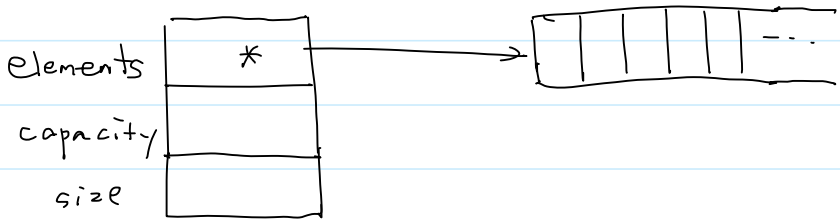
动态数组

2024年5月2日 10:03

一、模型

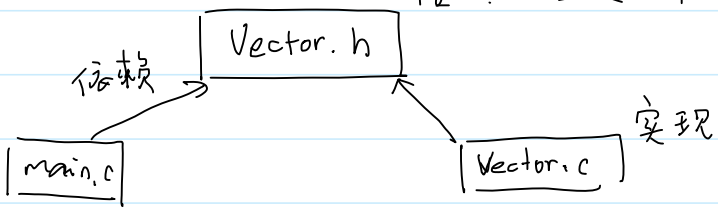


依赖接口，不要依赖具体的实现！

↓

*.h，Interface，抽象类

接口：类型定义和API声明，（稳定）



（变化）

realloc

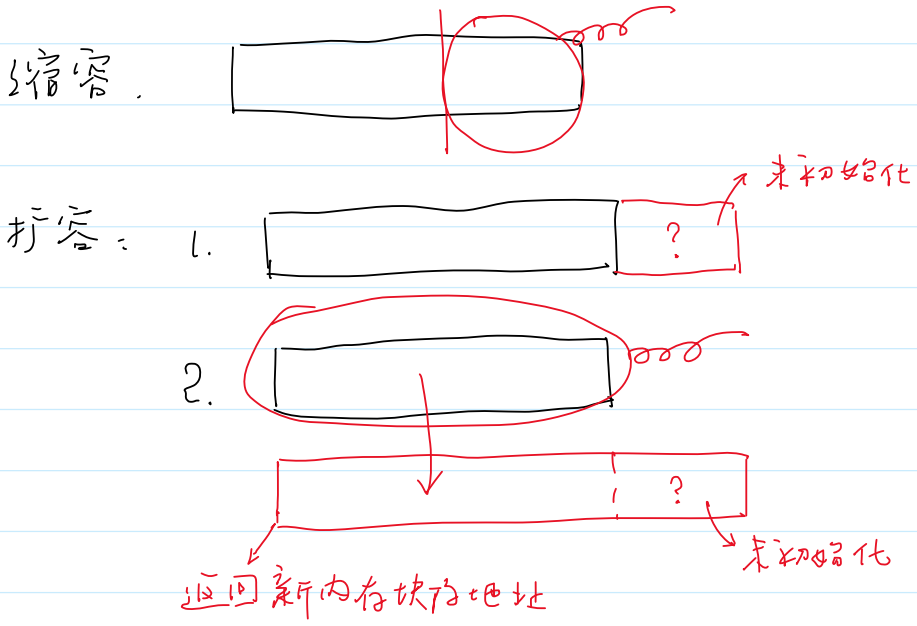
2024年5月2日 11:10

指向先前分配的内存块

```
void* realloc(void *ptr, size_t size)
```

新内存块的大小。

调整先前分配内存块的大小。如果重新分配内存大小成功，返回指向新内存块的指针，否则返回空指针。



大端法和小端法

2024年5月2日 11:25

	D0	01	D2	D3
0x00F4EBD0	00	00	00	00
0x00F4EBD4	01	00	00	00

→小端法

D4

值 —编码→ 二制制表示 —大端法/小端法→ 内存布局

1 —补码→ 0x 00 00 00 01

⇓

大端法：低有效位放在高地址

低 ——→ 高

00 00 00 01

小端法：低有效位放在低地址

低 ——→ 高

01 00 00 00

小端法：Intel 或 Intel 兼容

大端法：网络 Sparc

free

2024年5月2日 11:37

垃圾回收器：减轻程序员负担

不确定因素 → Stop the world → 不适合写实时系统

↓

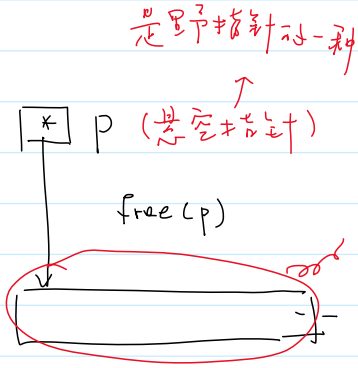
以为总在某个确定时间内完成某个任务

没有垃圾回收器：

C：free

C++：析构函数 RAII、智能指针

Rust：所有权机制




```c
int* p = malloc(100 * sizeof(int));
free(p);    // free(p) 不会改变p的值，只是将p指向的内存空间释放了
```

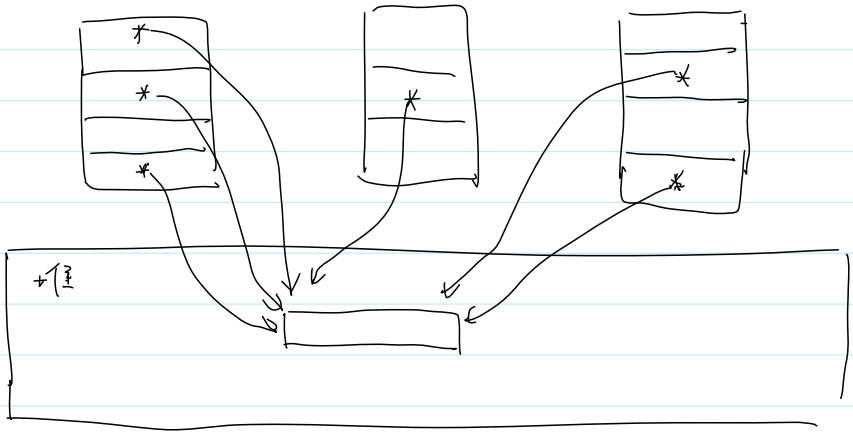
问题：

(1) double free

(2) use after free

⇓

结论：当堆上数据不再使用时，应该有且只释放一次！

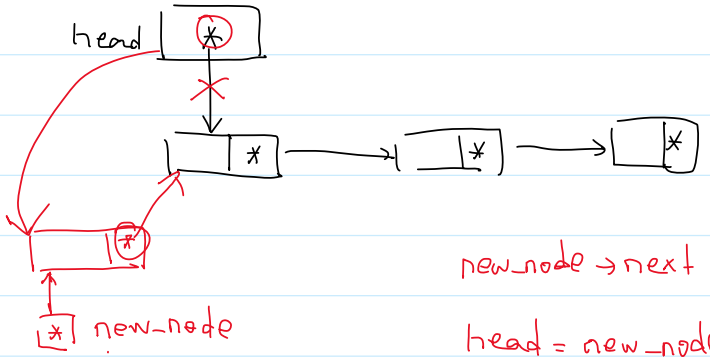


动态内存分配是链式结构的基础

2024年5月2日 14:25

基本步骤：① 定义结点

② 动态链表 → 动态内存分配



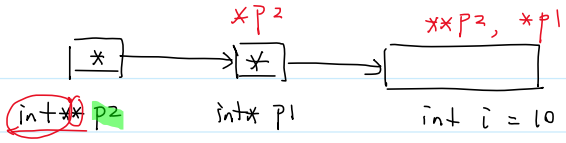
new_node->next = head;

head = new_node;

⇒ p = q 指针的赋值

二级指针

2024年5月2日 14:55

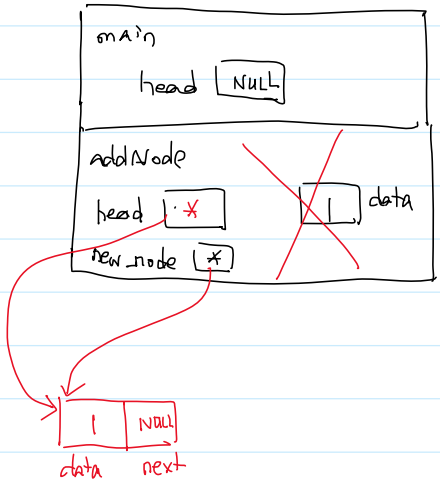


i: 访问内存一次

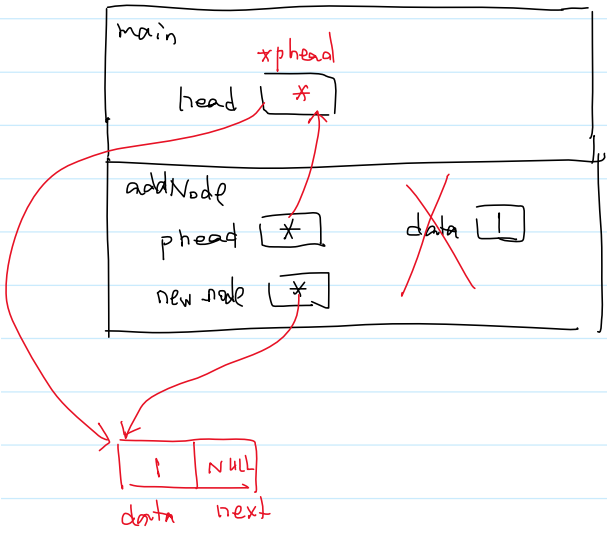
*p1: 两次

**p2: 三次

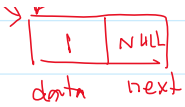
```c
int main(void) {
    Node* head = NULL;  // 空链表
    addNode(head, 1);
    addNode(head, 2);
    addNode(head, 3);
    addNode(head, 4);    // 4 --> 3 --> 2 --> 1

    return 0;
}
void addNode(Node* head, int data) {
    // 1. 创建结点
    Node* new_node = malloc(sizeof(Node));
    if (!new_node) {
        printf("Error: malloc failed in addNode.\n");
        exit(1);
    }
    // 2. 初始化结点
    new_node->data = data;
    new_node->next = head;

    head = new_node;
}
```




```c
int main(void) {
    Node* head = NULL;  // 空链表
    addNode(&head, 1);
    addNode(&head, 2);
    addNode(&head, 3);
    addNode(&head, 4);  // 4 --> 3 --> 2 --> 1

    return 0;
}
void addNode(Node** phead, int data) {
    // 1. 创建结点
    Node* new_node = malloc(sizeof(Node));
    if (!new_node) {
        printf("Error: malloc failed in addNode.\n");
        exit(1);
    }
```



```c
    if (!new_node) {
        printf("Error: malloc failed in addNode.\n");
        exit(1);
    }
    // 2. 初始化结点
    new_node->data = data;
    new_node->next = *phead;

    *phead = new_node;
}
```



总结：传一级指针，还是传二级指针？

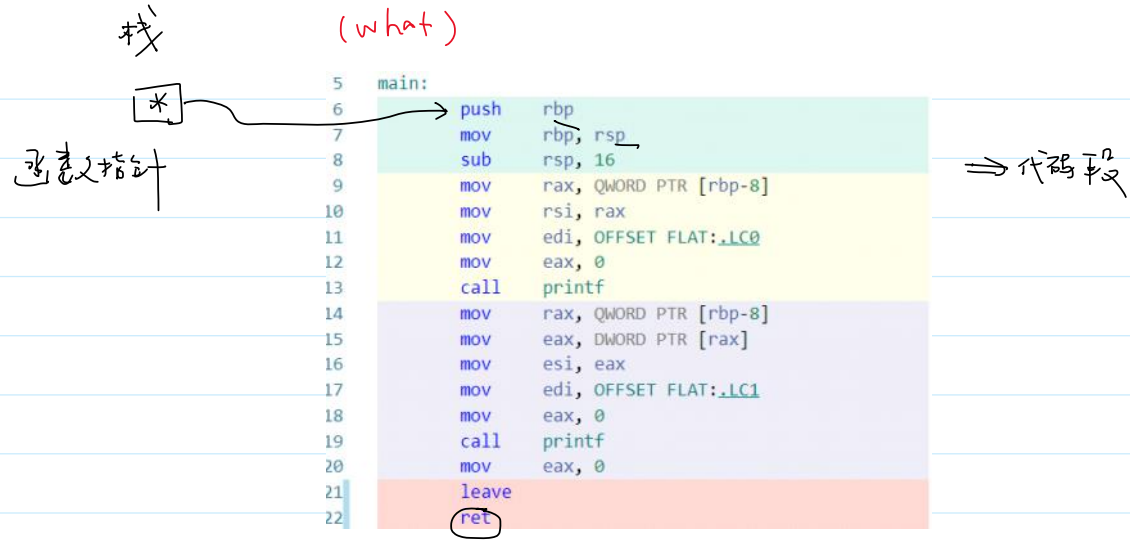
想修改哪个变量，就传那个变量的地址。

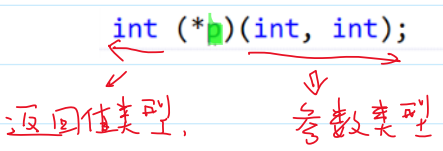
A. 想修改指针指向时，传一级指针

B. 想修改指针的指向（值），传二级指针。

函数指针

2024年5月2日 15:16






```
int (*p1)(int, int);     // 声明指针变量
int* p2(int, int);       // 声明函数
```

qsort

2024年5月2日 15:59

对任意一个数组进行排序

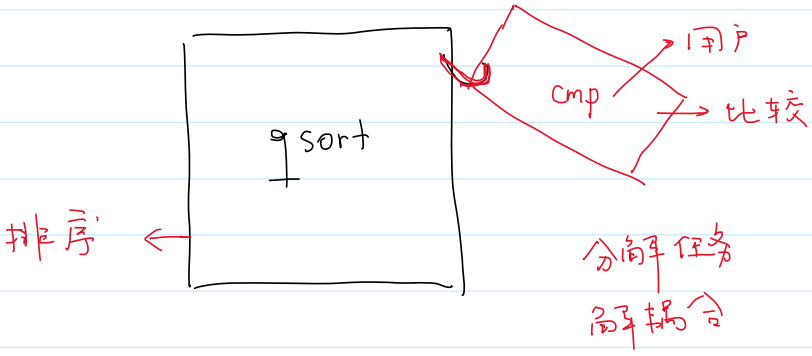
`qsort` 函数的原型如下：

```
void qsort(void* base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
```

- `base` 指向数组中要排序的第一个元素（一般是数组的第一个元素）。
- `nmemb` 是要排序元素的数量（可以小于或等于数组中元素的个数）。
- `size` 表示数组元素的大小。
- `compar` 是比较函数，如果第一个参数比第二个参数小则返回负数，相等则返回零，第一个参数大于第二个参数则返回正数。

```
// 作用：（Why）
// 1. 函数式编程（传递函数，返回函数），  函数指针支持函数式编程
//     分解任务，解耦合
//
// 2. 编写非常通用的函数（功能非常强大的函数）
// qsort(void* base, size_t n, size_t size, int (*compare)(const void*, const void*));
```

钩子函数



数据结构和算法 ⟶ 逻辑比较强。

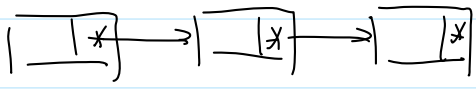
2024年5月2日 16:31

网站推荐：数据结构与算法可视化

视频推荐：王道 数据结构和算法

⇒ 预习：模型 + 时间复杂度。

1、模型



2. API

增；

删；

查；

遍历。每一个元素有且只经过一次

3. 实现 （~~~~）

基础、动态数组，链表

栈、队列、哈希表、位图，二叉树。

算法：排序、二分查找

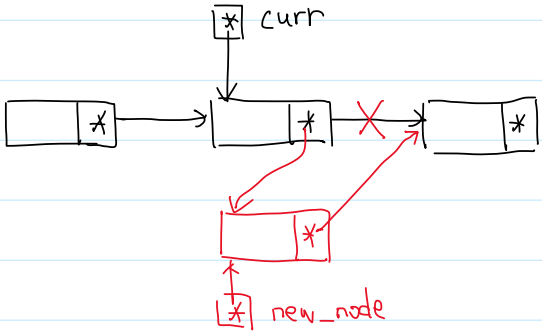
单链表

2024年5月2日 17:04

单向链表

基本操作

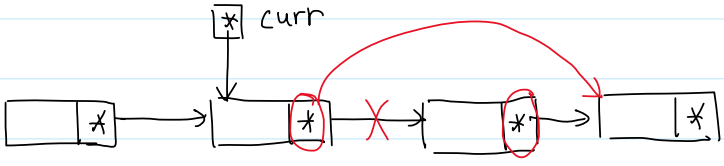
增：在某个结点后面添加 O(1)



new_node->next = curr->next;

curr->next = new_node;

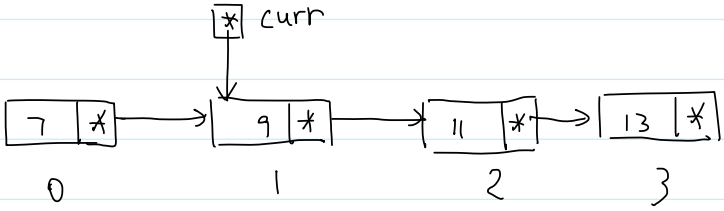
删：在某个结点后面删除 O(1)



curr->next = curr->next->next;

查

a. 根据索引查找结点 O(n)

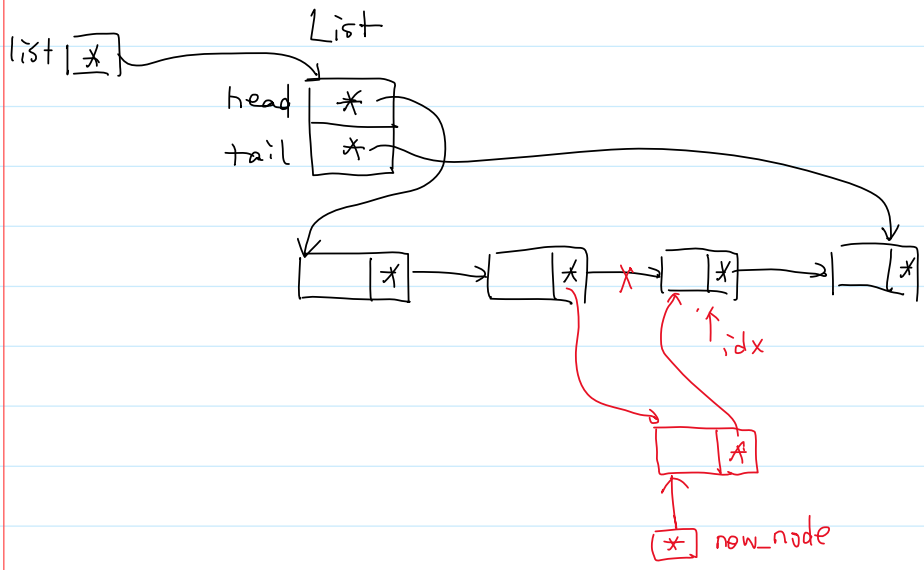


b. 查找与特定值相等的结点

(1) 元素大小无序 O(n)

(2) 元素大小有序 O(n)

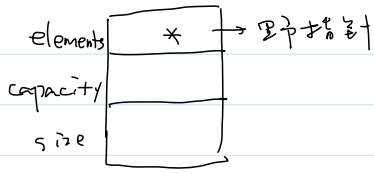
遍历：正向遍历



晚自习答疑

2024年5月2日 21:34

① `Vector v;`

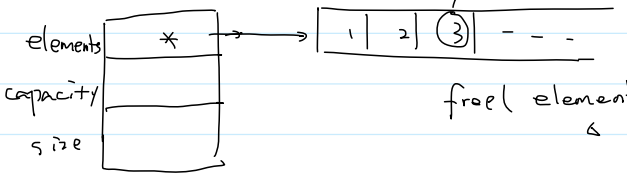


realloc (ptr, size)

↓

扩容原先分配的内存块

②



free(elements + list->size -1);

---

free的实现：

free (ptr)