Списки и их применение

Т.И. Комаров

НИЯУ МИФИ

2023

Уровни представления информации

- Интуитивный
- Логический (абстрактный)
- Конкретный (физический)

Логические структуры данных

- Массивы
- Строки
- Стеки, деки, очереди
- Таблицы
- Деревья
- Графы

Важно

Для каждой логической структуры данных определён свой набор операций

Стек (логическая структура данных)

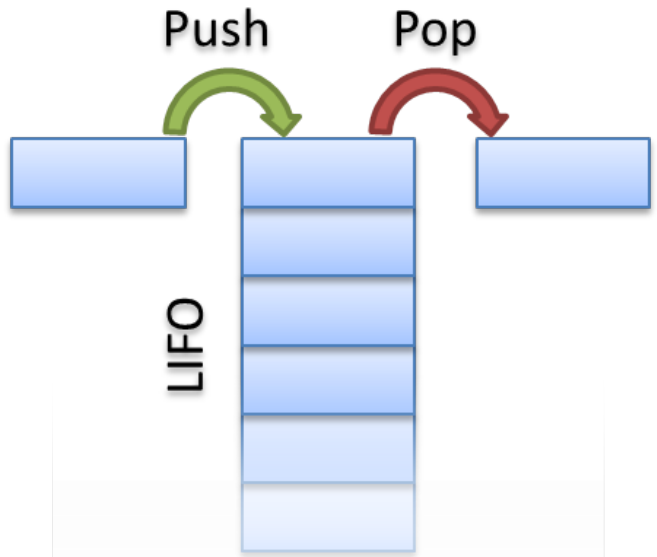
Определение

Стек (англ. *stack* — стопка) — логическая структура данных, представляющая собой упорядоченный набор элементов, организованный по принципу **LIFO** (англ. *last in — first out*, «последним пришёл — первым вышел»)

Добавление новых и удаление существующих элементов из стека осуществляется с одного конца, называемого **вершиной**

Набор операций:

- **push** — добавление элемента
- **pop** — удаление элемента
- **peek** — чтение элемента, находящегося на вершине



Очередь (логическая структура данных)

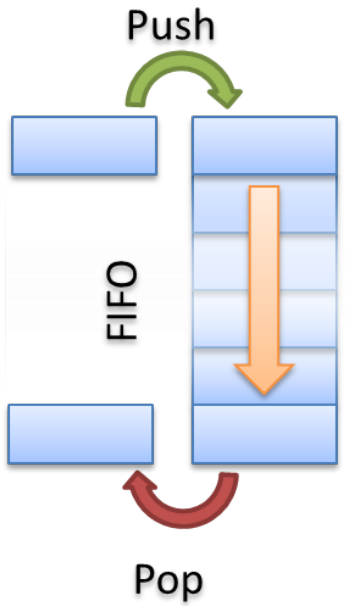
Определение

Очередь (англ. *queue*) — логическая структура данных, представляющая собой упорядоченный набор элементов, организованный по принципу **FIFO** (англ. *first in — first out*, «первым пришёл — первым вышел»)

Добавление элементов осуществляется в **конец** очереди, а удаление — из **начала**

Набор операций:

- **put** — добавление элемента
- **get** — удаление элемента
- **peek** — чтение элемента, находящегося в начале

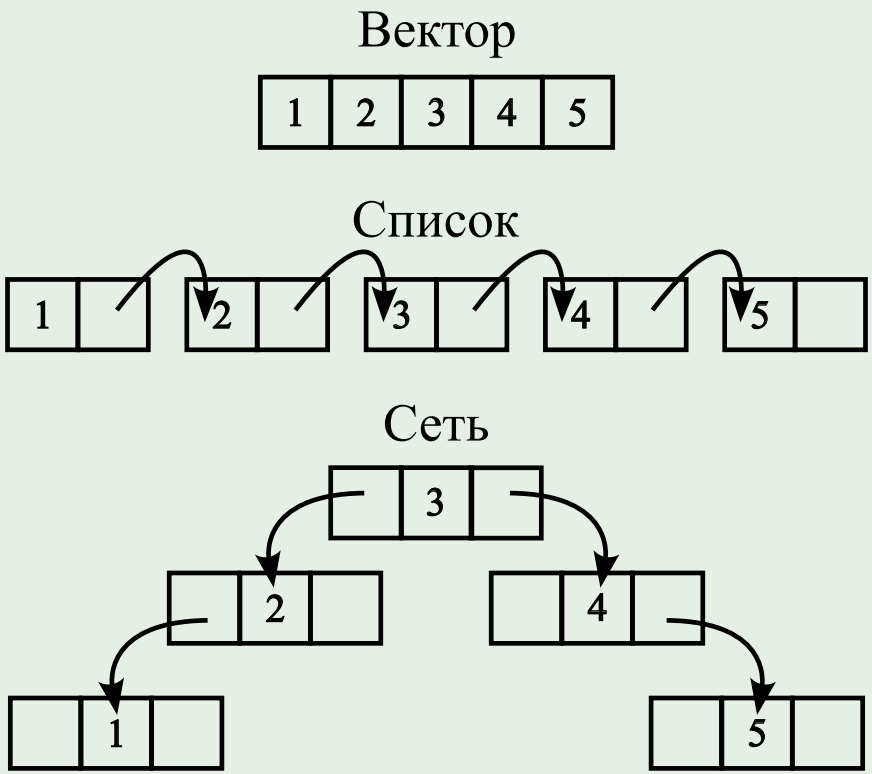


Физические структуры данных

Представление данных в памяти компьютера:

- Вектор
- Список
- Сеть (не путать с компьютерными сетями!)

Примеры



Классификация списков

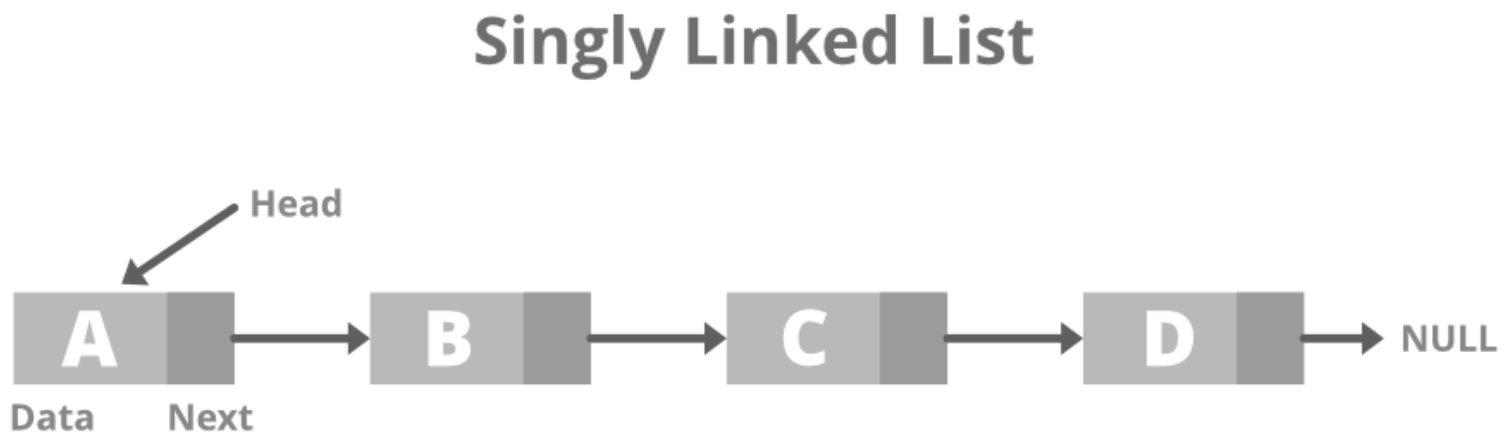
По количеству полей указателей:

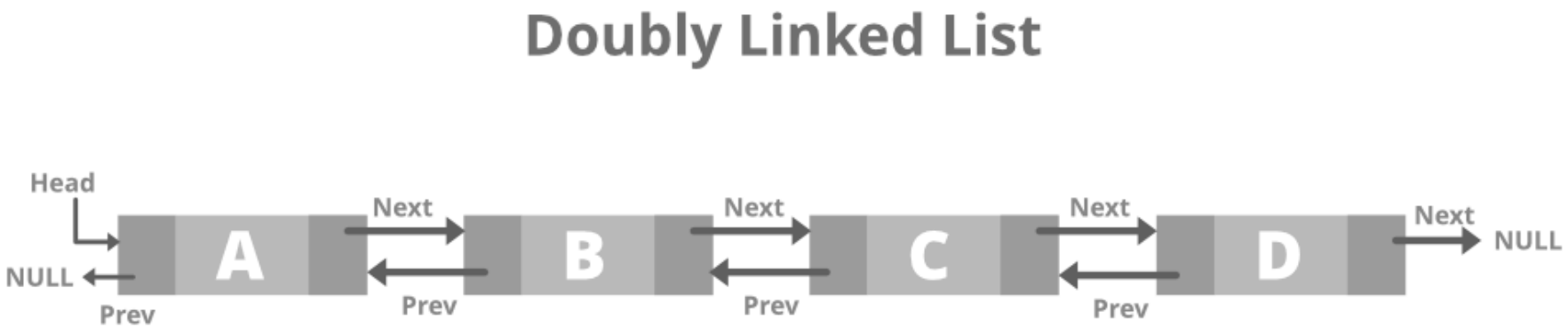
- Односвязный (однонаправленный)
- Двусвязный (двунаправленный)

По способу связи элементов:

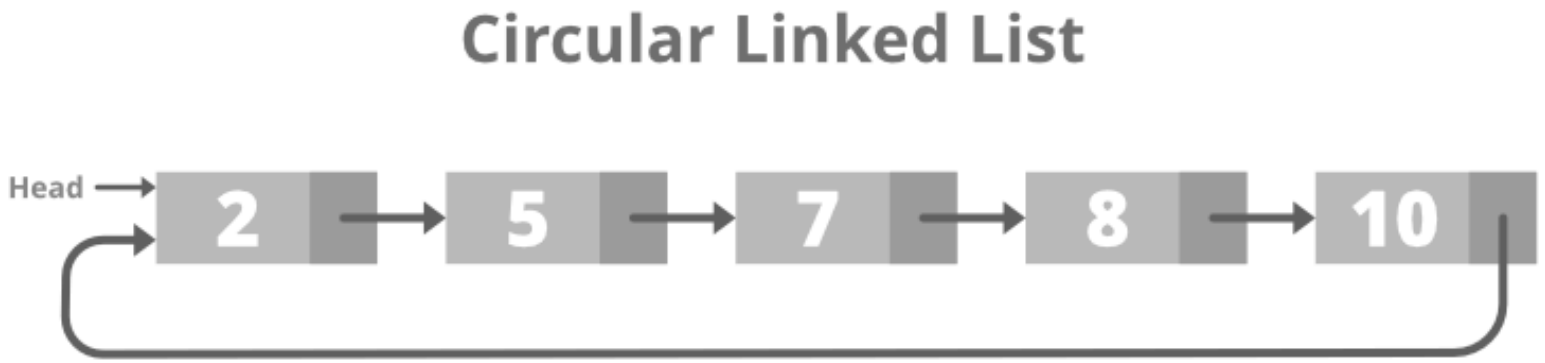
- Линейный
- Циклический

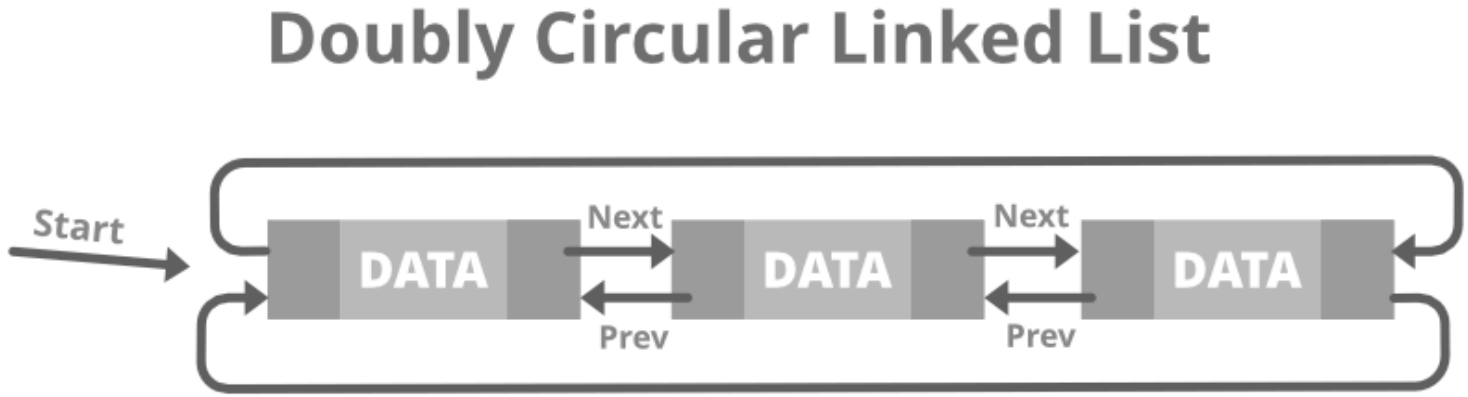
Линейный односвязный и линейный двусвязный списки





Циклический односвязный и циклический двусвязный списки





Пример реализации стека на основе списка I

Примеры

```c
typedef struct Item {
    int data;
    struct Item *next;
} Item;

typedef struct Stack {
    Item *head;
} Stack;

Stack *stack_new();
void stack_delete(Stack *stack);
int stack_push(Stack *stack, int data);
int stack_pop(Stack *stack, int *output);
void stack_print(const Stack *stack);
```

Пример реализации стека на основе списка II

Примеры

```
Stack *stack_new() {
    return (Stack *) calloc(1, sizeof(Stack));
}

void stack_delete(Stack *stack) {
    Item *ptr = stack->head, *ptr_prev;
    while (ptr) {
        ptr_prev = ptr;
        ptr = ptr->next;
        free(ptr_prev);
    }
    free(stack);
}
```

Пример реализации стека на основе списка III

Примеры

```
int stack_push(Stack *stack, int data) {
    Item *new = (Item *) malloc(sizeof(Item));
    if (!new) {
        return 1;
    }
    new->data = data;
    new->next = stack->head;
    stack->head = new;
    return 0;
}
```

Пример реализации стека на основе списка IV

Примеры

```
int stack_pop(Stack *stack, int *output) {
    if (!stack->head) {
        return 1;
    }
    *output = stack->head->data;
    Item *head = stack->head;
    stack->head = head->next;
    free(head);
    return 0;
}
```

Пример реализации стека на основе списка V

Примеры

```
void stack_print(const Stack *stack) {
    Item *ptr = stack->head;
    while (ptr) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

Пример реализации очереди на основе списка I

Примеры

```
typedef struct Item {
    int data;
    struct Item *next;
} Item;

typedef struct Queue {
    Item *head;
    Item *tail;
} Queue;

Queue *queue_new();
void queue_delete(Queue *queue);
int queue_put(Queue *queue, int data);
int queue_get(Queue *queue, int *output);
void queue_print(const Queue *queue);
```

Пример реализации очереди на основе списка II

Примеры

```
Queue *queue_new() {
    return (Queue *) calloc(1, sizeof(Queue));
}

void queue_delete(Queue *queue) {
    Item *ptr = queue->head, *ptr_prev;
    while (ptr) {
        ptr_prev = ptr;
        ptr = ptr->next;
        free(ptr_prev);
    }
    free(queue);
}
```

Пример реализации очереди на основе списка III

Примеры

```
int queue_put(Queue *queue, int data) {
    Item *new = (Item *) malloc(sizeof(Item));
    if (!new) { return 1; }
    new->data = data;
    new->next = NULL;
    if (!queue->head) {
        queue->head = new;
        queue->tail = new;
    } else {
        queue->tail->next = new;
        queue->tail = new;
    }
    return 0;
}
```

Пример реализации очереди на основе списка IV

Примеры

```
int queue_get(Queue *queue, int *output) {
    if (!queue->head) {
        return 1;
    }
    *output = queue->head->data;
    if (queue->head == queue->tail) {
        queue->tail = NULL;
    }
    Item *head = queue->head;
    queue->head = head->next;
    free(head);
    return 0;
}
```

Пример реализации очереди на основе списка V

Примеры

```
void queue_print(const Queue *queue) {
    Item *ptr = queue->head;
    while (ptr) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

Пример реализации упорядоченного списка I

Примеры

```
typedef struct Item {
    int data;
    struct Item *next;
} Item;
typedef struct List {
    Item *head;
    Item *tail;
} List;

List *list_new();
void list_delete(List *list);
void list_print(const List *list);
int list_push_back(List *list, int data);
int list_insert(List *list, int data);
int list_remove(List *list, int data);
```

Пример реализации упорядоченного списка II

Примеры

```
List *list_new() {
    return (List *) calloc(1, sizeof(List));
}

void list_delete(List *list) {
    Item *ptr = list->head, *ptr_prev;
    while (ptr) {
        ptr_prev = ptr;
        ptr = ptr->next;
        free(ptr_prev);
    }
    free(list);
}
```

Пример реализации упорядоченного списка III

Примеры

```
int list_push_back(List *list, int data) {
    Item *ptr = (Item *) malloc(sizeof(Item));
    if (!ptr) { return 1; }
    ptr->data = data;
    ptr->next = NULL;
    if (!list->head) {
        list->head = ptr;
        list->tail = ptr;
    } else {
        list->tail->next = ptr;
        list->tail = ptr;
    }
    return 0;
}
```

Пример реализации упорядоченного списка IV

Примеры

```
int list_insert(List *list, int data) {
    Item *ptr = list->head, *ptr_prev = NULL;
    while (ptr && (ptr->data < data)) {
        ptr_prev = ptr; ptr = ptr->next;
    }
    Item *new = (Item *) malloc(sizeof(Item));
    if (!new) { return 1; }
    new->data = data; new->next = ptr;
    if (ptr_prev) { ptr_prev->next = new; }
    else { list->head = new; }
    if (!ptr) { list->tail = new; }
    return 0;
}
```

Пример реализации упорядоченного списка V

Примеры

```
int list_remove(List *list, int data) {
    Item *ptr = list->head, *ptr_prev = NULL;
    while (ptr && ptr->data != data) {
        ptr_prev = ptr; ptr = ptr->next;
    }
    if (!ptr) { return 1; }
    if (ptr == list->head) {
        list->head = ptr->next;
    }
    if (ptr == list->tail) {
        list->tail = ptr_prev;
    }
    if (ptr_prev) { ptr_prev->next = ptr->next; }
    free(ptr); return 0;
}
```