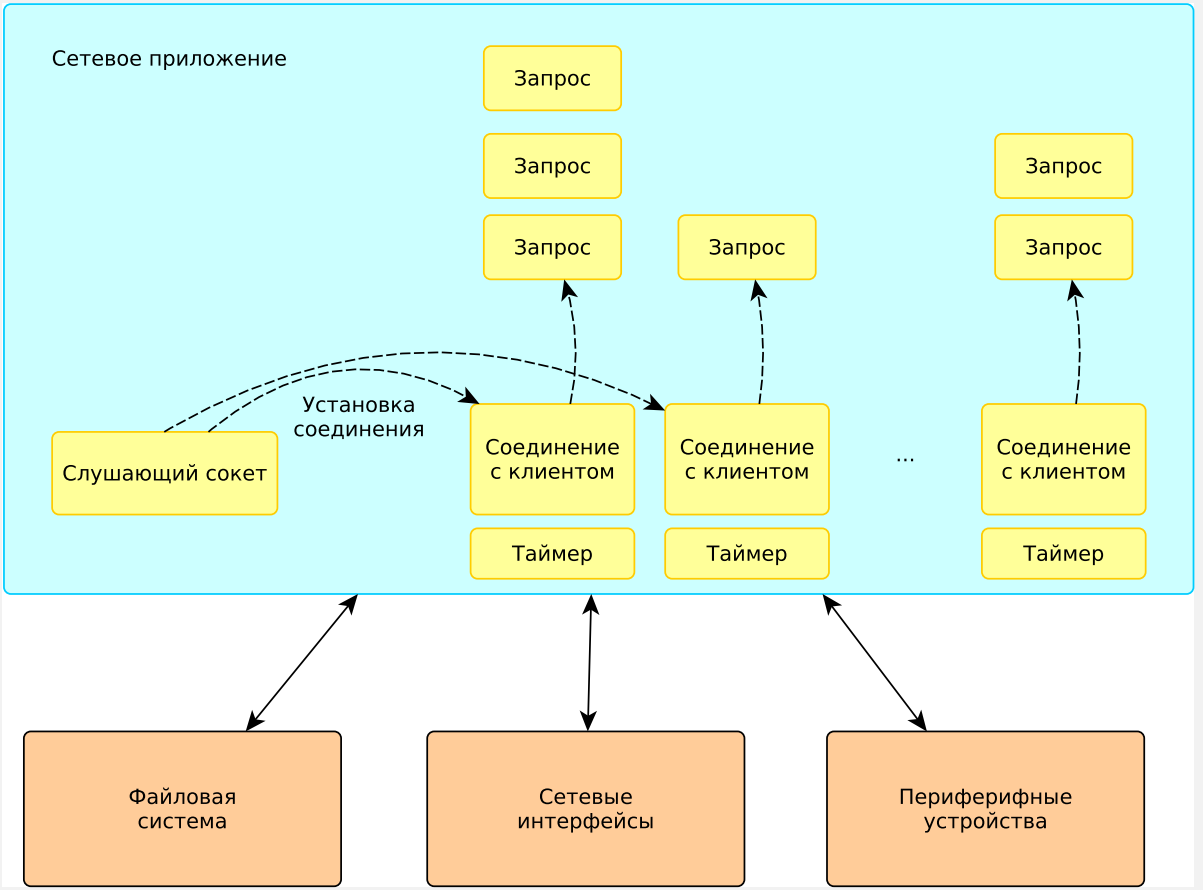
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

17.05.19

Структура сетевого приложения



Контейнеры

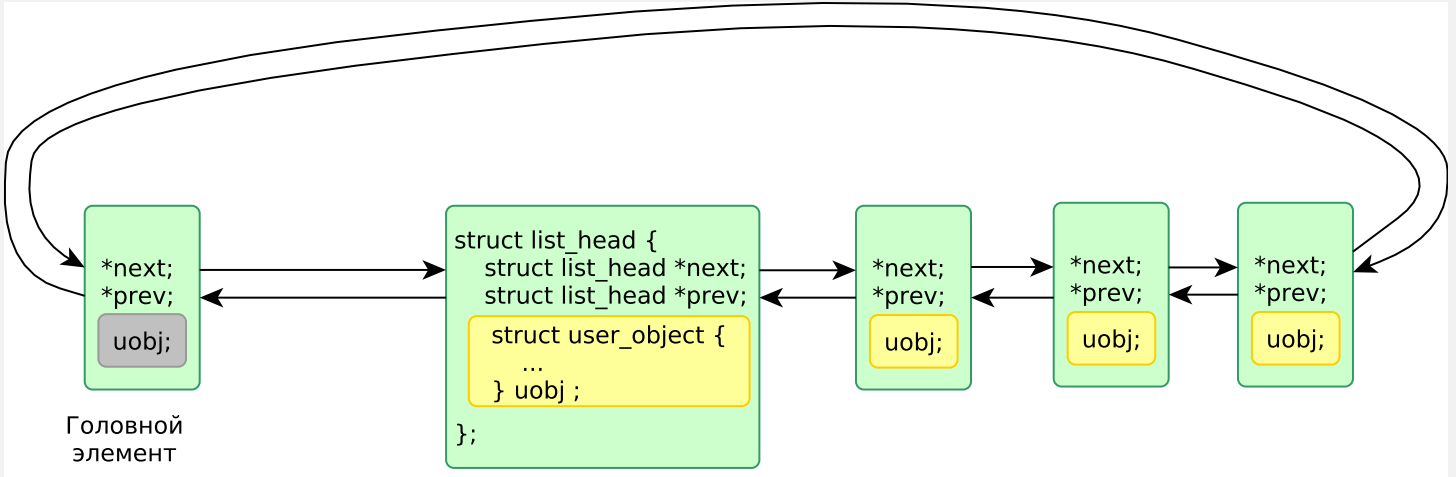
Что нужно?

- Динамическое добавление и удаление элементов.
- Извлечение элемента по номеру или ключу.
- Перебор всех элементов контейнера.

Виды контейнеров.

- Списки - последовательность элементов, связанных указателями. Обеспечивает быстрое добавление и поиск по порядковому номеру.
- Словари (ассоциативные массивы) - множество пар ключ-значение. Обеспечивает быстрый поиск по ключу.

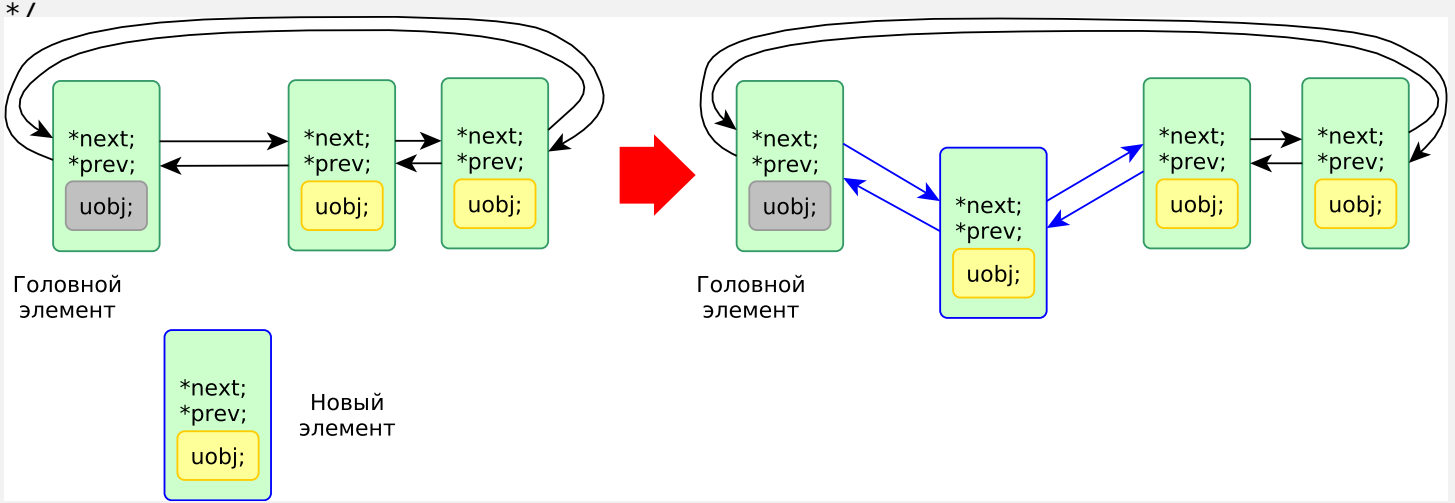
Простой список



Добавление и удаление элементов списка

```
/* Добавление элемента new в начало */
struct list_head head;
struct list_head new;
...
new.prev = &head;
new.next = head.next
head.next->prev = &new;
head.next = &new;
/* То же самое: list_add(&new, &head);
*/
```

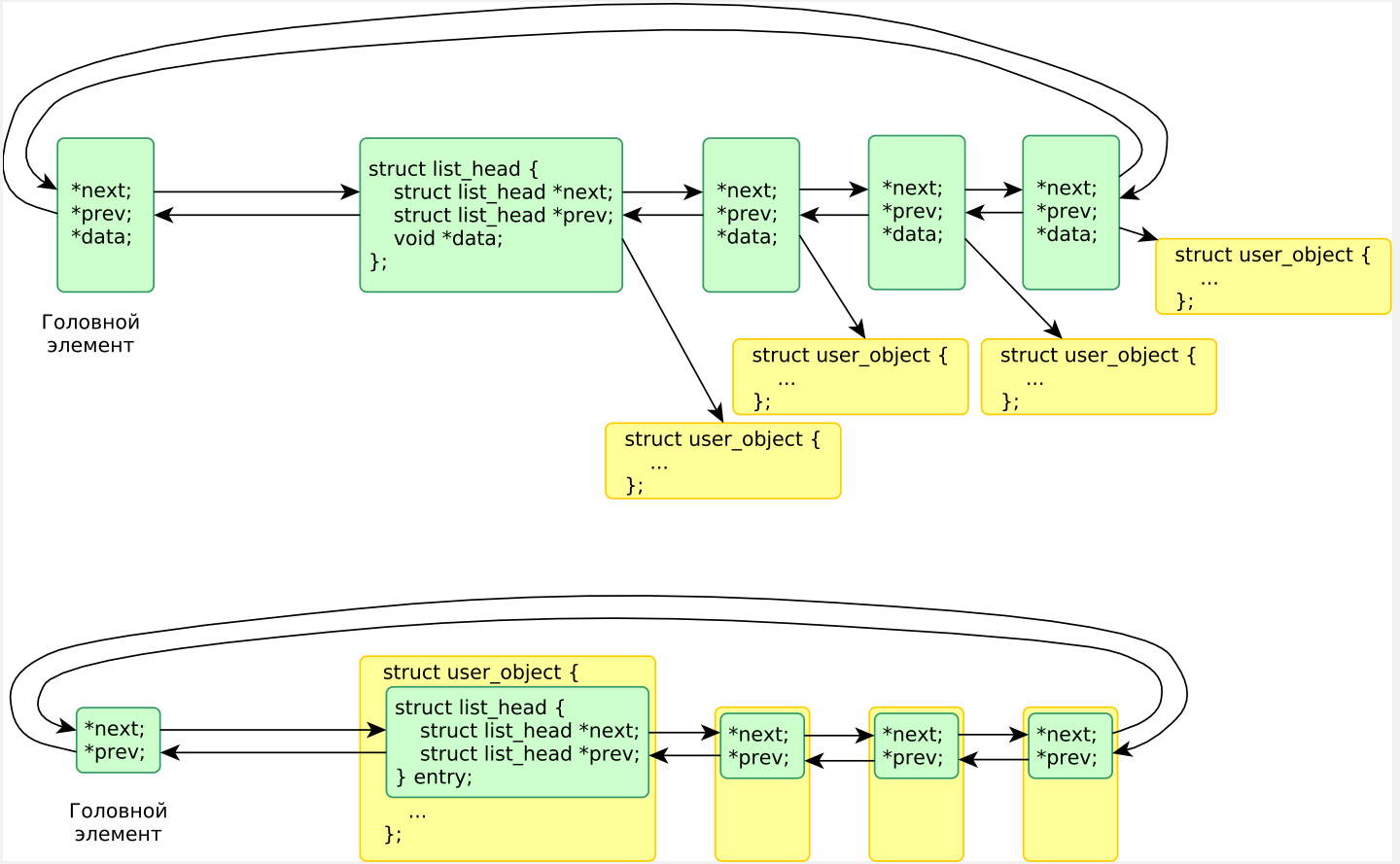
```
/* Удаление элемента new из списка */
struct list_head new;
...
new.prev->next = new.next;
new.next->prev = new.prev;
/* То же самое: list_del(&new); */
```



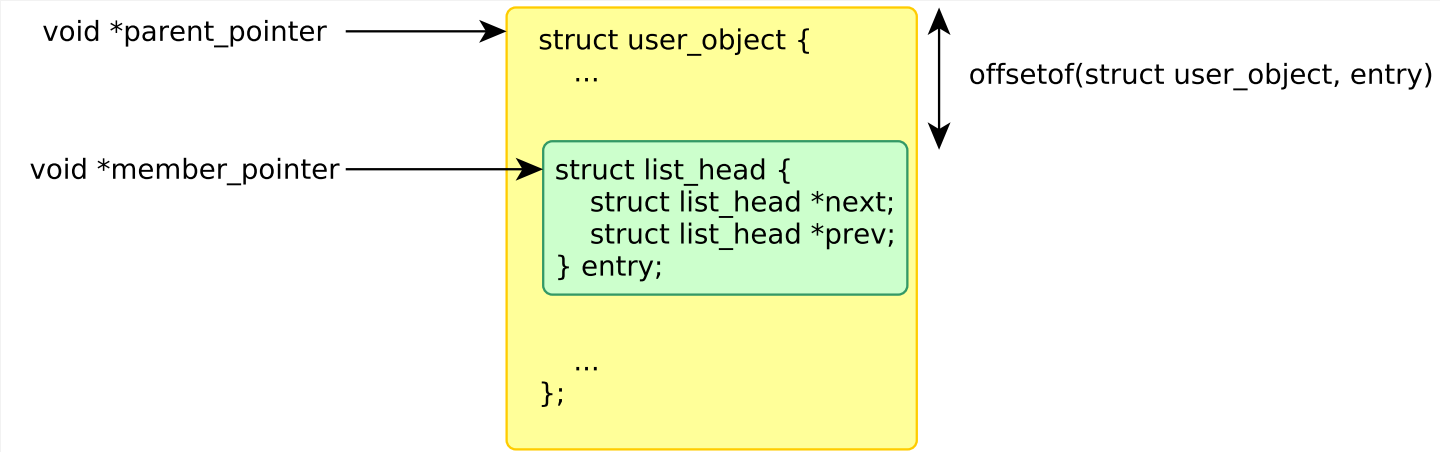
Перебор элементов простого списка

```
struct list_head {
       struct list_head *next;
       struct list_head *prev;

       struct user_object user_data;
} head, *ptr;

for (ptr = head.next;
     ptr->next != &head; \
     ptr = ptr->next) {

     struct user_object *uobj = &ptr->user_data;
     /* Операции с uobj */
}
```

Универсальный список



Получение указателя на родительский элемент




```
/* Смещение между адресом родительского объекта типа type и адресом его поля
member. */
#define offsetof(type, member)     ((size_t)&((type *)0)->member)

/* Получение указателя на родительский объект типа type из указателя ptr на его
поле member. */
#define container_of(ptr, type, member) ({              \
       void *__mptr = (void *)(ptr);                    \
       ((type *)(__mptr - offsetof(type, member))); })
```

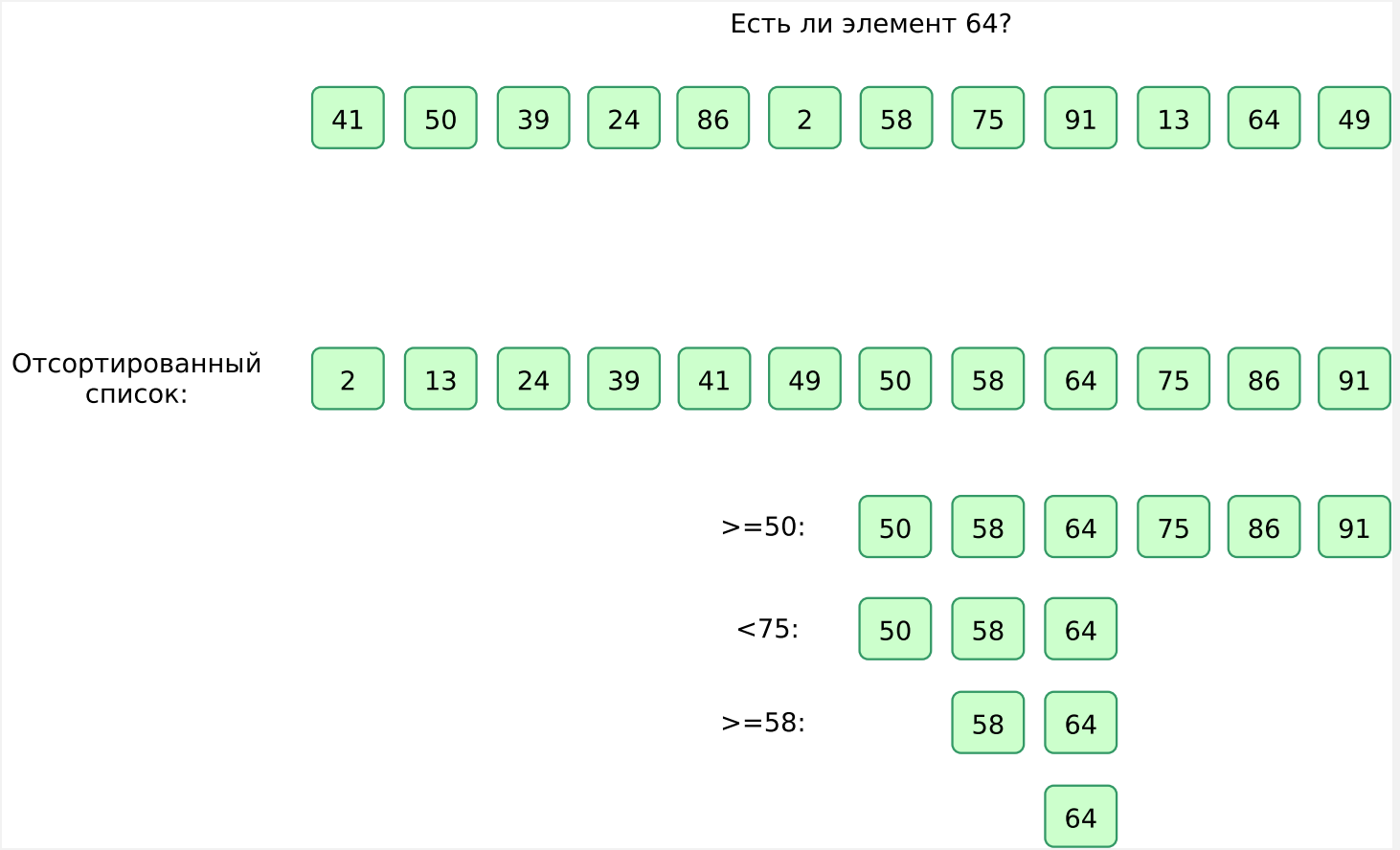
Перебор элементов универсального списка

```
struct list_head head;
struct user_object {
       void *data;
       struct list_head entry;
} *uobj;

for (uobj = container_of(head.next, typeof(*uobj), entry);
     &uobj->entry != &head; \
     uobj = container_of(uobj->entry.next, typeof(*uobj), entry)) {
     /* Операции с uobj */
}

/* Тот же самый перебор. */
list_for_each_entry(uobj, &head, entry) {
     /* Операции с uobj */
}
```

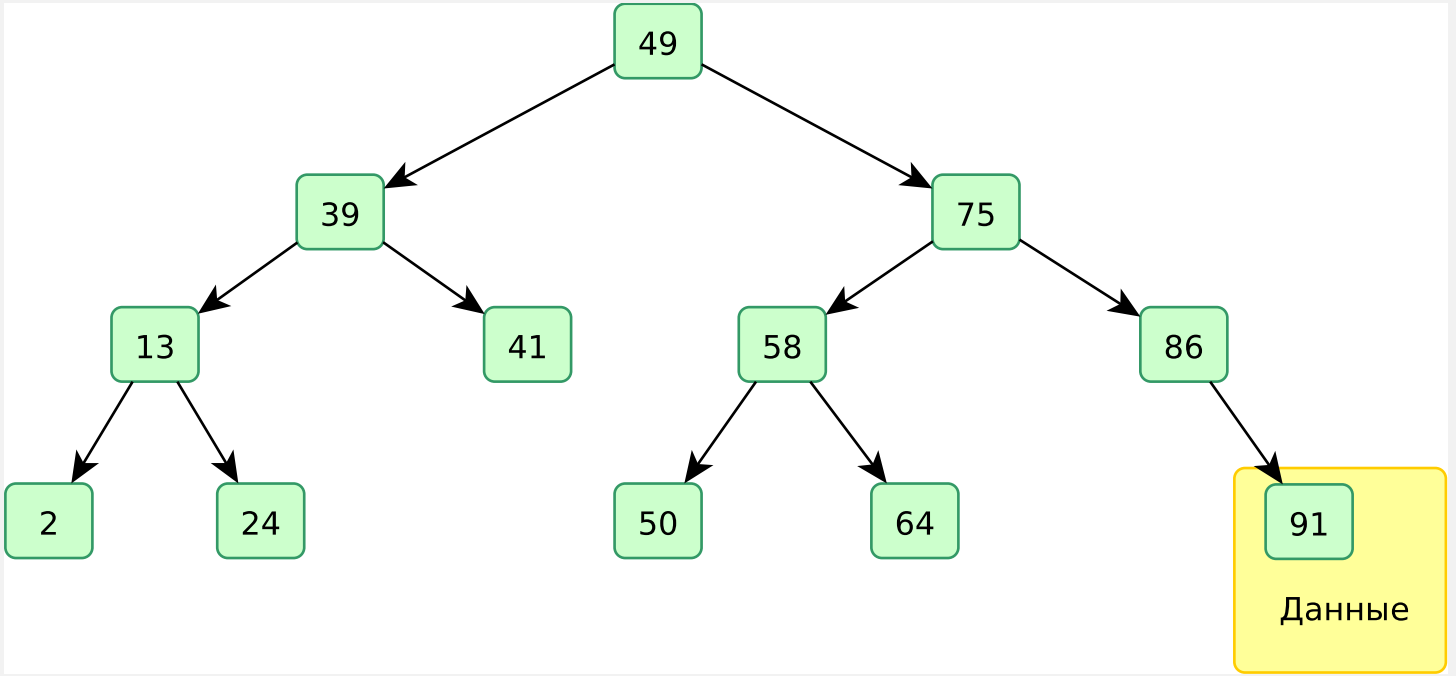
Использование struct list_head

```
struct list_head list;
struct user_object {
       int data;
       struct list_head entry;
} *uobj1, *uobj2;

INIT_LIST_HEAD(&list);

uobj1 = malloc(sizeof(*uobj1));
uobj1->data = 7857;

list_add(&uobj1->entry, &list);

uobj2 = list_first_entry(&list, struct user_object, entry);

list_for_each_entry(uobj2, &list, entry) {
       printf("%d\n", uobj2->data);
}
```

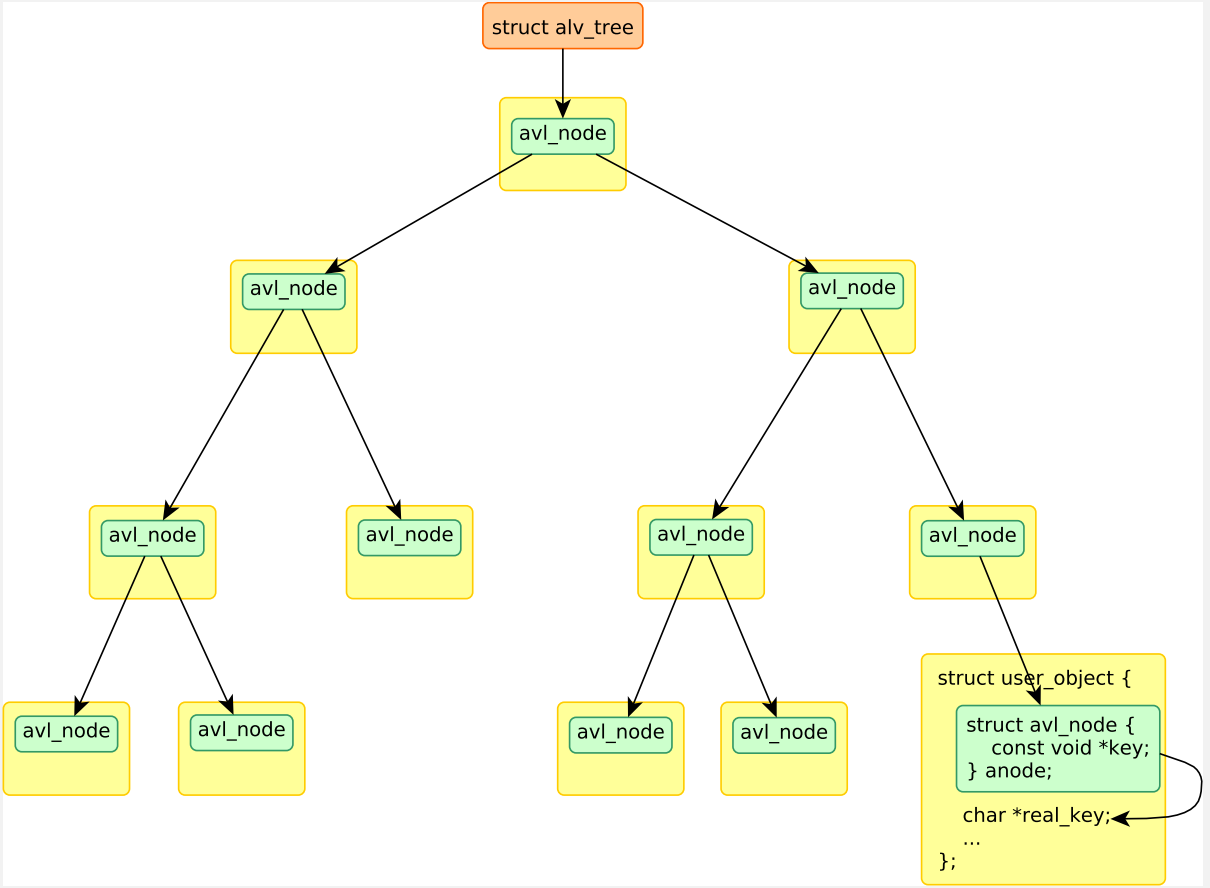
Поиск элемента по ключу



Дерево поиска



АВЛ-дерево


struct alv_tree
avl_node
avl_node
avl_node
avl_node
avl_node
avl_node
avl_node
avl_node
avl_node
avl_node
struct user_object {
struct avl_node {
const void *key;
} anode;
char *real_key;
...
};

Использование struct avl_tree и struct avl_node

```
struct avl_tree dict;
struct user_object {
       int data;
       char *key;
       struct avl_node anode;
} *uobj1, *uobj2;

avl_init(&dict, avl_strcmp, 0, NULL);

uobj1 = malloc(sizeof(*uobj1));
uobj1->data = 7857;
uobj1->key = strdup("some_key");
uobj1->anode.key = uobj1->key;

avl_insert(&dict, &uobj1->anode);

uobj2 = avl_find_element(&dict, "another_key", uobj2, anode);

avl_for_each_element(&dict, uobj2, anode) {
       printf("%s\n", uobj2->key);
}
```

Удаление элементов во время перебора

```
struct list_head list;
struct user_object {
       int data;
       struct list_head entry;
} *uobj, *next_uobj;

/* Неправильно! */
list_for_each_entry(uobj, &list, entry) {
       list_del(&uobj->entry);
       free(uobj);
       /* Дальше произойдет Segmentation fault, потому что указатель на
       следующий элемент должен получаться из текущего эелемента, который уже
       удален. */
}

/* Правильно */
list_for_each_entry_safe(uobj, next_uobj, &list, entry) {
       list_del(&uobj->entry);
       free(uobj);
}
```

Упражнения

- Разработать приложение, создающее список на основе `struct list_head` и добавляющее в него строковые элементы "aaa", "bbb", ..., "zzz". Приложение должно вывести в терминал элемент списка под номером n, где n - значение первого аргумента командной строки. После вывода приложение должно удалить все элементы списка.

- Разработать приложение, создающее словарь на основе `struct alv_tree` и добавляющее в него строковые элементы "aaa", "bbb", ..., "zzz" с ключами 'a', 'b', ..., 'z' . Приложение должно вывести в терминал элемент словаря с ключом k, где k - значение первого аргумента командной строки. После вывода приложение должно удалить все элементы словаря.