* Chapter 6 STL Containers
  + Container class – classic data structure and ccam hold objects of other classes
  + Iterator – pointer like object, its used to point to objects or instances of container classes. Possible through operator overloading.
* Chapter 8 Vectors
  + Generalization of arrays. Random access DS(indexed)
    - \*d+index+\*length
  + #include <vector>
  + Vector <int> a;
  + Vector <double> d;
    - <double> is a template parameter type casting the vector
  + Syntax for template classes – Templates are only in header files.
    - Template <class T>
    - Class Vector – upper case class.
    - {
      * T \*buffer;
      * Int mysize; - the amount of locations actually used.
      * Int my compacity; - max memory alloted for the array
      * Public:
        + (Methods)
        + Typedef T \*iterator; - (Typedef is a construct from C. WIll rename something old (T\*) for something new (iterator). Iterator then becomes a pointer of type T.
        + T back(); (return value is type T for the template)
    - };
    - Template<class T>
    - T Vector<T>::back()
    - {
      * Assert(mysize > 0);
      * Return buffer[mysize-1]; //Remember that buffer is underlying array
    - }
    - If you want v.back() = 500; to be able to assign values then you have to have & in front of Vector in function title
    - Template<class T>
    - Vector<T>::Vector//Default constructor
    - {
      * Buffer = 0;
      * My\_size= 0;
      * My\_cap = ;
    - }
    - Template <class T>
    - Vector<T>::Vector(unsigned int size)
    - {
      * My\_size= size;
      * My\_cap = size;
      * Buffer = new T [size];
      * For (int I = 0; I<size; I++)
        + Buffer[I] = T(); // constructor for this type.
    - }
    - Methods of this template class
      * Access front(); - returned the value of the first location. == a[0];
      * Back(); - returns the last value of the last location, == a[n];
      * Operator[] - allows us to call different locations or loops
      * \*iter -
        + Vector <int>::iterator iter = a.begin(); allows the iter to point to the beginning of the array. (Note: end() can point to anywhere except the begin location.)
        + Vector<double>::iterator I; can not call anything from the a vector;
      * :: is the scope resolution operator.
      * If you iter++ then it would move from first location to pointing to the second location.
      * Add new elem – push\_back(); insert(); resize(); are all O(n).
    - Example
      * Vector<int> v(10);
      * v.push\_back(60);
        + 60 gets push backed to the back of the array at location v[10]. Increases the size by one loaction
      * For (vector<int>::iterator I =v.begin(); I ! V.end(); I++)
        + Cout << \*I;
      * Same thing – For (int I = 0; I < v.size(); I++)
        + Cout << a[I];
    - Example
      * Vector<int> v;
      * v.reserver(10);
      * v.pushback(60);
      * For ((same as before)
        + 60 would be the output.
    - More functions – O(n)
      * Remove();
      * Erase(iter) - erase the value at location iter and then move the values up one location.
      * Pop\_back(); is constant. O(1). Just have to decrement the size by one.
      * Inclusion test. O(n) or O(log n) if sorted.
    - V[3] = 501; will not check to see if that location belongs to the vector to prevent erros can use v.at(3) = 501; will check to see if the location does belong to you then it assigns the value.