常见数据结构

01 队列

队列中的数据存取方式是"先进先出"，只能往 **队尾插入数据** 、 **从队头移出数据** 。

下图是队列的原理，队头 $head$ 指向队列中的第一个元素 $a1$ ，队尾 $tail$ 指向队尾最后一个元素 an 。元素只能从队头方向出去，元素只能从队尾进入队列。

C/C++手写队列

队列的代码很容易实现。如果使用环境简单，最简单的手写队列代码用数组实现。

```
const int N = 10000;      //定义队列容量，确保够用
int que[N];               //队列，用数组模拟
int head = 0;             //head始终指向队头。que[head]是队头。开始时队列为空，head = 0
int tail = -1;            //tail始终指向队尾。que[tail]是队尾。开始时队列为空，tail = -1
                          //队列长度等于tail-head+1
head++;                   //弹出队头元素，让head指向新队头。注意保持head <= tail
que[head];                //读队头
que[++tail] = data;       //入队：先tail加1，然后数据data入队。注意tail必须小于N
```

> 如果进入队列的数据太多，使得 $tail$ 超过了 N ，数组 $que[N]$ 就会溢出，导致出错。

STL 队列queue

```
queue<Type> q;   // 定义队列，Type为数据类型，如int，float，char 等
push();          // 在队列尾部插入一个元素
front();         // 返回队首元素，但不会删除
pop();           // 删除队首元素
back();          // 返回队尾元素
size();          // 返回元素个数
empty();         // 检查队列是否为空
```

02 栈 Stack

栈 $stack$ 是比队列更简单的数据结构，它的特点是“ **先进后出** ”。

队列有两个口，一个入口和一个出口。而栈只有唯一的一个口，既从这个口进入，又从这个口出来。

C/C++手写栈

如果使用环境简单，最简单的手写栈代码用数组实现。

```
const int N = 300008;                    // 定义栈的大小
struct mystack{
    int a[N];                            // 存放栈元素，从a[0]开始
    int t = -1;                          // 栈顶位置，初始栈为空，置初值为-1
    void push(int data) {               // 把元素data送入栈
         a[++t] = data;
    }
    int top() {                          // 读栈顶元素，不弹出
        return a[t];
    }
    void pop() {                         // 弹出栈顶
        if(t > -1)
            t--;
    }
    int size() {                         // 栈内元素的数量
        return t + 1;
    }
    int empty() {                       // 若栈为空返回1
        return t == -1 ? 1 : 0;
    }
};
```

> 使用栈时要注意不能超过栈的空间。

STL 栈stack

```
stack<Type> s;  // 定义栈，Type为数据类型，如int，float，char 等
push(item);     // 把 item 放到栈顶
top();          // 返回栈顶的元素，但不会删除
pop();          // 删除栈顶的元素，但不会返回
size();         // 返回栈中元素的个数
empty();        // 检查栈是否为空，如果为空返回 true，否则返回 false
```

03 优先队列

优先队列 `std::priority_queue` 是一种 **堆**，一般为 **二叉堆。**

```
priority_queue<int> i;
// less是从大到小 : 大根堆
// greater是从小到大 : 小跟堆
priority_queue<int, vector<int>, less<int> > q;
priority_queue<int, vector<int>, greater<int> > q;
```

```
// 以下所有函数均为常数复杂度
top();      // 访问堆顶元素（此时优先队列不能为空）
empty();    // 询问容器是否为空
size();     // 查询容器中的元素数量

// 以下所有函数均为对数复杂度
push(x);    // 插入元素，并对底层容器排序
pop();      // 删除堆顶元素（此时优先队列不能为空）
```

> 拓展：`pair`

`pair` 是将2个数据组合成一组数据，`pair<type, type>`

> $pair$ 的实现是一个结构体，主要的两个成员变量是 $first, second$ 因为是使用 $struct$ 不是 $class$ ，所以可以直接使用 $pair$ 的成员变量。

功能：

1. **pair将一对值(T_1 和 T_2)组合成一个值**，
2. 这一对值可以具有不同的数据类型（ T_1 和 T_2 ）
3. 两个值可以分别**用 $pair$ 的两个公有函数 $first$ 和 $second$ 访问**

```
#define PII pair<int, int>
PII age(18, 185);

// 两个pair对象间的小于运算，其定义遵循字典次序：
// 如 p1.first < p2.first 或者 !(p2.first < p1.first) && (p1.second < p2.second) 则返回true。
p1 < p2;
// 如果两个对象的first和second依次相等，则这两个对象相等；该运算使用元素的==操作符。
p1 == p2;
p1.first;          // 返回对象p1中名为first的公有数据成员
p1.second;         // 返回对象p1中名为second的公有数据成员
```

04 双端队列

STL 中的 deque 容器提供了一众成员函数以供调用。其中较为常用的有：

```
deque<int> q;
```

```
// 元素访问
q.front();  // 返回队首元素
q.back();   // 返回队尾元素

// 修改
q.push_back();  // 在队尾插入元素
q.pop_back();   // 弹出队尾元素
q.push_front(); // 在队首插入元素
q.pop_front();  // 弹出队首元素
q.insert();     // 在指定位置前插入元素（传入迭代器和元素）
q.erase();      // 删除指定位置的元素（传入迭代器）

// 容量
q.empty() 队列是否为空
q.size() 返回队列中元素的数量
```