Standard Template Library (STL)

C++ STL Topcoder

Original Articles:

- Power up C++ with the Standard Template Library: Part 1
- Power up C++ with the Standard Template Library: Part 2

Container types (algorithms and all STL) are not declared in the global namespace, they are defined in a special namespace called `std`. Hence add the following line:

```
using namespace std;
```

The type of the container is defined by its template parameter specified as follows:

```
vector<int> N;
```

When declaring nested containers make sure the brackets are not consecutive or the compiler may confuse it with the `>>` operator.

```
vector<vector<int>> N;  // incorrect
vector<vector<int> > N; // correct
```

Vectors

Vector is just an array with extended capabilities. BTW vector is the only container that is backwards compatible with C.

```
vector<int> v;      // an empty vector is created (with no elements)
/* be careful with declarations like these */
vector<int> v[10];  // this creates an array of 10 empty vectors
vector<int> v(10);  // this creates a vector with 10 integers all initialized to 0
```

Checking the size of the vectors is one of the most frequent operations. But, make sure if you are just checking if the size of the vector is 0 you utilize the `empty` function. Because, not all containers can report their size if O(1) time and it is unecessary to count each element if you only need to check if it is empty or not.

```
int num_of_elems = v.size();      // returns the size in unsigned int (sometimes may give problems)
bool is_empty = v.size() != 0;    // Not recommended
bool is empty = !v.empty();       // Better implementation
```

Another popular function is the `push_back` function in vectors. It appends a single value to the back of the vector. Don't worry about memory allocation, it never just allocates memory for 1 element. It allocates more memory than it actually needs. Sometimes slower, better idea to initialise the vector beforehand if you know the size. Also need to worry about the memory usage.

```
vector<int> v;
for (int i = 0; i < 100000; ++i){
  v.push_back(i*i);
}
int num_elements = v.size();
```

When you need to resize the vector use the `resize` function. If the new size is greater than the old size, old elements are preserved and new elements are initialized to 0. Otherwise, the last ones will be deleted.

```
vector<int> v(20);
for (int i = 0; i < 20; ++i){
  v[i] = i*i;
}
v.resize(25);
for (int i = 20; i < 25; ++i){
  v[i] = 2*i + 1;
}
/* Note that if you push_back after size the new elements will be appended after not
filled in place of */
```

To delete all elements use the `clear` function. To insert elements at any position use `insert`, to delete use `erase`.
There are many ways to initialise a vector.

```
vector<int> v1;
// initialize using another vector
vector<int> v2 = v1;
vector<int> v3(v1);
// create vector of a specific size
vector<int> v(1000);
// create vector of a specific size and filled with an initial value other than 0.
vector<string> arr(10, "Unkown");
// Multidimensional vectors
vector<vector<int> > Matrix;
// create a 2D vector of any size say MxN filled with -1, it uses a recursive approach
that can be extended to higher dimensions
vector<vector<int> > grid(M, vector<int>(N, -1));
```

When vectors are passed to a function they are copied as whole (this is required very rarely). It is better to pass a reference to the vector.

```
void operate_vector(vector<int> v){   // shouldn't unless really sure
  // ...
  // ...
}

void operate_vector(const vector<int> &v){  // vector passed as reference
```

```
  // ...
  // ...
}

void modify_vector(vector<int> &v){   // remove the const
  // ...
  // ...
}
```