11-5 讓 Vec 可以儲存 int 以外的資料型別

目前我們 Vec 雖然可以動態成長，但是只能存放 int 型別的資料，這讓它變得很沒用。

明明只有型別不同，難道我們要寫一個 Vec_int 給 int 用，寫一個 Vec_double 給 double 用，.....。

在這裡我們要使用一個新東西 `樣版(template)`，class 是用來產生物件的 [模版]，而 template 則是用來產生 class 的 [模版]。

你可以這樣想，在宣告時寫 `Vec<int>` 這時 template 就會幫我們生成一個可以儲存 int 的 class，宣告時寫 `Vec<double>` 這時 template 就會幫我們生成一個可以儲存 double 的 class。

在程式碼中插入 `template<typename T>` 表示接下來這個 類別、函數 裡，看到 `T` 時，把它替代成指定的型別。

例如：

```
template<typename T>
class Vec {
private:
  T* m_data;
  int m_size;
  int m_capacity;
};
```

想像在主程式中，我們宣告一個 `Vec<double> a;`，會自動產生一個像這樣的 Vec class。

```
class Vec<double> {
private:
  double* m_data;
  int m_size;
  int m_capacity;
};
```

如此一來，我們就可以讓 Vec 裡儲存各種型別的元素，甚至是我們自己設計的型別。

修改後的 [vec.h]

```
#ifndef VEC_H_INCLUDED
#define VEC_H_INCLUDED

template<typename T>
class Vec {
private:
    T *m_data;
    int m_size;
    int m_capacity;
public:
    Vec();
    ~Vec();
    int size();
    int capacity();
    void push_back(T val);
    void pop_back();
    T& operator[] (int index);
};

template<typename T>
Vec<T>::Vec()
{
    m_data = nullptr;
    m_size = 0;
```

```
    m_capacity = 0;
}

template<typename T>
Vec<T>::~Vec()
{
    if(m_data!=nullptr)
    {
        delete [] m_data;
    }
}

template<typename T>
int Vec<T>::size()
{
    return m_size;
}

template<typename T>
int Vec<T>::capacity()
{
    return m_capacity;
}

template<typename T>
void Vec<T>::push_back(T val)
{
    // 如果空間不夠用
    if(m_size==m_capacity)
    {
        // 每次成長為原來的 2 倍大小
        int new_capacity = m_capacity*2;

        if(new_capacity==0)
            new_capacity = 1;

        // 配置一塊新的空間
        T *new_data = new T[new_capacity];
        // 把資料搬到新的空間
        for(int i=0; i<m_size; i++)
        {
            new_data[i] = m_data[i];
        }
        // 釋放舊的空間，改用新空間
        if(m_data!=nullptr)
            delete [] m_data;

        m_data = new_data;
        m_capacity = new_capacity;
    }

    m_data[m_size] = val;
    m_size++;
}

template<typename T>
void Vec<T>::pop_back()
{
    m_size--;
}

template<typename T>
T& Vec<T>::operator[] (int index)
{
    return m_data[index];
}

#endif // VEC_H_INCLUDED
```

Revision #3
Created 8 June 2026 01:09:43 by huihui
Updated 8 June 2026 11:03:50 by huihui