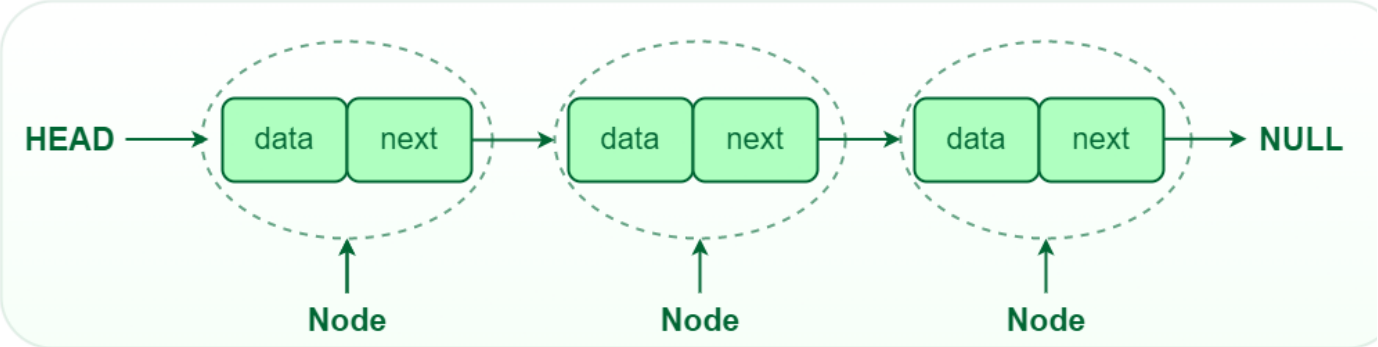
Linked List

Будет показана реализация на языке программирования **C**.

Что такое связный список

Связный список - это структура данных, представляющая собой соединенные указателями элементы. Каждый такой элемент должен содержать элемент (данные) и указатель на соседние элементы.



Бывает два вида связных списков - **односторонние** и **двусторонние**. Односторонние содержат только **head** - указатель на следующий элемент, а двусторонние - помимо **head** содержат и **tail** - указатель на предыдущий элемент.

Асимптотические сложности операций

- `insert` - выполняется за время **O(n)**, поскольку в худшем случае нам необходимо пройти по всем элементам списка, если знаем предыдущую ноду, то за **O(1)** (перецепляем указатели)
- `delete` - выполняется за время `O(n)`, поскольку может быть так, что придется удалить элемент в самом конце, если знаем предыдущую ноду, то за **O(1)** (перецепляем указатели)
- `search()` - поиск в связном списке занимает **O(n)**.

Реализация связных списков на C

Инициализация структуры и применение

```c
typedef struct node
{
    int value;         // элемент
    struct node *next; // указатель на следующий
} node_t;
```

```c
int main()
{
        // инициализируем первый элемент
    node_t *head = malloc(sizeof(node_t));

    head->value = 1;
    head->next = NULL;
    // создали второй элемент и присвоили его указатель в head->next
    node_t *elem2 = malloc(sizeof(node_t));
    head->next = elem2;
    elem2->value = 2;
    elem2->next = NULL;
    print_linked_list(head);
}
```

Вывод связного списка на печать

Вывод элементов происходит при помощи бесконечного цикла `while`. Так намного удобнее прекратить вывод в тот момент, когда следующего элемента попросту нет!

```c
void print_linked_list(node_t *head)
{
    node_t *current = head;
    while (current!= NULL)
    {
        printf("Current value: %d\n", current->value);
        current = current->next;
    }
}
```

Операции над элементами связного списка

Добавление элемента в список

Добавление элемента в список происходит на сложность **O(n)** поскольку нам необходимо пробежаться по каждому элементу списка и так дойти до последнего.

```c
// добавление value в конец списка

void push_into_end(node_t *head, int value)
{
    node_t *current = head;
    while (current->next != NULL)
    {
        current = current->next;
    }

    // после того как нашли последний элемент
    current->next = malloc(sizeof(node_t));
    current->next->value = value;
```

```
    current->next->next = NULL;
}

// добавление элемента в начала списка
void push_into_beginning(node_t** head, int value){
    node_t* new_node;
    new_node = malloc(sizeof(node_t));

    new_node->next = *head;
    new_node->value = value;
    *head = new_node;
}
```

Удаление элемента из списка

```
// удаление первого элемента из списка

int pop_first_elem(node_t** head)
{
    int retval = -1;
    node_t* next_node = NULL;

    if (*head == NULL){
        return -1;
    }
    next_node = (*head)->next;
    retval = (*head)->value;
    free(*head);
    *head = next_node;

    return retval;
}



// удаление последнего элемента из списка

int pop_last_elem(node_t* head)
{
    int retval = 0;
    if (head->next == NULL){
        retval = head->value;
        free(head);
        return retval;
    }

    // находим вторую с конца ноду,
    // чтобы потом легко было перевязать адреса
    node_t* current = head;
    while (current->next->next != NULL){
        current = current->next;
    }
```

```
    retval = current->next->value;
    free(current->next);
    current->next = NULL;
    return retval;
}
```

Удаление/добавление элемента в произвольное место

```
// удаление/добавление в произвольное место

int pop_by_index(node_t ** head, int n){
    int retval = -1;
    node_t*current = *head;
    node_t* temp_node = NULL;

    if (n == 0){
        return pop_first_elem(head);
    }

    for (int i = 0; i < n - 1; i++){
        if (current->next == NULL){
            return -1;
        }
        current = current->next;
    }

    if (current->next == NULL){
        return -1;
    }

    temp_node = current->next;
    retval = temp_node->value;
    current->next = temp_node->next;
    free(temp_node);
    return retval;
}
// добавление элемента по индексу
void push_by_index(node_t ** head, int n, int value){
    if (n == 0){
        *head = (*head)->next;
    }
    node_t* current = *head;
    for (int i = 0; i < n - 1; i++){
        if (current->next == NULL){
            return;
        }
        current = current->next;
    }

    if (current->next == NULL){
        return;
    }
```

```
    node_t* new_node = malloc(sizeof(node_t));
    new_node->value = value;
    new_node->next = current->next;
    current->next = new_node;
}
```

Как использовать

Код функции `main`, которая работает со связными списками.

```
int main()
{
    node_t *head = malloc(sizeof(node_t));

    if (head == NULL)
    {
        return 1;
    }

    head->value = 1;
    head->next = NULL;
    push_into_end(head, 2);
    push_into_end(head, 3);
    push_into_end(head, 5);
    push_into_end(head, 6);
    push_into_end(head, 7);
    push_by_index(&head, 3, 4);
    int elem = pop_by_index(&head, 4);
    printf("Pop elem: %d\n", elem);
    print_linked_list(head);
}
```