ECE2036: (Week 13) Introduction to Vector Templates

In the Standard Template Library, we have many special class templates. In this sample code I use examples for vector templates. In short, we can make dynamic arrays *of any type* using vectors templates. Later we will learn about the use of iterators with these templates.

After including <vector> library, the syntax is:

vector <type of elements in vector> nameOfVector;

The member functions we illustrate here are: size(), empty, push_back(), pop_back(), resize(), and the overloaded operator[].

```
#include <iostream>
#include <vector>
using namespace std;

template <typename T> void printVector(vector<T> &listRef)
{
  if (listRef.empty() == true)
     cout << "The list is empty!!!" << endl;
  else
  {
    for (int i = 0; i < listRef.size(); i++)
      cout << listRef[i] << " " ;
    cout << "\n\n"<< endl;
  }
}

int main()
{
vector <double> values;

system("clear");

cout << "Examples of pushing from the end" << endl;
values.push_back(2.71);
values.push_back(3.14);
values.push_back(1.61);
printVector(values);


cout << "I can print out the size and capacity " << endl;
cout << "Size: " << values.size() << endl;
cout << "Capacity: " << values.capacity() << endl;

cout << "I can pop of the back" << endl;
values.pop_back();
printVector(values);

cout << "I can pop off the back again " << endl;
values.pop_back();
printVector(values);

cout << "Examples of pushing 2.71, 3.14, 1.61" << endl;
values.push_back(2.71);
values.push_back(3.14);
values.push_back(1.61);
printVector(values);

cout << "I can also resize the vector to have 10 elements" << endl;
values.resize(10);
printVector(values);

return 0;}
```