**Deque Operations**

**Constructors and Destructor of Deques**

|  |  |
| --- | --- |
| **Operation** | **Effect** |
| deque< El > **c** | Creates an empty deque without any elements |
| deque< El > **c1**(c2) | Creates *c1* as a copy of *c2* of the same type(the copy has the same size as the original) |
| deque< El > **c**(nr) | Creates a deque *c* of size *nr* containing *nr* values, each equal to the default value of type *El* |
| deque< El > **c**(nr,el) | Creates a deque initialized with *nr* copies of element *el* |
| deque< El > **c**(beg,end) | Creates a deque initialized with the elements of the range *[beg,end)* |
| **c.~deque < El >**() | Destroy all components of *c* and free the associated memory |

**Nonmodifying Operations of Deques**

|  |  |
| --- | --- |
| **Operation** | **Effect** |
| **c.size**() | Returns a value of type size\_type giving the number of values currently in *c* |
| **c.empty**() | Returns true if *c* is empty (contains zero values); otherwise returns false |
| **c.max\_size**() | Returns the maximum number of elements possible |
| **c1==c2** | Returns if *c1* is equal to *c2* |
| **c1!=c2** | Returns if *c1* is not equal to *c2* (equivalent to **!(c1==c2)**) |
| **c1<c2** | Returns if *c1* is less than *c2* |
| **c1>c2** | Returns if *c1* is greater than *c2* (equivalent to **c2<c1**) |
| **c1<=c2** | Returns if *c1* is less than or equal to *c2* (equivalent to **!(c2<c1)**) |
| **c1>=c2** | Returns if *c1* is greater than or equal to *c2* (equivalent to **!(c1<c2)**) |
| **c.at**(i) | Returns the element with index *i* (throws range error exception if *i* is out of range) |
| **c[i]** | Returns the element with index *i* (without range checking) |
| **c.front**() | Returns the first element (no check if a first element exists) |
| **c.back**() | Returns the last element (no check if a last element exists) |
| **c.begin**() | Returns a random access iterator for the first element |
| **c.end**() | Returns a random access iterator for the position after the last element |
| **c.rbegin**() | Returns a reverse iterator for the first element of a reverse iteration |
| **c.rend**() | Returns a reverse iterator for the position after the last element of a reverse iteration |

**Modifying Operations of Deques**

|  |  |
| --- | --- |
| **Operation** | **Effect** |
| **c1=c2** | Assigns *c2* to *c1*, and returns the common value. The deque on the left of an assignment receives the values and size of the one on the right |
| **c.assign**(nr,el) | Assigns *nr* copies of *el* to *c*, overwriting the entire previous contents of *c* |
| **c.assign**(beg,end) | Assigns the elements of the range *[beg,end)* |
| **c1.swap**(c2) | Exchanges values in *c1* with those in *c2*. The sizes are swapped as well |
| **swap**(c1,c2) | Same (as global function) |
| **c.insert**(pos,el) | Inserts *el* into *c* immediately before the value pointed to by *pos*, and returns an iterator pointing at the new component with value *el* |
| **c.insert**(pos,nr,el) | Inserts *nr* copies of *el* into *c*, immediately before the value pointed to by *pos* |
| **c.insert**(pos,beg,end) | Inserts at iterator position *pos* a copy of all elements of the range *[beg,end)* (returns nothing) |
| **c.push\_back**(el) | Adds *el* to the end of *c*, increasing the size of *c* by one |
| **c.pop\_back**() | Removes the last value of *c*. The size of *c* is reduced by one |
| **c.push\_front**(el) | Adds *el* to the front of *c*, increasing the size of *c* by one |
| **c.pop\_front**() | Removes the first value of *c*. The size of *c* is reduced by one |
| **c.erase**(pos) | Removes the element at iterator position *pos* and returns the position of the next element |
| **c.erase**(beg,end) | Removes all elements of the range *[beg,end)* and returns the position of the next element |
| **c.resize**(nr) | Changes the number of elements to *nr* (if **size**() grows, new elements are created by their default constructor) |
| **c.resize**(nr, el) | Changes the number of elements to *nr* (if **size**() grows, new elements are copies of *el*) |
| **c.clear**() | Removes all values of *c*. The size of *c* is reduced to zero |