode *newnode;
    newnode = list_createnode(data, value);

    if (newnode != NULL) {
        newnode->next = list;
        return newnode;
    }
    return list;
}
```

$T_{AddFront} = O(1)$

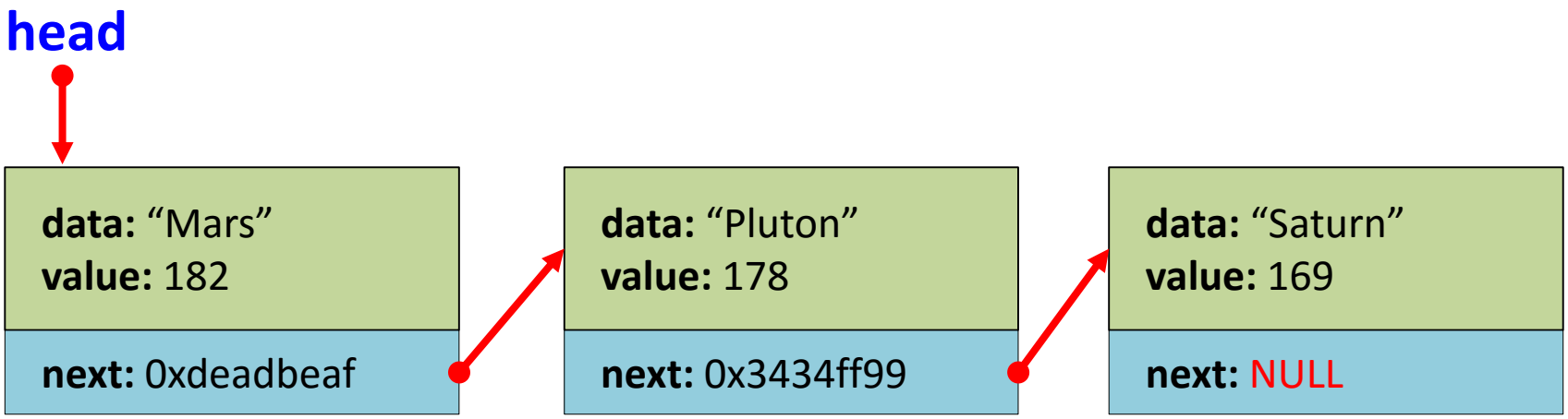
Добавление элемента в начало списка

```
int main()
{
    struct listnode *head;

    head = list_addfront(NULL, "Ivanov Ivan", 178);
    head = list_addfront(head, "Petrov Petr", 182);

    return 0;
}
```



Поиск элемента в списке (Lookup)

- Начиная с головы списка просматриваем все узлы и сравниваем ключи
- В худшем случае требуется просмотреть все узлы, это требует $O(n)$ операций



Поиск элемента в списке (Lookup)

```
struct listnode *list_lookup(
    struct listnode *list,
    char *data, int value)
{
    for ( ; list != NULL; list = list->next) {
        if (strcmp(list->data, data) == 0 &&
            list->value == value)
        {
            return list;
        }
    }
    return NULL;  // Не нашли
}
```

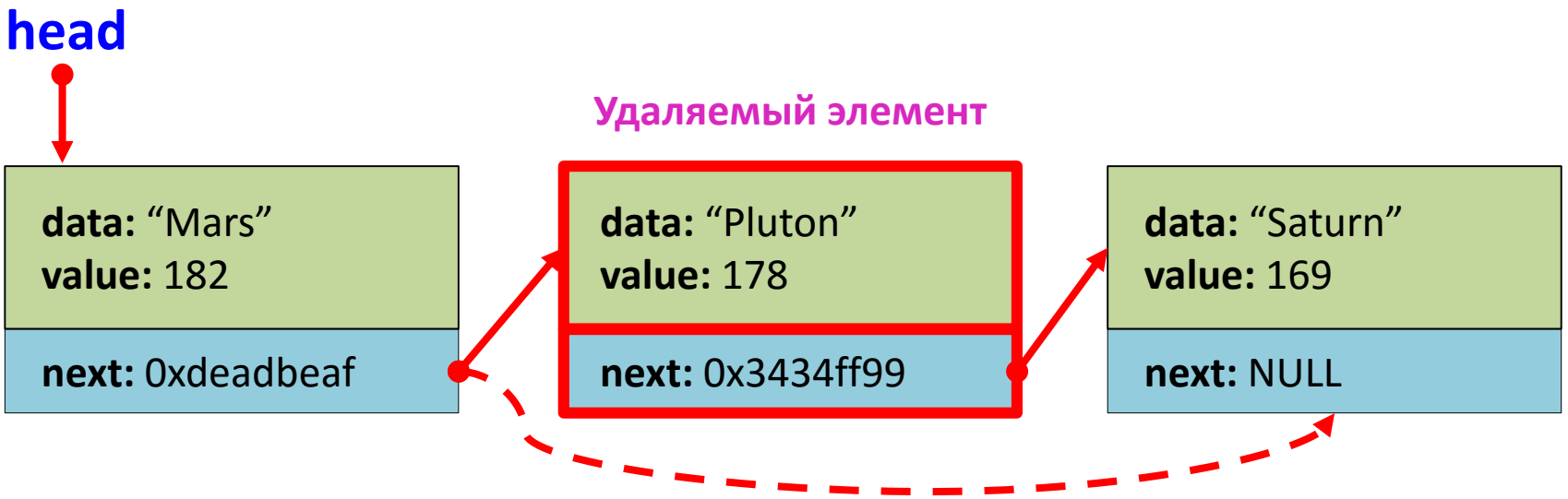
$T_{Lookup} = O(n)$

Поиск элемента в списке (Lookup)

```
int main()
{
    struct listnode *node, *p;

    node = list_addfront(NULL, “Mars", 178);
    node = list_addfront(node, “Pluton", 182);
    node = list_addfront(node, “Saturn", 169);

    p = list_lookup(node, “Pluton", 182);
    if (p != NULL) {
        printf("Data: %s\n", p->data);
    }

    return 0;
}
```

Удаление элемента (Delete)

1. Находим элемент в списке (за время $O(n)$)
2. Корректируем указатели (за $O(1)$)
3. Удаляем элемент из памяти (за $O(1)$)



Три возможных ситуации:

- ❑ удаляемый узел – в начале списка
- ❑ удаляемый узел – внутренний узел (есть элементы слава и справа)
- ❑ удаляемый узел – в конце списка

Удаление элемента (Delete)

```
struct listnode *list_delete(struct listnode *list,
                             char *data, int value)
{
    struct listnode *p, *prev = NULL;

    for (p = list; p != NULL; p = p->next) {
        if (strcmp(p->data, data) == 0 && p->value == value) {
            if (prev == NULL)
                list = p->next;        // Удаляем голову
            else
                prev->next = p->next;  // Есть элемент слева
            free(p);                   // Освобождаем память
            return list;   // Указатель на новую голову
        }
        prev = p;  // Запоминаем предыдущий элемент (левый)
    }
    return NULL;   // Не нашли
}
```

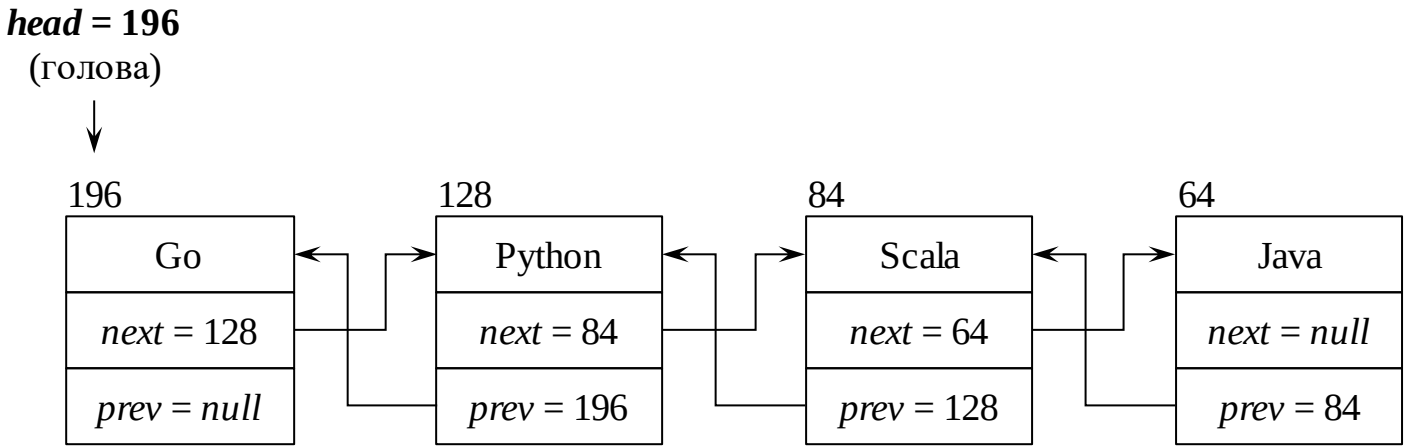
$$T_{Delete} = O(n)$$

Удаление элемента

```
int main()
{
    struct listnode *node, *p;

    node = list_addfront(NULL, “Mars", 178);
    node = list_addfront(node, “Pluton", 182);
    node = list_addfront(node, “Saturn", 169);

    p = list_delete(node, “Pluton", 182);
    if (p != NULL) {
        node = p;  // Указатель на новую голову
        printf(“Item deleted\n");
    }
    return 0;
}
```

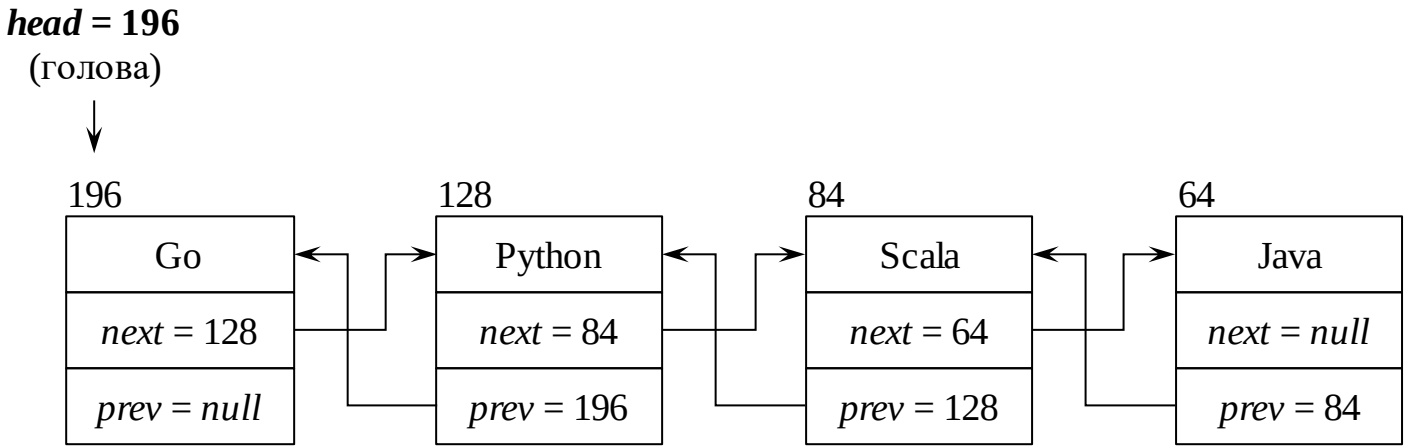
Двусвязные списки

- Каждый узел двусвязного списка имеет три поля: *value*, *next* и *prev*.
- Поле *value* – это некоторые данные, ассоциированные с узлом
- В поле *next* хранится адрес узла, следующего за текущим, а в поле *prev* – адрес предшествующего узла



Двусвязные списки

- Каждый узел двусвязного списка имеет три поля: *value*, *next* и *prev*.
- Поле *value* – это некоторые данные, ассоциированные с узлом
- В поле *next* хранится адрес узла, следующего за текущим, а в поле *prev* – адрес предшествующего узла



Создание нового узла

```
1  function DoublyLinkedListCreateNode(value)
2      node = AllocateMemory()        /* Выделяем память под узел */
3      node.value = value
4      node.next = null
5      node.prev = null
6      return node
7  end function
```

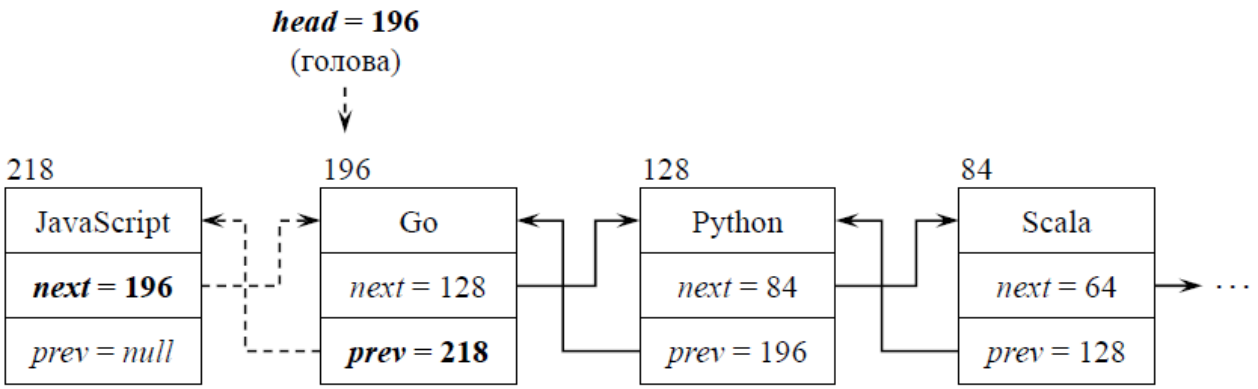
Вычислительная сложность: O(1)

Добавление узла в начало списка

- Функция создает в памяти новый узел с заданным значением поля *value*
- В поле *next* нового узла заносится адрес головы списка
- Если список не пуст, то необходим записать в указатель *prev* первого узла адрес нового элемента

```
1  function DoublyLinkedListAddFront(list, value)
2      node = LinkedListCreateNode(value)
3      node.next = list            /* Делаем головой списка новый узел */
4      if list ≠ null then
5          list.prev = node
6      end if
7      return node
8  end function
```

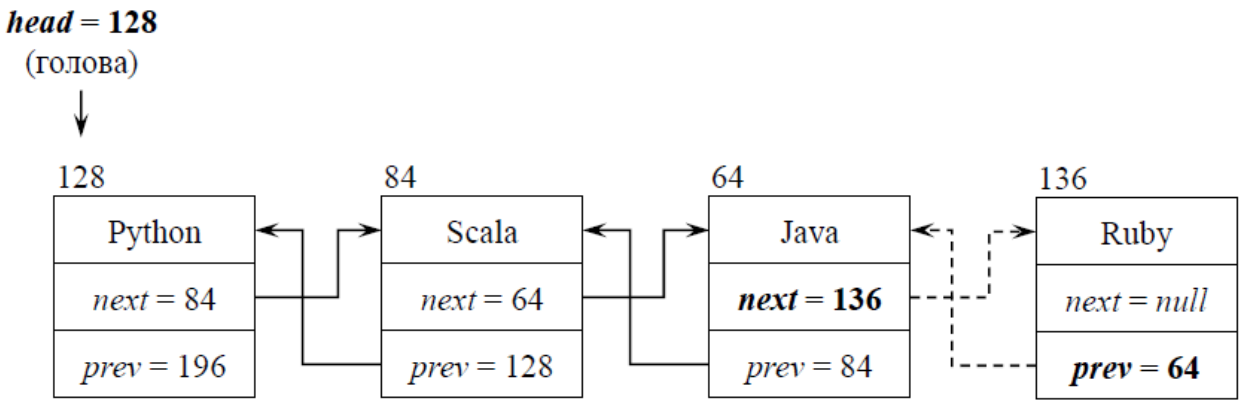
Вычислительная сложность: O(1)



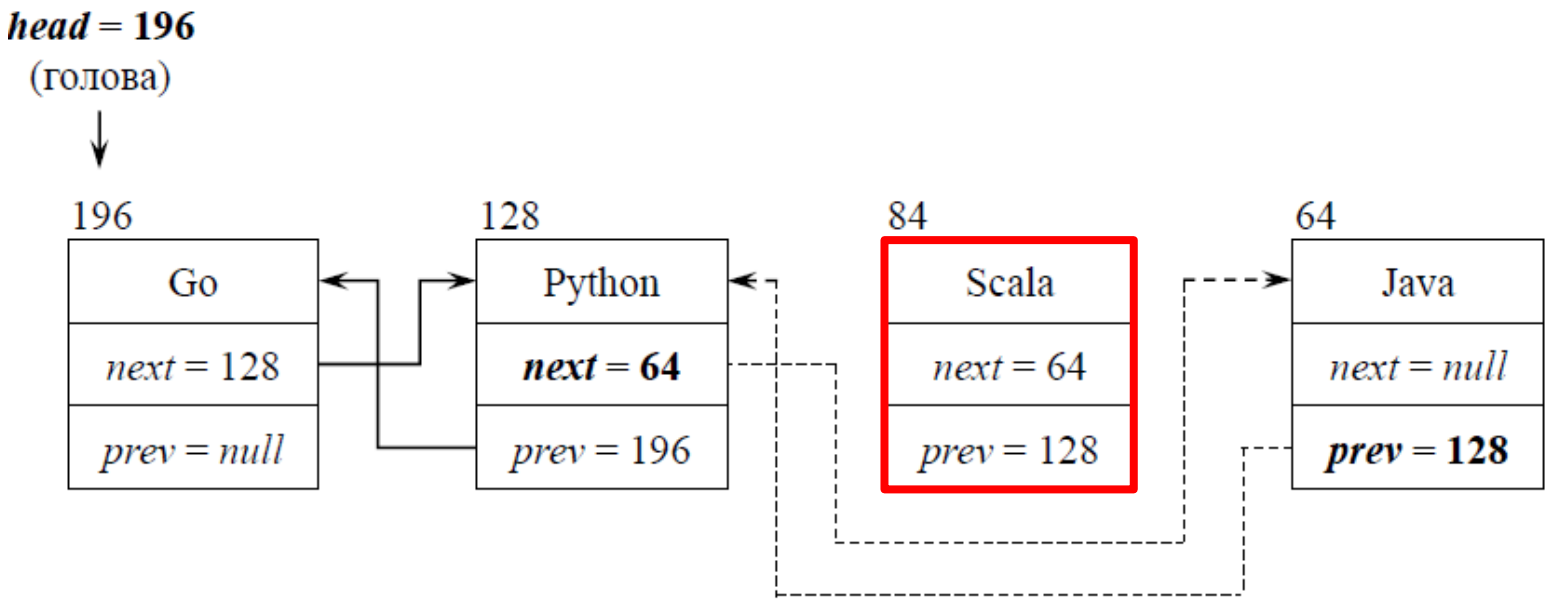
Добавление узла в конец списка

```
1  function DoublyLinkedListAddEnd(list, value)
2      newnode = LinkedListCreateNode(value)
3      if list = null then
4          return newnode
5      end if
6      node = list                          /* Поиск последнего узла */
7      while node.next ≠ null do
8          node = node.next
9      end while
10     node.next = newnode                  /* Связываем с новым узлом */
11     newnode.prev = node
12     return list
13 end function
```

Вычислительная сложность: O(*n*)



Удаление узла



Удаление узла

```
1  function DoublyLinkedListDelete(list, value)
2      node = list
3      while node ≠ null do                    /* Отыскиваем удаляемый узел */
4          if node.value = value then
5              if node.prev = null then
6                  list = node.next    /* Удаляемый узел – голова списка */
7              else
8                  node.prev.next = node.next
9              end if
10             if node.next ≠ null then
11                 node.next.prev = node.prev
12             end if
13             FreeMemory(node)                /* Освобождаем память */
14             return list
15         end if
16         node = node.next
17     end while
18     return null                                  /* Узел не найден */
19 end function
```

Вычислительная сложность: O(*n*)

Домашнее чтение

- Прочитать о реализации связных списков в “практике программирования” [Kernighan2011, C. 61-66]
- [DSABook, Глава 6]