一、容器分类：
1.序列式容器(Sequence containers)
*每个元素有固定位置-取决于插入时间,与元素值无关
*vector,deque,list,stack,queue

2.关联式容器(Associated containers)
*元素位置取决于特定的排序准则,与插入顺序无关
*set,multiset,map,muliset

vector }动态数组
deque }动态数组
list }链表
stack
queue
set和multiset
Map和multimap

序列式容器

二、vector
1.vector简介
*元素放到动态数组中进行管理,内部实现采用动态数组
*可以随机存取元素（索引值直接存取,用[]或者at()方法）
*尾部添加或移除元素非常快,但在中部或头部插入移除比较慢

2.vector默认构造
采用模板类实现
vector 采用模板类实现, vector 对象的默认构造形式: vector<T> vecT;
vector<int> vecInt; // 一个存放 int 的 vector 容器
vector<float> vecFloat; // 一个存放 float 的 vector 容器
vector<string> vecString; // 一个存放 string 的 vector 容器
... // 尖括号内还可以设置指针类型或自定义类型

Class CA{};
vector<CA*> vecpCA; // 用于存放 CA 对象的指针的 vector 容器
vector<CA> vecCA; // 用于存放 CA 对象的 vector 容器

node:由于容器元素的存放是按值复制的方式 进行的,所以此时 CA 必须提供
CA 的拷贝构造函数,以保证 CA 对象间拷贝正常。

3.vector带参构造
1)vector(beg,end);//构造函数将[beg,end)区间的元素拷贝给本身
例子:int iarr[] = {1,2,3,4,5}; vector<int > v1(iarr,iarr+5);//左闭右开[)
2)vector(n,elem);//构造函数将n个elem拷贝给本身
3)vector(const vector& vec);拷贝构造函数

4.vector赋值 assign会清空后赋值,覆盖原本的元素
1) vector.assign(beg,end);//将[beg,end)区间的元素拷贝赋值给本身。
vecIntB.assign(vecIntA.begin(),vecIntA.end());//将vecIntA的所有元素赋给vecIntB

2) vector.assign(n,elem);将n个elem赋值给本身
3) vector& operator=(const vector &vec);//重载等号操作符
4) vector.swap(vec);//将vec与本身的元素互换

5.vector的大小
    1)vector.size();//返回容器中元素个数
    2)vector.empty();//判断容器是否为空
    3)vector.resize(num);//重新指定容器长度为num,容器变长,则以默认值填充;
        容器变短,则末尾超出容器长度的元素被删除
    4)vector.resize(num,elem);//重新指定容器长度为num,容器变长,则以elem填充;
        容器变短,则末尾超出容器长度的元素被删除

6.vector访问元素
    1) vec.[idx]; //返回索引idx的元素,下标越界和数组的越界出错基本相同
                    ,可能导致程序异常终止,没有报错信息
    2) vec.at(idx); //返回索引idx的元素,下标越界会报错,推荐使用

7.vector末尾的添加移除操作
    vector<int> vecInt;
    1) vecInt.push_back(1);//容器尾部添加一个元素 1
    2) vecInt.push_back(3);
    3) vecInt.push_back(5);
    4) vecInt.pop_back(); //容器尾部删除一个元素

7.vector插入元素 insert
    1) vector.insert(pos,elem);//在下标pos插入elem,pos应为指针,比
            //;vector.begin()+n; 返回值为新数据的位置
    2) vector.insert(pos,n,elem);//在pos插入n个elem,无返回值
    3) vector.insert(pos,beg,end);//在pos插入[beg,end)的元素,无返回值

8.vector删除元素
https://blog.csdn.net/lishichengyan/article/details/82669153
    1)vector::clear();clear用来清空整个vector,同时将size变成0,无返回值
    2)vector::erase();erase通过传入迭代器进行删除,既可以删除单个元素,
                ;//也可以删除某一范围的元素,删除之后它将返回下一个位置的迭代器
    3)vector::pop_back();pop_back用来删除末尾元素,同时将size减1,无返回值。

8.迭代器 可以不用[]或at()进行访问,可避免出现越界的
    迭代器是一种检查容器内元素并遍历容器内元素的数据结构
    通过迭代器统一了对所有容器的访问方式。

    迭代器作用：迭代器提供对一个容器的方法的访问方法,并且定义了容器中对象的范围

    为什么需要迭代器?
        *STL提供每种容器的实现原理各不相同,没有迭代器我们需要记住每一种容器中
          对象的访问方法,很明显这样会变得麻烦
        *每个容器中都实现了一个迭代器用于对容器中对象的访问,虽然每个容器中迭代器
          的实现方式不一样,但对于用户来说操作方式一致,
          即：通过迭代器统一了对所有容器的访问方式。
          例 ：无论哪个容器,访问当前元素的下一个元素我们可以通过迭代器自增进行访问
        *迭代器是为了提高编程效率而开发的

    vector容器的迭代器iterator类型
        1) vector<int>::iterator iter;//变量名为iter
        2) vector容器的迭代器属于“随机访问迭代器”:迭代器一次可以移动多个位置

9.迭代器的失效
注:insert和erase之后都可能导致迭代器失效,因此需要把返回值赋给新的迭代器来使用

1) 插入元素后失效
```
it = vector.insert(it,elem); //insert会返回一个新的有效的迭代器
```
2) 删除元素后失效
```
it2 = vector.erase(it);//返回一个新的有效的迭代器,此迭代器指向删除元素
    ;//的下一个元素
#if 0
for(it = cond.begin(); it != cond.end() ;)
{
    if(*it == 3)
        cond.erase(it);//it可能失效
            ;//g++对这种有特殊处理,it指向下一个,别的编译器就可能失效了,无法继续使用迭代器

    else
        it++;

}
#endif

for(it = cond.begin() ; it != cond.end();)
{
    if(*it == 3){
        it = cond.erase(it);//推荐使用
    }
    else it++;
}
```

因为在insert时,vector可能需要进行扩容,而扩容的本质是new一块新的空间,
再将数据迁移过去。而迭代器的内部是通过指针访问容器中的元素的,插入后,
若vector扩容,则原有的数据被释放,指向原有数据的迭代器就成了野指针,
所以迭代器失效了。

https://blog.csdn.net/weixin_42157432/article/details/108203601

三、deque容器
1.deque容器简介
*deque是"double-ended queue"的缩写,和vector一样都是STL的容器
*deque是双端数组而vector是单端的。
*deque在接口上和vector非常相似,在许多操作的地方可以直接替换。
*deque可以随机存取元素(支持索引值直接存取,用[]操作符或at()方法)
*deque头部和尾部添加或移除元素都非常快速。但是在中部安插元素或移除元素比较费时。
*#include<deque>

2.deque容器的操作
1)deque与vector在操作上几乎一样,deque多两个函数
2)deque.push_front(elem);//在容器头部插入一个数据
3)deque.pop_front();//删除容器第一个元素

queue并非是连续空间存储的,他是分段存储的(应该是链表加数组)

四、deque容器

1.deque简介

1)list是一个双向链表容器,可高效的进行插入删除操作

2)list不可以随机存取元素,所以不支持at()或[]操作符

```
it++ (Ture)     it+5(err)
```

3)#include<list>

2.list对象的默认构造

list采用模板类实现,对象的默认构造形式:list<T> lst;

3.list头尾的添加移除操作

```
list.push_back(elem);//容器尾部插入一个元素
list.pop_back();
list.push_front(elem);
list.pop_front();
```

4.list访问元素

访问 list 容器中存储元素的方式很有限,即要么使用 front() 和 back() 成员函数,要么使用 list 容器迭代器

```
int x = list.front();
int y= list.front();
list.front() = 10
list.back() = 100;
```

5.list与迭代器

list容器迭代器是“双向迭代器”: 双向迭代器从两个方向读写容器。除了提供前向迭代器的全部操作之外,双向迭代器还提供前置和后置的自减运算

```
list.begin(); //返回容器中第一个元素的迭代器
list.end(); //返回容器中最后一个元素的迭代器
list.rbegin();  //返回容器中倒数第一个元素的迭代器
list.rend(); //返回容器中倒数最后一个元素的后面的迭代器

//反向输出list
//list容器的反向迭代器
list<int>::reverse_iterator it1;
for(it1 = lst.rbegin() ; it1 != lst.rend(); it1 ++)
    cout<<*it1<<" ";
cout<<endl;
```

6.list对象的带参构造

```
1) list(n,elem); //构造函数将n个elem拷贝给本身
2) list(beg,end); //参数是迭代器,构造函数将[beg,end)区间的元素拷贝给本身
3) list(const list&list); //拷贝构造函数
```

7.list的赋值

assign是覆盖式的函数,赋值

```
list.assign(beg,end); //将[beg,end)区间的数据拷贝赋值给本身
list.assign(n,elem);    //将n个elem拷贝赋值给本身
list& operator=(const list&list);   //重载等号操作符
list.swap(lst);将lst与本身的元素互换
```

8.list的大小

```
1) list.size(); //返回容器中元素的个数
2) list.empty(); //判断容器是否为空
3) list.resize(num); //重新指定容器长度为num,若容器变长,则以默认值填充新位置。
```

```
            //若容器变短,则末尾超出容器长度的元素被删除
    4) list.resize(num,elem);//重新指定容器长度为num,若容器变长,则以elem填充新位置。
            //若容器变短,则末尾超出容器长度的元素被删除
```

9.list的插入

```
    list的insert不会导致迭代器失效
    1).list.insert(pos,elem); //在pos位置插入一个elem元素的拷贝,返回新数据的位置
    2).list.insert(pos,n,elem); //在pos位置插入n个elem数据,无返回值
    3) list.insert(pos,beg,end); //在pos位置插入[beg,end)区间的数据,无返回值
```

10.list的删除

```
    1) list.clear(); //移除容器的所有数据
    2) list.erase(beg,end); //删除[beg,end)区间的数据,返回下一个数据的位置
    3) list.erase(pos); //删除pos位置的数据,返回下一个数据的位置
    4) list.remove(elem); //删除容器中所有与elem值匹配的元素
```

11.list的反序排列

```
    1)list.reverse(); //反转链表,比  list包含1,3,5元素,运行此方法后,list逆转
      例: list<int> lstA;
            lstA.push_back(1);
            lstA.push_back(3);
            lstA.push_back(5);
            lstA.push_back(7);
            lstA.push_back(9);

            lstA.reverse();//9 7 5 3 1
```

12.list迭代器失效

```
    1) 删除结点导致迭代器失效
        erase返回值为删除元素的下一个元素的迭代器
    2) 插入不会导致迭代器失效
```

五、stack容器

1.stack对象的默认构造

```
    stack采用类模板实现,stack对象的默认构造形式stack<T> s;
        stack<int> stkint; //一个存放int类型的stack容器
        stack<float> stkint; //一个存放float类型的stack容器
```

2.stack的入栈与出栈

```
    1).stack的push()与pop()方法
        stack.push(elem); //往栈顶添加元素
        stack.pop(); //从栈顶移除第一个元素,没有返回值
    2).stack的pop()方法
        stack.pop(); //返回栈顶元素,不删除
```

3.stack对象的拷贝构造与赋值

```
    1)stack(const stack& stk); //拷贝构造函数
    2)stack & operator=(const stack &stk);//重载等号操作符
        stack<int> stkintA;
        stkintA.push(1);
        stkintA.push(3);
        stkintA.push(5);
        stkintA.push(7);
        stkintA.push(9);
```

```
        stack<int> stkIntB(stkintA);//拷贝构造
        stack<int> stkIntC;
        stkIntC = stkintA; //赋值
```

4.stack的大小

1) stack.empty();//判断堆栈是否为空
2) stack.size(); //返回堆栈的大小

注:stack容器没有提供resize方法

六、queue容器

1.queue容器简介

1)queue是队列容器,是一种“先进先出”容器
2)#include<queue>

2.queue对象的默认构造

queue采用模板类实现,queue对象的默认构造: queue<T> q;

3.queue的入栈和出栈

1)queue.push(elem); //往队尾添加一个元素
2)queue.pop(); //从对头移除第一个元素

```
    //queue容器也没有提供迭代器,因为不能遍历
    //    cout<<q1.front()<<endl; //输出队首的元素
    //    q1.pop();//删除队首,出队
```

4.queue容器对象的拷贝构造与赋值

1)queue(const queue&que); //拷贝构造函数: 用已经构造好的容器去构造一个未初始化的容器
2)queue& operator=(const queue& que); //重载等号操作符

```
    queue<int> queIntB(queIntA); //拷贝构造
    queue<int> queIntB = queIntA; //拷贝构造

    queue<int> queIntC;
    queIntC = queIntA; //拷贝
```

5.queue容器数据存取

1)queue.back(); //返回最后一个元素,队尾
2)queue.front(); //返回第一个元素,队首,返回值可以作为表达式的左值

6.queue容器的大小

1)queue.empty(); //判断队列是否为空
2)queue.size(); front//返回队列的大小

关联式容器

七、Set和multiset容器

1.set和multiset容器简介

1） set是一个集合容器,其中所包含的元素是唯一的,集合的元素按一定的顺序排列。元素插入过程是按派序规则插入,所以不能指定插入位置
2） set采用红黑数变体的数据结构实现,红黑数属于平衡二叉树。在插入操作和删除操作上比vector快
3)set不可以直接存取元素(不可以使用at(pos)与[]操作符)
4)multiset与set的区别:set支持唯一键值,每个元素只能出现一次。而multiset中同一值可以出现多次
5)不可以直接修改set或multiset容器中的元素值,因为该类容器是自动排序的。 果希望修改一个元素值,必须先删除原有的元素,再插入新的元素
6)#include<set>

2.set容器插入与迭代器
1)set.insert(elem); //在容器中插入元素
2)set.begin(); //返回容器中第一个数据的迭代器
3)set.end(); //返回容器中最后一个数据之后的迭代器
4)set.rbegin(); //返回容器中倒数第一个元素的迭代器
5)set.rend(); //返回容器中倒数最后一个元素的后面的迭代器

//默认升序排序

3.set容器拷贝构造与赋值
1)set(const set& st); 拷贝构造函数
2)set&operator=(const set&st);
3)set.swap(st);//交换两个集合容器