deque 双端队列

```
1-  d.push_back( 10 ); 在后面添加
    d.push_front( 5 );  在前面添加

    d.front(); d.pop_front();        取出前面的并弹出
    d.back(); d.pop_back();          取出后面的并弹出

     d.erase(d.begin() + 2);         删除d[2]

2- 空间可动态增加。队列容积增加不会发生元素的拷贝和重新填充。
```

1- 函数说明

在头文件`<deque>`中。

函数模板

```
template <class T, class Alloc = allocator <T>> class deque;
```

- 双端队列，可以在两端（前端或后端）扩展或收缩。
- 允许通过随机访问迭代器直接访问各个元素，并根据需要通过扩展和收缩容器来自动处理存储。
- 提供了类似于矢量的功能，但是在序列的开始部分也有效地插入和删除元素，而不仅仅是在其末尾。但是，与矢量不同，deques不保证将其所有元素存储在连续的存储位置：`deque`通过偏移指向其他元素的指针访问元素会导致*未定义的行为*。

Vector和deques提供了一个非常相似的接口，可用于类似的目的，但内部工作方式完全不同：虽然向量使用单个数组，偶尔需要重新分配以增长，但deque的元素可以分散在不同的块中的容器，容器在内部保存必要的信息以提供对其任何元素的持续时间和统一的顺序接口（通过迭代器）的直接访问。因此，deques在内部比vector更加复杂一些，但是这使得它们可以在某些情况下更有效地增长，特别是在重新分配变得更加昂贵的很长序列的情况下。

对于涉及频繁插入或移除开始或结束位置以外的元素的操作，deques表现得更差, 相比于 consistent iterators and references than lists and forward lists.

特点：

1- 内部有序

2- 允许以动态数组实现，允许直接访问序列中的任何元素，并在序列的开始或结束处提供相对较快的元素添加/删除。

3- 用 allocator object to dynamically handle its storage needs.

C++队列queue模板类的定义在头文件中,queue 模板类需要两个模板参数，一个是元素类型，一个容器类型，元素类型是必要的，容器类型是可选的，默认为deque 类型。

C++队列Queue是一种容器适配器，它给予程序员一种先进先出(FIFO)的数据结构。

2- 成员函数

deque容器为一个给定类型的元素进行**线性处理，像向量一样，它能够快速地随机访问任一个元素，并且能够高效地插入和删除容器的尾部元素。但它又与vector不同，deque支持高效插入和删除容器的头部元素，因此也叫做双端队列。**deque类常用的函数如下。

(1) 构造函数

deque():创建一个空deque

deque(int nSize):创建一个deque,元素个数为nSize

deque(int nSize,const T& t):创建一个deque,元素个数为nSize,且值均为t

deque(const deque &):复制构造函数

(2) 增加函数

void push_front(const T& x):双端队列头部增加一个元素X

void push_back(const T& x):双端队列尾部增加一个元素x

iterator insert(iterator it,const T& x):双端队列中某一元素前增加一个元素x

void insert(iterator it,int n,const T& x):双端队列中**某一元素前增加n个相同的元素x**

void insert(iterator it,const_iterator first,const_iteratorlast):双端队列中**某一元素前插入另一个相同类型向量的[forst,last)间的数据**

(3) 删除函数

Iterator erase(iterator it):删除双端队列中的某一个元素

Iterator erase(iterator first,iterator last):**删除双端队列中[first,last）中的元素**

void pop_front():删除双端队列中最前一个元素

void pop_back():删除双端队列中最后一个元素

void clear():清空双端队列中最后一个元素

(4) 遍历函数

reference at(int pos):返回pos位置元素的引用

reference front():返回手元素的引用

reference back():返回尾元素的引用

iterator begin():返回向量头指针，指向第一个元素

iterator end():返回指向向量中最后一个元素下一个元素的指针（不包含在向量中）

reverse_iterator rbegin():反向迭代器，指向最后一个元素

reverse_iterator rend():反向迭代器，指向第一个元素的前一个元素

(5) 判断函数

bool empty() const:向量是否为空，若true,则向量中无元素

(6) 大小函数

Int size() const:返回向量中元素的个数

int max_size() const:返回最大可允许的双端对了元素数量值

(7) 其他函数

void swap(deque&):交换两个同类型向量的数据

void assign(int n,const T& x):向量中第n个元素的值设置为x

```
// deque.cpp : 定义控制台应用程序的入口点。
//

#include <bits/stdc++.h>
//#include<iostream>
//#include<deque>

using namespace std;
int main(){
    deque<int> d;
    d.push_back( 10 );
    d.push_back(20);
    d.push_back(30);
    cout<<"原始双端队列："<<endl;
    for(int i = 0; i < d.size(); i++)
    {
        cout<<d.at(i)<<"\t";        // 10   20  30
    }
    cout<<endl;
    d.push_front(5);
    d.push_front(3);
    d.push_front(1);

    cout<<"after push_front(5.3.1):"<<endl;
    for(int i = 0;i < d.size();i++)
    {
```

```
        cout<<d.at(i)<<"\t";        // 1    3   5   10  20  30
    }
    cout<<endl;

    /// 弹出前两个元素
    d.pop_front();
    d.pop_front();
    cout<<"after pop_front() two times:"<<endl;
    for(int i = 0;i < d.size();i++)
    {
        cout<<d.at(i)<<"\t";        // 5    10  20  30
    }
    cout<<endl;

    d.erase(d.begin() + 2);
    for( int val: d )
        cout << val << "\t" ;       // 5    10  30

    cout<<endl;
    return 0;
}
```

deque与vector内存分配比较：

```
// deque.cpp : 定义控制台应用程序的入口点。
//


#include <bits/stdc++.h>
using namespace std;

int main(){
    vector<int>v(2);
    v[0]=10;
    int *p = &v[0];
    cout<<"vector第一个元素迭代指针*p="<<*p<<endl;
    v.push_back(20);
    cout<<"vector容量变化后原vector第1个元素迭代指针*p="<<*p<<endl;

    deque<int> d(2);
    d[0]=10;
    int *q = &d[0];
    cout<<"deque第一个元素迭代指针*q="<<*q<<endl;
    d.push_back(20);
    cout<<"deque容量变化后第一个元素迭代器指针*q="<<*q<<endl;
```

```
}
```

结果为：

```
vector第一个元素迭代指针*p=10
vector容量变化后原vector第1个元素迭代指针*p=10
deque第一个元素迭代指针*q=10
deque容量变化后第一个元素迭代器指针*q=10
```

下面这个应该是以前的内容，发现现在的编译器做了优化，指针并没有发生变化。移动语义赋值？？？

> 该段程序的功能是：deque、vector初始化后大小为2，第一个元素都为10，当通过push_back函数分别给两个容器增加一个元素后，从结果发现原先保持的指针元素值对vector容器前后发生了变化，而对deque容器前后没有发生变化。原因为，在建立vector容器时，一般来说伴随这建立空间->填充数据->重建更大空间->复制原空间数据->删除原空间->添加新数据，如此反复，保证vector始终是一块独立的连续内存空间；在建立deque容器时，一般便随着建立空间->建立数据->建立新空间->填充新数据，如此反复，没有原空间数据的复制和删除过程，是由多个连续的内存空间组成的。