顺序表实验

1 实现用于模拟 std::vector 类的 simVector

1.1 类结构、构造与析构函数

新建头文件 simVector.h，并输入以下代码。

```
#ifndef simVector_h
#define simVector_h

template <class T>
class simVector{
private:
    int m_size;     //标记当前数据元素个数
    int m_capacity; //标记当前容器容量
    T *m_data;      //用于存储数据元素的数组

public:
    simVector(int n=4);
    ~simVector();

    int size();                 //获取当前元素个数
    int capacity();             //获取当前容器容量
    T& operator[](int id);          //获取位于id位置的元素
    void reserve(int newCapacity);  //扩容
    void insert(int index, T data); //在index位置插入data
    void erase(int index);          //删除index位置的元素
    void push_back(T data);         //在表尾插入数据
};

template <class T>
simVector<T>::simVector(int n) {
    m_size = 0;
    m_capacity = n;
    m_data = new T[m_capacity];
}

template <class T>
simVector<T>::~simVector() {
    delete[] m_data;
}

#endif /* simVector_h */
```

在 main.cpp 文件中输入以下代码

```
#include <iostream>
#include "simVector.h"
using namespace std;
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;
    return 0;
}
```

检查程序是否能够正确编译运行。
🥳 如果能够正确运行，说明 simVector 的声明结构、构造和析构函数正确。可以继续下面的操作。

1.2 元素个数与容量 getter

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```
template <class T>
int simVector<T>::size(){
    return m_size;
}

template <class T>
int simVector<T>::capacity(){
    return m_capacity;
}
```

> **注意：** 在.h文件中添加的所有代码都应该在 #endif 语句之前，即 #endif 必须是头文件的最后一行。

修改 main.cpp 文件：

```
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;
    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;
    return 0;
}
```

编译运行后应该能打印出：

> 测试simVector:
> 元素个数:0

> 容量:4

🥳 如果一切顺利，继续下面的操作。

1.3 插入元素

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```
template <class T>
void simVector<T>::insert(int index, T data) {
    m_size++;
    for(int i=m_size-1; i> index; i--){
        m_data[i] = m_data[i-1];
    }
    m_data[index] = data;
}
```

修改 main.cpp 文件：

```
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;

    a.insert(0, 1);
    a.insert(1, 2);

    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;
    return 0;
}
```

编译并运行程序，应得到以下输出：

> 测试simVector:
> 元素个数:2
> 容量:4

🥳 如果一切顺利，继续下面的操作。

1.4 像数组一样使用 simVector

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```cpp
template<class T>
T& simVector<T>::operator[](int id){
    return m_data[id];
}
```

修改 main.cpp 文件：

```cpp
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;

    a.insert(0, 1);
    a.insert(1, 2);

    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;

    cout<<"表中的元素为：";
    for(int i=0; i<a.size(); ++i){
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

编译并运行程序，应得到以下输出：

> 测试simVector:
> 元素个数:2
> 容量:4
> 表中的元素为：1 2

🥳 如果一切顺利，继续下面的操作。

1.5 扩容

修改 simVector.h 文件中的 insert 函数：

```
template <class T>
void simVector<T>::insert(int index, T data) {
    if(m_size == m_capacity){
        reserve(m_capacity*2);
    }
    m_size++;
    for(int i=m_size-1; i> index; i--){
        m_data[i] = m_data[i-1];
    }
    m_data[index] = data;
}
```

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```
template <class T>
void simVector<T>::reserve(int newCapacity) {
    T* old = m_data;
    m_data = new T[newCapacity];
    for (int i=0; i<m_size; ++i) {
        m_data[i] = old[i];
    }
    m_capacity = newCapacity;
    delete [] old;
}
```

修改 main.cpp 文件：

```
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;

    a.insert(0, 1);
    a.insert(1, 2);
    a.insert(0, 3);
    a.insert(0, 4);
    a.insert(0, 5);

    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;

    cout<<"表中的元素为: ";
    for(int i=0; i<a.size(); ++i){
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

编译并运行程序，应得到以下输出：

> 测试simVector:
> 元素个数:5
> 容量:8
> 表中的元素为：5 4 3 1 2

🥳 如果一切顺利，继续下面的操作。

1.6 删除元素

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```
template <class T>
void simVector<T>::erase(int index){
    for(int i=index; i<m_size-1; ++i){
        m_data[i] = m_data[i+1];
    }
    m_size--;
}
```

修改 main.cpp 文件：

```
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;

    a.insert(0, 1);
    a.insert(1, 2);
    a.insert(0, 3);
    a.insert(0, 4);
    a.insert(0, 5);

    a.erase(2);

    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;

    cout<<"表中的元素为：";
    for(int i=0; i<a.size(); ++i){
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

编译并运行程序，应得到以下输出：

> 测试simVector:
> 元素个数:4
> 容量:8
> 表中的元素为：5 4 1 2

🥳 如果一切顺利，继续下面的操作。

1.7 更方便地向表尾插入元素

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```
template<class T>
void simVector<T>::push_back(T data){
    insert(m_size, data);
}
```

修改 main.cpp 文件：

```
int main(){
    simVector<int> a;
    cout<<"测试simVector:"<<endl;

    a.insert(0, 1);
    a.insert(1, 2);
    a.insert(0, 3);
    a.insert(0, 4);
    a.insert(0, 5);

    a.erase(2);

    a.push_back(10);

    cout<<"元素个数:"<<a.size()<<endl;
    cout<<"容量:"<<a.capacity()<<endl;

    cout<<"表中的元素为：";
    for(int i=0; i<a.size(); ++i){
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

编译并运行程序，应得到以下输出：

> 测试simVector:
> 元素个数:5
> 容量:8
> 表中的元素为：5 4 1 2 10

至此，simVector的实现全部完成！
🥳🎉

2 simVector的使用挑战

2.1 从文件中读入数据到 simVector

将文件 sequential_list_exp.txt 中的数据按顺序读入到一个 simVector<string> 中。

sequential_list_exp.txt 文件的第一行为元素个数，之后的每一行都为一个字符串元素，共20000个元素。
sequential_list_exp.txt 的前几行如下：

```
20000
wkoIqIEA
avHUcm
Cexi
QDtA
GHAaWOvL
……
```

2.2 数据操作

读入数据后，完成以下操作：

- 输出顺序表中元素个数；
- 输出顺序表的容量；
- 输出顺序表从第2022位开始的10个元素；
- 删除顺序表从第2012位开始的5个元素；
- 将自己的名字插入至2024位；
- 再次输出顺序表从第2022位开始的十个元素。

参考输出：

> 元素个数：20000
> 容量：32768

> 从第2022位开始的10个元素：
> ju nvQfYChywU DX QqMWICdsMs gGiQte VROcTz II UUuIaooax IUk qPwfD
> 再次从第2022位开始的10个元素：
> VROcTz II 张三 UUuIaooax IUk qPwfD GZvhWZJQ ttZxWHFlhm htxJcUN YcqXP

3 std::vector 的使用

使用 C++ 标准模板库中的 std::vector 类将实验2的操作重新实现一遍。

注意 std::vector 的插入函数与 simVector 的不同，第一个参数传入的不是下标，而是迭代器。例如，在vector的第5位插入数据0的代码如下：

```
vector<int> a;
...
a.insert(a.begin()+5, 0);

```