```
// Adopted from M.A. Weiss, DSAAC++ textbook
// by KV, Jan 2020, for CSE 330 lab2

#ifndef VECTOR_H
#define VECTOR_H

#include <cstdlib>  // for swap …
#include <iostream>
#include <cassert>

template <typename T>
class Vector
{
public:
    explicit Vector(int initSize = 0)
        : theSize( initSize ),
          theCapacity( initSize + SPARE_CAPACITY )
    { data = new T[theCapacity]; }

    // added by KV for lab2 ... good to have this one ...
    Vector(int initSize, int initVal)
        :theSize(initSize),
         theCapacity(initSize + SPARE_CAPACITY)
    {
        data = new T[theCapacity];
        for (int i = 0; i < theCapacity; i++)
            data[i] = initVal;
    }

    Vector(const Vector& rhs)
        : theSize( rhs.theSize ),
          theCapacity( rhs.theCapacity ),
          data( nullptr )
    {
        data = new T[theCapacity];
        for (int k = 0; k < theSize; ++k)
            data[k] = rhs.data[k];
    }

    Vector& operator= (const Vector& rhs)
    {
        Vector copy = rhs;
        std::swap(*this, copy);
        return *this;
    }
```

```
~Vector()
{
    delete[] data;
}

Vector(Vector&& rhs)
    : theSize{ rhs.theSize },
      theCapacity{ rhs.theCapacity }, data{ rhs.data }
{
    rhs.data = nullptr;
    rhs.theSize = 0;
    rhs.theCapacity = 0;
}

Vector& operator= (Vector&& rhs)
{
    std::swap(theSize, rhs.theSize);
    std::swap(theCapacity, rhs.theCapacity);
    std::swap(data, rhs.data);

    return *this;
}

bool empty() const
{
    return size() == 0;
}
int size() const
{
    return theSize;
}
int capacity() const
{
    return theCapacity;
}

T& operator[](int index)
{
    assert(index >= 0 && index < theSize);
    return data[index];
}

const T& operator[](int index) const
{
    assert(index >= 0 && index < theSize);
    return data[index];
}
```

```
void resize(int newSize)
{
    if (newSize > theCapacity)
        reserve(newSize * 2);
    theSize = newSize;
}

void reserve(int newCapacity)
{
    if (newCapacity < theSize)
        return;

    T* newArray = new T[newCapacity];
    for (int k = 0; k < theSize; ++k)
        newArray[k] = std::move(data[k]);

    theCapacity = newCapacity;
    std::swap(data, newArray);
    delete[] newArray;
}

void push_back(const T& x)
{
    if (theSize == theCapacity)
        reserve(2 * theCapacity + 1);
    data[theSize++] = x;
}

void push_back(T&& x)
{
    if (theSize == theCapacity)
        reserve(2 * theCapacity + 1);
    data[theSize++] = std::move(x);
}

void pop_back()
{
    assert(theSize >= 1);
    --theSize;
}

const T& back() const

    assert(theSize >= 1);
    return data[theSize - 1];
}
```

```
    // Iterators (new concept)
    typedef T* iterator;
    typedef const T* const_iterator;

    iterator begin()
    {
        return &data[0];
    }
    const_iterator begin() const
    {
        return &data[0];
    }
    iterator end()
    {
        return &data[size()];
    }
    const_iterator end() const
    {
        return &data[size()];
    }

    static const int SPARE_CAPACITY = 2;

private:
    int theSize;
    int theCapacity;
    T* data;
};

#endif
```