3 Vectors

- Vectors are like arrays that can change size as your program runs
- Vectors, like arrays, have a base type
- Member functions `http://www.cplusplus.com/reference/vector/vector/`
- To declare an empty vector with base type **int**:
  ```
  vector<int> v;
  ```
  - <**int**> identifies vector as **a template class**
  - You can use any base type in a template class:
    ```
    vector<string> v;
    ```
- To use the vector class
- Include the vector library
  ```
  #include <vector>
  ```
- Vector names are placed in the standard namespace so the usual using directive is needed:
  ```
  using namespace std;
  ```

3.1 Accessing vector Elements

- Vectors elements are indexed starting with 0
  - [ ]'s are used to read or change the value of an item:
    ```
    v[i] = 42;
    cout << v[i];
    ```
  - [ ]'s cannot be used to initialize a vector element

3.2 Initializing vector Elements

- Elements are added to a vector using the member function **push_back**
  - **push_back** adds an element in the next available position
  - Example:
    ```
    vector<double> sample;
    sample.push_back(0.0);
    sample.push_back(1.1);
    sample.push_back(2.2);
    ```

3.2.1 Alternate vector Initialization

- A vector constructor exists that takes an integer argument and initializes that number of elements
  - Example:
    ```
    vector<int> v(10);
    ```
    initializes the first 10 elements to 0 **v.size( )** would return 10
    * [ ]'s can now be used to assign elements 0 through 9
    * **push_back** is used to assign elements greater than 9
- A vector constructor exists that initializes the number of elements and initial value.
  - Example:
    ```
    vector<int> v(10,5);
    ```
    initializes the first 10 elements to 5 **v.size( )** would return 10
    * [ ]'s can now be used to assign elements 0 through 9
    * **push_back** is used to assign elements greater than 9

3.3 The size of a vector

- The member function **size** returns the number of elements in a vector
  - Example: To print each element of a vector given the previous vector initialization:
    ```
    for (int i= 0; i < sample.size( ); i++)
            cout << sample[i] << endl;
    ```
- The vector class member function **size** returns an **unsigned int**
  - **unsigned int**'s are nonnegative integers
  - Some compilers will give a warning if the previous for-loop is not changed to:
    ```
    for (unsigned int i= 0; i < sample.size( ); i++)
            cout << sample[i] << endl;
    ```
- Example1
  ```
  #include <iostream>
  #include <iomanip>
  #include <vector>

  int main(int argc, const char * argv[])
  {
      std::vector<int> number;
      for (int i = 1 ; i <= 10 ; i++){
          number.push_back(i);
      }

      for (int i = 0 ; i < number.size() ; i++){
          std::cout << std::setw(3) << number[i];
      }
      std::cout << std::endl;
      return 0;
  }
  ```
  - output

  ```
    1  2  3  4  5  6  7  8  9 10
  ```

- Example2
  ```
  #include <iostream>
  #include <iomanip>
  #include <vector>

  int main(int argc, const char * argv[])
  {
      std::vector<int> number{1,2,3,4,5,6,7,8,9,10};

      for (int i = 0 ; i < number.size() ; i++){
          std::cout << std::setw(3) << number[i];
      }
      std::cout << std::endl;
      return 0;
  }
  ```
  - output

```
  1  2  3  4  5  6  7  8  9 10
```

- Example3

```
#include <iostream>
#include <iomanip>
#include <vector>

int main(int argc, const char * argv[])
{
    std::vector<int> number(10);
    for (int i = 0 ; i < number.size() ; i++){
        number[i] = i + 1;
    }

    for (int i = 0 ; i < number.size() ; i++){
        std::cout << std::setw(3) << number[i];
    }
    std::cout << std::endl;
    return 0;
}
```

  - output

```
  1  2  3  4  5  6  7  8  9 10
```

3.4 Vector Initialization With Classes

- The vector constructor with an integer argument

```
vector<double> values(10);
vector<string> buf(10);
```

- Initializes elements of number types to zero
- Initializes elements of class types using the default constructor for the class
- **sample42.cpp**

```
#include <iostream>
#include <string>
#include <vector>

int main(int argc, const char * argv[])
{
    // input strings
    std::vector<std::string> buf;
    while(1){
        std::cout << "input:";
        std::string s1;
        std::getline(std::cin, s1);
        if (s1.empty()) break;
        buf.push_back(s1);
    }
    // print all
    for (int i = 0 ; i < buf.size() ; i++){
        std::cout << buf[i] << std::endl;
    }
    return 0;
}
```

3.5 vector Issues

- Attempting to use [ ] to set a value beyond the size of a vector may not generate an error
  - The program will probably misbehave
- The assignment operator with vectors does an element by element copy of the right hand vector
  - For class types, the assignment operator must make independent copies

3.6 vector Efficiency

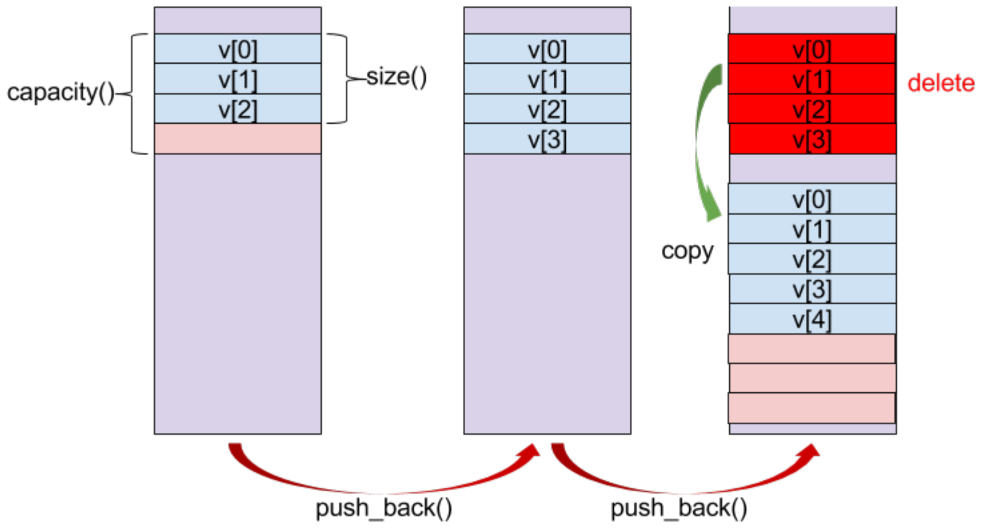
- A vector's **capacity()** is the number of elements allocated in memory
  - Accessible using the **capacity( )** member function
- **size()** is the number of elements initialized
- When a vector runs out of space, **the capacity is automatically increased**
  - A common scheme is to double the size of a vector
    - More efficient than allocating smaller chunks of memory
- **sample43.cpp**

```
#include <iostream>
#include <vector>

int main(int argc, const char * argv[])
{
    const int size = 100;
    std::vector<int> x;
    for (int i = 0 ; i < size ; i++){
        std::cout << "size=" << x.size()
        << " capacity=" << x.capacity()
        << " address=" << x.data() << std::endl;
        x.push_back(i);
    }

    return 0;
}
```

  - Note that **data()** method returns a pointer to the first address of array data block.



3.6.1 Controlling vector Capacity

- When efficiency is an issue
  - Member function reserve can increase the capacity of a vector
    - Example:
      ```
      v.reserve(32); // at least 32 elements
      v.reserve(v.size( ) + 10); // at least 10 more
      ```
  - resize can be used to shrink a vector
    - Example:
      ```
      v.resize(24); //elements beyond 24 are lost
      ```

3.7 Iterator

- Iterator is the abstract data type (ADT) of pointer.
- Declare an iterator
  ```
  std::vector<int>::iterator itr;     // Iterator for type vector<int>
  ```

3.7.1 begin(), end()

- Member function of std::vector, **begin()** returns an iterator to the first element.
- **end()** returns an iterator to the element that one next from the last element.
  ```
      std::vector<int> v{1, 2, 3, 4};
      std::vector<int>::iterator itr = v.begin();   // terator to the first element.
      std::cout << *itr << "\n";      // iterator to the last element
      ++itr;   // move to next element
      *itr = 9;    // change the scond element to 9
  ```

3.7.2 for loop with iterator

```
    std::vector<int> v{1, 2, 3, 4};
    for (std::vector<int>::iterator itr = v.begin() ; itr != v.end() ; itr++)
        {
                // do something with *itr
        }
```

3.8 Remove elements

When deleting the last element,

```
    std::vector<int> v{3, 1, 4, 1, 5};
    v.pop_back();    //  Remove the last element "5"
```

3.8.1 erase()

To remove elements in arbitrary position, you have to use **iterator**.

```
    std::vector<int> v{3, 1, 4, 1, 5};
    v.erase(v.begin() + 1, v.begin() + 3);        // remove 1 and 4
```

To remove specific elements from array,

```
    std::vector<int> v{3, 1, 4, 1, 5};
    for (std::vector<int>::iterator itr = v.begin() ; itr != v.end() ; itr++)
    {
        if (*itr == 1)
        {
                    itr = v.erase(itr);
        }
    }
```

Note that you cannot do this with integer index of array.

3.9 algorithm

http://www.cplusplus.com/reference/algorithm/

Algorithm library provides many useful functions. Here introducing some functions that commonly used to process **std::vector**.

3.9.1 count()

Returns the number of element that has the value specified.

```
    std::vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    std::cout << std::count(v.begin(), v.end(), 5) << "\n";    //  There are 3 "5"
```

3.9.2 find()

Returns the iterator to the first element that has the value specified. If not found returns **end()**

```
    std::vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    //  return iterator to the first element of "5"
    std::vector<int>::iterator itr = std::find(v.begin(), v.end(), 5);
    if( itr != v.end() ) {     // When it found
        // do something
    }
```

3.9.3 sort()

sort elements in ascending order.

```
    std::vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    std::sort(v.begin(), v.end());
```

3.9.4 reverse()

Reverse the order of elements

```
    std::vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    std::reverse(v.begin(), v.end());       // reverse!
    for(auto x : v ) std::cout << x << " ";
    std::cout << "\n";
```

This is equivalent to

```
    std::vector<int> v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    std::reverse(v.begin(), v.end());       // reverse!
    std::vector<int>::iterator itr = v.begin();
    for(int x = *itr ; itr != v.end() ; itr++,x=*itr)
        std::cout << x << " ";
    std::cout << "\n";
```

3.10 Example : sorting class objects

Another version of "**sort**" function takes three inputs. The third input is a pointer to a function that is :

- Binary function that accepts two elements in the range as arguments, and returns a value **bool**.
- The value returned indicates whether the element passed as first argument is considered to go before the second.
- The function shall not modify any of its arguments.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "timeday.h"

// Binary function that accepts two TimeOfDay objects
// and returns a value convertible to bool.
// The value returned indicates whether the first argument is
// earlier than the second.
bool isEarlier (const cosc3000::TimeOfDay& t1, const cosc3000::TimeOfDay& t2)
{
    int min1 = t1.get_hours() * 60 + t1.get_minutes();
    int min2 = t2.get_hours() * 60 + t2.get_minutes();
    if (min1 < min2) return true;
    return false;
}
```

```
int main(int argc, const char * argv[])
{
    // Making List of Time
    std::vector<cosc3000::TimeOfDay> times;
    for (int i = 0 ; i < 10 ; i++){
        cosc3000::TimeOfDay time(rand() % 1440);
        std::cout << time << std::endl;
        times.push_back(time);
    }
    std::cout << "sort\n";
    std::sort(times.begin(),times.end(),isEarlier);
    std::vector<cosc3000::TimeOfDay>::iterator it;
    for (it =  times.begin() ; it != times.end() ; it++){
        std::cout << *it << std::endl;
    }

    return 0;
}
```

  – See **sample44.cpp**