原创 枫叶先生 已于 2023-10-30 12:56:34 修改 阅读量1.2k 收藏 17 点赞数 16 版权

分类专栏： C嘎嘎 # STL（C++） 文章标签： c++ 开发语言 数据结构

> 「前言」
> 「归属专栏」C嘎嘎
> 「主页链接」个人主页
> 「笔者」枫叶先生(fy)

目录

一、*priority_queue*介绍

priority_queue文档介绍

class template

std::priority_queue <queue>

```
template <class T, class Container = vector<T>,
  class Compare = less<typename Container::value_type> > class priority_queue;
```

Priority queue

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some *strict weak ordering* criterion.

This context is similar to a *heap*, where elements can be inserted at any moment, and only the *max heap* element can be retrieved (the one at the top in the *priority queue*).

Priority queues are implemented as *container adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *popped* from the *"back"* of the specific container, which is known as the *top* of the priority queue.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall be accessible through *random access iterators* and support the following operations:

- `empty()`
- `size()`
- `front()`
- `push_back()`
- `pop_back()`

The standard container classes `vector` and `deque` fulfill these requirements. By default, if no container class is specified for a particular `priority_queue` class instantiation, the standard container `vector` is used.

Support of *random access iterators* is required to keep a heap structure internally at all times. This is done automatically by the container adaptor by automatically calling the algorithm functions `make_heap`, `push_heap` and `pop_heap` when needed.

CSDN @Maple_叶卿川

> 翻译：

1. 优先队列是一种容器适配器，根据严格的弱排序标准，它的第一个元素总是它所包含的元素中最大的
2. 此上下文类似于堆，在堆中可以随时插入元素，并且只能检索最大堆元素(优先队列中位于顶部的元素)
3. 优先队列被实现为容器适配器，容器适配器即将特定容器类封装作为其底层容器类，queue提供一组特定的成员函数来访问其元素。元素从

empty()：检测容器是否为空

size()：返回容器中有效元素个数

front()：返回容器中第一个元素的引用

push_back()：在容器尾部插入元素

pop_back()：删除容器尾部元素

- （5）标准容器类 *vector* 和 *deque* 满足这些需求。默认情况下，如果没有为特定的 *priority_queue* 类实例化指定容器类，则使用 *vector*
- （6）需要支持随机访问迭代器，以便始终在内部保持堆结构。容器适配器通过在需要时自动调用算法函数 *make_heap*、*push_heap* 和 *pop_heap* 来自动完成此操作

```
template <class T, class Container = vector<T>,
  class Compare = less<typename Container::value_type> > class priority_queue;
```

优先级队列 priority_queue 默认使用的适配器是 *vector*（*class Container = vector<T>*），第三个模板参数是仿函数（*class Compare = less<typename Container::value_type>* ），仿函数　下面解释

Template parameters

`T`
Type of the elements.
Aliased as member type `priority_queue::value_type`.

`Container`
Type of the internal *underlying container* object where the elements are stored.
Its `value_type` shall be `T`.
Aliased as member type `priority_queue::container_type`.

`Compare`
A binary predicate that takes two elements (of type `T`) as arguments and returns a `bool`.
The expression `comp(a,b)`, where *comp* is an object of this type and *a* and *b* are elements in the container, shall return `true` if *a* is considered to go before *b* in the *strict weak ordering* the function defines.
The `priority_queue` uses this function to maintain the elements sorted in a way that preserves *heap properties* (i.e., that the element popped is the last according to this *strict weak ordering*).
This can be a function pointer or a function object, and defaults to `less<T>`, which returns the same as applying the *less-than operator* (`a<b`).

使用 *priority_queue* 要包含头文件 <*queue*>，与 *queue* 的头文件一致

```
#include <queue>
```

注：***priority_queue* 的结构就是堆，即二叉树**

二、*priority_queue*使用

成员函数介绍：

Member functions

(constructor)	Construct priority queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access top element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

接口简单介绍，C++11先不介绍，后面再学

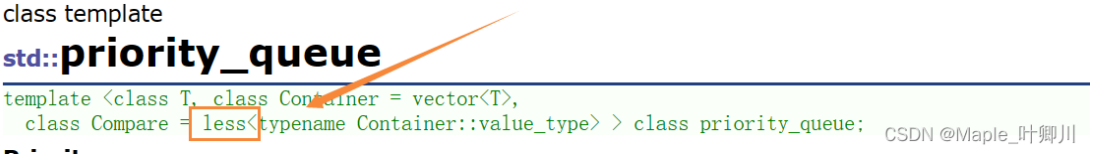
函数声明	接口说明
priority_queue()	构造一

priority_queue(first,last)	以迭代器区间构造一个优先级队列
empty( )	检测优先级队列是否为空，是返回true，否则返回false
top( )	返回优先级队列中最大(最小元素)，即堆顶元素
push(x)	在优先级队列中插入元素x
pop()	删除优先级队列中最大(最小)元素，即堆顶元素

> 优先级队列默认使用 *vector* 作为其底层存储数据的容器，在 *vector* 上又使用了堆算法将 *vector* 中元素构造成堆的结构，因此 *priority_queue* 就是堆，所有需要用到堆的位置，都可以考虑使用 *priority_queue*

注意：默认情况下 *priority_queue* 是大堆（即降序）

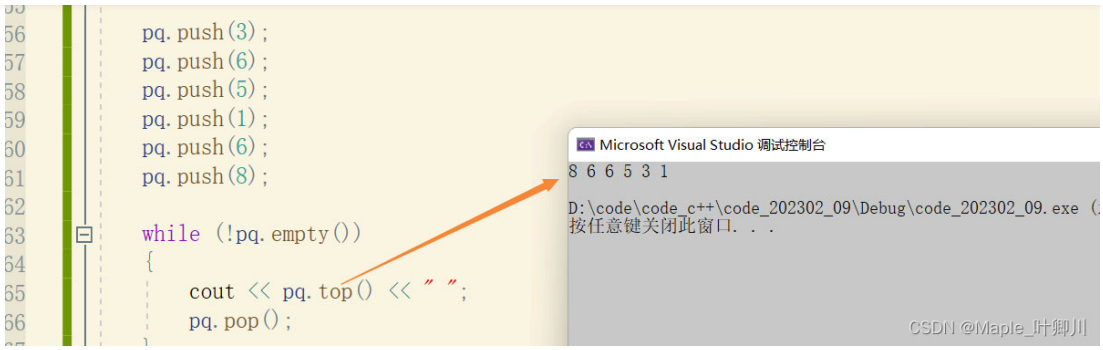
> 优先级队列默认大的优先级高，传的是 *less* 仿函数，底层是一个大堆；如果想控制小的优先级高，需手动传 *greater* 仿函数，其底层是一个小堆



测试代码

```cpp
#include <iostream>
#include <queue>
using namespace std;

void Test_priority_queue()
{
    priority_queue<int> pq;

    pq.push(3);
    pq.push(6);
    pq.push(5);
    pq.push(1);
    pq.push(6);
    pq.push(8);

    while (!pq.empty())
    {
        cout << pq.top() << " ";
        pq.pop();
    }
    cout << endl;
}

int main()
{
    Test_priority_queue();
    return 0;
}
```

运行结果，降序



如果要改成小堆（即升序），需要手动传第三个参数 *greater* ，使用 greater 需要包含头文件 <functional>

```
#include <functional>  // greater的头文件
```

要传第三个模板参数，就必须传第二个模板参数，这是使用缺省参的语法规定

```
priority_queue<int, greater<int>> pq; //error
```

正确使用如下

```
priority_queue<int, vector<int>, greater<int> > pq;
```

```
#include <iostream>
#include <queue>
#include <functional>
using namespace std;

void Test_priority_queue()
{
    priority_queue<int, vector<int>, greater<int> > pq;

    pq.push(3);
    pq.push(6);
    pq.push(5);
    pq.push(1);
    pq.push(6);
    pq.push(8);

    while (!pq.empty())
    {
        cout << pq.top() << " ";
        pq.pop();
    }
    cout << endl;
}

int main()
{
    Test_priority_queue();
    return 0;
}
```

运行结果，升序

```
pq.push(3);
pq.push(6);
pq.push(5);
pq.push(1);
pq.push(6);
pq.push(8);

while (!pq.empty())
{
    cout << pq.top() << " ";
```

```
Microsoft Visual Studio 调试控制台
1 3 5 6 6 8

D:\code\code_c++\code_202302_09\Debug\code_202302_
按任意键关闭此窗口. . .
```


三、仿函数

仿函数是 STL 的六大组件之一

> 仿函数：**使一个类的使用看上去像一个函数，该类定义出来的对象称为函数对象**

仿函数的实现就是在类中重载 () 操作符，使得该类具备类似函数的行为，就是一个仿函数

测试，创建一个仿函数

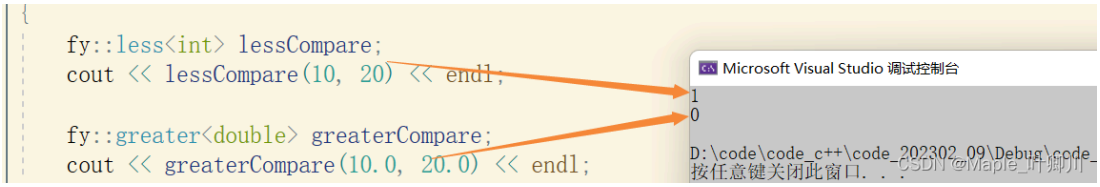
```
namespace fy
{
    class less
    {
    public:
        bool operator()(int x, int y)
        {
            return x < y;
        }
    };
}

void Test()
{
    fy::less lessCompare;
    cout << lessCompare(10, 20) << endl;
}
```

运行结果

仿函数支持泛型

```cpp
namespace fy
{
    template<class T>
    class less
    {
    public:
        bool operator()(const T& x, const T& y)const
        {
            return x < y;
        }
    };

    template<class T>
    class greater
    {
    public:
        bool operator()(const T& x, const T& y)const
        {
            return x > y;
        }
    };
}

void Test()
{
    fy::less<int> lessCompare;
    cout << lessCompare(10, 20) << endl;

    fy::greater<double> greaterCompare;
    cout << greaterCompare(10.0, 20.0) << endl;
}
```

运行结果



这就是仿函数，实际上仿函数调用的只是重载的 *operator()* 而已

为什么要有仿函数？为了替代 C语言中的函数指针，使用代码更直观，函数指针使用起来头大，难理解！！

```cpp
void( * set_malloc_handler(void (*f)()))() //这是函数指针.......
```

所以，仿函数的出现就是为了替代 C语言的函数指针

四、*priority_queue*模拟实现

优先级队列的模拟实现也是很简单，因为它是容器适配器，可以使用已有的生成我们想要的

4.1 版本1

这里要注意的向上调整和向下调整的写法，数据结构已经学过，就不废话了。这个版本的缺陷是不能很好的支持升序和降序，只能是其中的一种，使用改升序或降序就必须修改源代码，修改后也是两者不能兼并，极其不好，版本2 则是使用仿函数，可以使两者可以同