栈、堆和队列

1 栈

1.1 定义

栈（stack）是一种只能从表的一端存取数据且遵循 "先进后出" 原则的线性存储结构。先进后出即最先进栈的元素最后出栈。通常，栈的开口端被称为**栈顶**，封口端被称为**栈底**，**栈顶元素**指的就是距离栈顶最近的元素。栈是一种 "特殊" 的线性存储结构，可用顺序存储结构（顺序栈）或链式存储结构（链栈）实现。

1.2 进栈和出栈

进栈（入栈或压栈）：向栈中添加元素。

出栈（弹栈）：从栈中提取出指定元素。

1.3 顺序表实现

在顺序表中设定一个实时指向栈顶元素的变量（top），top 初始值为 -1，表示栈中没有存储任何数据元素，及栈是"空栈"。一旦有数据元素进栈，则 top 就做 +1 操作；反之，如果数据元素出栈，top 就做 -1 操作。

```
typedef struct Stack {
    int *a;
    int top;
    int size;
}stack;
/* 入栈 */
stack push(stack s, int data) {
    s.a[++top] = data;
    return s;
}
/* 出栈 */
stack pop(stack s) {
    if(top == -1) return s;
```

```
    printf("栈顶元素: %d\n", s.a[top--]);
    return s;
}
```

1.4 链表实现

通常将链表的头部作为栈顶，尾部作为栈底，可以避免出栈和入栈时做大量遍历链表的操作。

无头节点:

```
typedef struct stack{
    int data;
    struct stack *next;
}stack;
/* 入栈 */
stack *push(stack *s, int data){
    stack *tmp = (stack *)malloc(sizeof(stack));
    tmp->data = data;
    tmp->next = s;
    s = tmp;
    return s;
}
/* 出栈 */
stack *pop(stack *s){
    if(!s) return NULL;
    stack *tmp = (stack *)malloc(sizeof(stack));
    tmp = s;
    s = s->next;
    free(tmp);
    return s;
}
```

有头结点:

```
void push(stack *s, int data){
    stack *tmp = (stack *)malloc(sizeof(stack));
    if(!tmp) return;
    tmp->data = data;
    tmp->next = s->next;
    s->next = tmp;
}
void pop(stack *s){
    if(!s->next) return;  //空栈
    stack *tmp = s->next;
    s->next = tmp->next;
    free(tmp);
}
```

2 堆

2.1 定义

堆（heap）又称为优先队列，具有插入和删除最小值两种操作。

2.2 二叉堆

堆是一棵完全二叉树。堆具有堆序性，即在一个堆中，每个节点的值小于其父节点的值，即根节点具有最小值。

```c
/* 数组实现 */
typedef struct Heap{
    int capacity;
    int size;
    int *a;
}*Heap;
/* 插入元素 */
void insert(int data, Heap h){
    int i;
    for(i = ++h->size; h->a[i / 2] > data; i /= 2)
        h->a[i] = h->a[i / 2];
    h->a[i] = data;
}
/* 删除最小值 */
int deleteMin(Heap h){
    int i, child;
    int min, last;
    min = h->a[1];
    last = h->a[h->size--];
    for(i = 1; i * 2 <= h->size; i = child){
        child = i * 2;
        if(child != h->size && h->a[child + 1] < h->a[child])
            child++;
        if(last > h->a[child])
            h->a[i] = h->a[child];
        else
            break;
    }
    h->a[i] = last;
    return min;
}
```

3 队列

3.1 定义

队列的两端都**开口**，要求数据只能从一端进，从另一端出。通常，称进数据的一端为 "队尾"，出数据的一端为 "队头"，数据元素进队列的过程称为**入队**，出队列的过程称为**出队**。队列中数据的进出要遵循**先进先出**的原则，即最先进队列的数据元素，同样要最先出队列。

3.2 实现

3.3 顺序队列

即采用顺序表模拟实现的队列结构。

实现：定义两个指针（top 和 rear）分别用于指向顺序队列中的队头元素和队尾元素。由于顺序队列初始状态没有存储任何元素，因此 top 指针和 rear 指针重合。

入队：将要入队的数据元素存储在指针 rear 指向的数组位置，然后 rear+1。

出队：当需要队头元素出队时，仅需做 top+1 操作。

循环队列实现：

```c
#define MAXSIZE 5
typedef struct{
    int *data;
    int top;
    int rear;
}Queue;
/* 初始化队列 */
void initQueue(Queue *q){
    //队首等于队尾表示队列为空
    q->data = (int *)malloc(sizeof(int * MAXSIZE));
    if(!data) return;
    q->front = q->rear = 0;
}
/* 入队 */
void enQueue(queue *q, int data){
    //保留一个元素空间，表示队列已满
    if((q->rear+1)%MAXSIZE == q->front) return;
    q->data[q->rear] = data;
    q->rear = (q->rear+1) % MAXSIZE;
}
/* 出队 */
void deQueue(queue *q){
    //队列为空
    if(q->front == q->rear) return;
    q->front = (q->front + 1) % MAXSIZE;
}
```

3.4 链式队列

创建一个带有头节点的链表实现链式队列会更简单。

```c
/* 队列结点 */
typedef struct Node{
    int data;
    struct queue next;
}Node;
/* 队列 */
typedef struct{
    Node *front;
    Node *rear;
}Queue;
/* 初始化队列 */
void initQueue(Queue *q){
    //初始化队列队首和队尾地址相同，故第一个元素入队时，队首的next也会指向首元素
    q->front = q->rear = (Node *)malloc(sizeof(Node));
    if(!q->front) return;
    q->front->next = NULL;
}
/* 入队 */
void enQueue(Queue *q, int data){
    Node *tmp = (Node *)malloc(sizeof(Node));
    if(!tmp) return;
    tmp->data = data;
    tmp->next = NULL;
    q->rear->next = tmp;
    q->rear = tmp;
}
```

```c
/* 出队 */
void deQueue(Queue *q){
    if(q->front == q->rear) return;
    Node *tmp = q->front->next;
    //出队结点为队列中的最后一个结点时，使队尾指向队首，表示队列为空
    if(q->front->next == q->rear)
        q->rear = q->front;
    q->front->next = tmp->next;
    free(tmp);
    tmp = NULL;
}
```