## C++ vector class

The vector template class provides a form of dynamic array that expands at the end as necessary to accommodate additional elements.

Assume that **T** is some type (e.g., int) and the following declarations are in effect:

```
T e;
vector<T>  v1, v2;
vector<T>::iterator iter, iter2, beg, end;
vector<T>::reverse_iterator riter;  /* beg, end could also be here */
int i, n;
bool b;
```

### Common vector constructors, functions, operators, etc

| Result | Method | | Description |
|---|---|---|---|
| *Constructors and destructors* | | | |
| | vector<T> v1; | - | Creates an empty vector of T's. |
| | vector<T> v2(*v1*); | - | Creates vector v2 which is a duplicate of vector v1. |
| | vector<T> v(*n*); | - | Creates vector of size *n* with default values. |
| | vector<T> v(*n*, *e*); | - | Creates vector of *n* copies of *e*. |
| | vector<T> v(*beg*, *end*); | - | Creates vector with elements copied from range *beg*..*end*. |
| | v.~vector<T>(); | - | Destroys all elems and frees memory. |
| *Size* | | | |
| i = | v.**size**(); | - | Number of elements. |
| i = | v.**capacity**(); | - | Max number of elements before reallocation. |
| i = | v.**max_size**(); | - | Implementation max number of elements. |
| b = | v.**empty**(); | - | True if empty. Same as v.size()==0 |
| | v.**reserve**(*n*); | - | Sets capacity to *n* before reallocation |
| | v.**resize**(n) | - | Resizes v to have n additional elements. New elements initialized to default for data type. |
| *Altering* | | | |
| v1 = | v2; | - | Assigns *v2* to *v1*. v1 and v2 will be identical after this operation. |
| v**[i]** = | e; | - | Sets ith element (zero based indexing). |
| v.**at**(i) = | e; | - | Same as for v[i], but may throw out_of_range exception. |
| v.**front**() = | e; | - | Same as v[0] = e. |
| v.**back**() = | e; | - | Same as v[v.size()-1] = e. |
| | v.**push_back**(*e*); | - | Adds *e* to end of *v*. Expands v if necessary. |
| | v.**pop_back**(); | - | Removes last element of v. |
| | v.**clear**(); | - | Removes all elements. |
| iter = | v.**assign**(*n*, *e*); | - | Replaces existing elements with *n* copies of *e*. |
| iter = | v.**assign**(*beg*, *end*); | - | Replaces existing elements with copies from range *beg*..*end*. |
| iter2 = | v.**insert**(*iter*, *e*); | - | Inserts a copy of *e* at *iter* position and returns its position. |
| | v.**insert**(*iter*, *n*, *e*); | - | Inserts *n* copies of *e* starting at *iter* position. |
| | v.**insert**(*iter*, *beg*, *end*); | - | Inserts all elements in range *beg*..*end*, starting at *iter* position. |
| iter2 = | v.**erase**(*iter*); | - | Removes element at *iter* position and returns position of next element. |
| iter = | v.**erase**(*beg*, *end*) | - | Removes range *beg*..*end* and returns position of next element. |
| *Access* | | | |
| e = | v**[i]**; | - | ith element. No range checking. |
| e = | v.**at**(i); | - | Same as for v[i], but may throw out_of_range exception. |
| e = | v.**front**(); | - | First element. No range checking. |
| e = | v.**back**(); | - | Last element. No range checking. |
| *Iterators (operators apply to both forward and reverse iterators)* | | | |
| iter = | v.**begin**(); | - | Returns iterator to first element. |
| iter = | v.**end**(); | - | Returns iterator to *after* last element. |
| riter = | v.**rbegin**(); | - | Returns iterator to first (in reverse order) element. |
| riter = | v.**rend**(); | - | Returns iterator to *after* last (in reverse order) element. |
| | ++*iter;* | - | Preincrement *iter* to next element. Use in both forward and reverse iterators normally. Can also postincrement. Returns value. |
| | --*iter;* | - | Predecrement *iter*. Can also postincrement. Returns value. |
| iter2 = | *iter* + i; | - | Iterator *i* elements after *iter*. += assigment also defined. |
| iter2 = | *iter* - i; | - | Iterator *i* elements before *iter* -= assignment also defined. |
| e = | ***iter**; | - | Dereference the iterator to get the value. |