**CSCI 3110**

# Vector Container Class methods

## Constructors and Destructors of Vectors

| Operation | Effect |
|---|---|
| `vector<Elem> c` | Creates an empty vector without any elements |
| `vector<Elem> c1(c2)` | Creates a copy of another vector of the same type (all elements are copied) |
| `vector<Elem> c(n)` | Creates a vector with `n` elements that are created by the default constructor |
| `vector<Elem> c(n,elem)` | Creates a vector initialized with `n` copies of element `elem` |
| `vector<Elem> c(beg,end)` | Creates a vector initialized with the elements of the range `[beg,end)` |
| `c.~vector<Elem>()` | Destroys all elements and frees the memory |

## Nonmodifying Operations of Vectors

| Operation | Effect |
|---|---|
| `c.size()` | Returns the actual number of elements |
| `c.empty()` | Returns whether the container is `empty` (equivalent to `size()==0`, but might be faster) |
| `c.max_size()` | Returns the maximum number of elements possible |
| `capacity()` | Returns the maximum possible number of elements without reallocation |
| `reserve()` | Enlarges capacity, if not enough yet[7] |
| `c1 == c2` | Returns whether `c1` is equal to `c2` |
| `c1 != c2` | Returns whether `c1` is not equal to `c2` (equivalent to `!(c1==c2)`) |
| `c1 < c2` | Returns whether `c1` is less than `c2` |
| `c1 > c2` | Returns whether `c1` is greater than `c2` (equivalent to `c2<c1`) |
| `c1 <= c2` | Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2<c1)`) |
| `c1 >= c2` | Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1<c2)`) |

## Assignment Operations of Vectors

| Operation | Effect |
|---|---|
| `c1 = c2` | Assigns all elements of `c2` to `c1` |
| `c.assign(n,elem)` | Assigns `n` copies of element `elem` |
| `c.assign(beg,end)` | Assigns the elements of the range `[beg,end)` |
| `c1.swap(c2)` | Swaps the data of `c1` and `c2` |
| `swap(c1,c2)` | Same (as global function) |

## Direct Element Access of Vectors

| Operation | Effect |
|---|---|
| `c.at(idx)` | Returns the element with index `idx` (throws range error exception if `idx` is out of range) |
| `c[idx]` | Returns the element with index `idx` (*no* range checking) |
| `c.front()` | Returns the first element (*no* check whether a first element exists) |
| `c.back()` | Returns the last element (*no* check whether a last element exists) |

## Iterator Operations of Vectors

| Operation | Effect |
|---|---|
| `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 |

## Insert and Remove Operations of Vectors

| Operation | Effect |
|---|---|
| `c.insert(pos,elem)` | Inserts at iterator position `pos` a copy of `elem` and returns the position of the new element |
| `c.insert(pos,n,elem)` | Inserts at iterator position `pos` `n` copies of `elem` (returns nothing) |

## Insert and Remove Operations of Vectors

| Operation | Effect |
|---|---|
| `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(elem)` | Appends a copy of `elem` at the end |
| `c.pop_back()` | Removes the last element (does not return it) |
| `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(num)` | Changes the number of elements to num (if `size()` grows, new elements are created by their default constructor) |
| `c.resize(num,elem)` | Changes the number of elements to num (if `size()` grows, new elements are copies of `elem`) |
| `c.clear()` | Removes all elements (makes the container empty) |

## Special Operations of `vector<bool>`

| Operation | Effect |
|---|---|
| `c.flip()` | Negates all Boolean elements (complement of all bits) |
| `m[idx].flip()` | Negates the Boolean element with index `idx` (complement of a single bit) |
| `m[idx] =` *val* | Assigns *val* to the Boolean element with index `idx` (assignment to a single bit) |
| `m[idx1] = m[idx2]` | Assigns the value of the element with index `idx2` to the element with index `idx1` |