

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。

```
1  #ifndef simVector_h
2  #define simVector_h
3
4  template <class T>
5  class simVector{
6  private:
7      int m_size;      //标记当前数据元素个数
8      int m_capacity;  //标记当前容器容量
9      T *m_data;       //用于存储数据元素的数组
10
11 public:
12     simVector(int n=4);
13     ~simVector();
14
15     int size();          //获取当前元素个数
16     int capacity();      //获取当前容器容量
17     T& operator[](int id); //获取位于id位置的元素
18     void reserve(int newCapacity); //扩容
19     void insert(int index, T data); //在index位置插入data
20     void erase(int index);          //删除index位置的元素
21     void push_back(T data);         //在表尾插入数据
22 };
23
24 template <class T>
25 simVector<T>::simVector(int n) {
26     m_size = 0;
27     m_capacity = n;
28     m_data = new T[m_capacity];
29 }
30
31 template <class T>
32 simVector<T>::~~simVector() {
33     delete[] m_data;
34 }
35
36 #endif /* simVector_h */
```

在 main.cpp 文件中输入以下代码

```
1 | #include <iostream>
2 | #include "simVector.h"
3 | using namespace std;
4 | int main(){
5 |     simVector<int> a;
6 |     cout<<"测试simVector:"<<endl;
7 |     return 0;
8 | }
```

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：

```
1 | template <class T>
2 | int simVector<T>::size(){
3 |     return m_size;
4 | }
5 |
6 | template <class T>
7 | int simVector<T>::capacity(){
8 |     return m_capacity;
9 | }
```

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：

```
1 | int main(){
2 |     simVector<int> a;
3 |     cout<<"测试simVector:"<<endl;
4 |     cout<<"元素个数:"<<a.size()<<endl;
5 |     cout<<"容量:"<<a.capacity()<<endl;
6 |     return 0;
7 | }
```

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：

```
1 | template <class T>
2 | void simVector<T>::insert(int index, T data) {
3 |     m_size++;
4 |     for(int i=m_size-1; i> index; i--){
5 |         m_data[i] = m_data[i-1];
6 |     }
7 |     m_data[index] = data;
8 | }
```

修改 `main.cpp` 文件：

```
1 | int main(){
2 |     simVector<int> a;
3 |     cout<<"测试simVector:"<<endl;
4 |
5 |     a.insert(0, 1);
6 |     a.insert(1, 2);
7 |
8 |     cout<<"元素个数:"<<a.size()<<endl;
9 |     cout<<"容量:"<<a.capacity()<<endl;
10 |     return 0;
11 | }
```

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：

```

1 | template<class T>
2 | T& simVector<T>::operator[](int id){
3 |     return m_data[id];
4 | }

```

修改 main.cpp 文件：

```

1 | int main(){
2 |     simVector<int> a;
3 |     cout<<"测试simVector:"<<endl;
4 |
5 |     a.insert(0, 1);
6 |     a.insert(1, 2);
7 |
8 |     cout<<"元素个数:"<<a.size()<<endl;
9 |     cout<<"容量:"<<a.capacity()<<endl;
10 |
11 |     cout<<"表中的元素为： ";
12 |     for(int i=0; i<a.size(); ++i){
13 |         cout<<a[i]<<" ";
14 |     }
15 |     cout<<endl;
16 |     return 0;
17 | }

```

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：

```

1  template <class T>
2  void simVector<T>::insert(int index, T data) {
3      if(m_size == m_capacity){
4          reserve(m_capacity*2);
5      }
6      m_size++;
7      for(int i=m_size-1; i> index; i--){
8          m_data[i] = m_data[i-1];
9      }
10     m_data[index] = data;
11 }

```

在 simVector.h 文件的下方（#endif 之前）添加以下代码：

```

1  template <class T>
2  void simVector<T>::reserve(int newCapacity) {
3      T* old = m_data;
4      m_data = new T[newCapacity];
5      for (int i=0; i<m_size; ++i) {
6          m_data[i] = old[i];
7      }
8      m_capacity = newCapacity;
9      delete [] old;
10 }

```

修改 main.cpp 文件：

```

1  int main(){
2      simVector<int> a;
3      cout<<"测试simVector:"<<endl;
4
5      a.insert(0, 1);
6      a.insert(1, 2);
7      a.insert(0, 3);
8      a.insert(0, 4);
9      a.insert(0, 5);
10
11     cout<<"元素个数:"<<a.size()<<endl;
12     cout<<"容量:"<<a.capacity()<<endl;
13
14     cout<<"表中的元素为： ";
15     for(int i=0; i<a.size(); ++i){
16         cout<<a[i]<<" ";
17     }
18     cout<<endl;
19     return 0;
20 }

```

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：

```
1  template <class T>
2  void simVector<T>::erase(int index){
3      for(int i=index; i<m_size-1; ++i){
4          m_data[i] = m_data[i+1];
5      }
6      m_size--;
7  }
```

修改 `main.cpp` 文件：

```
1  int main(){
2      simVector<int> a;
3      cout<<"测试simVector:"<<endl;
4
5      a.insert(0, 1);
6      a.insert(1, 2);
7      a.insert(0, 3);
8      a.insert(0, 4);
9      a.insert(0, 5);
10
11     a.erase(2);
12
13     cout<<"元素个数:"<<a.size()<<endl;
14     cout<<"容量:"<<a.capacity()<<endl;
15
16     cout<<"表中的元素为： ";
17     for(int i=0; i<a.size(); ++i){
18         cout<<a[i]<<" ";
19     }
20     cout<<endl;
21     return 0;
22 }
```

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：

```
1  template<class T>
2  void simVector<T>::push_back(T data){
3      insert(m_size, data);
4  }
```

修改 `main.cpp` 文件：

```
1  int main(){
2      simVector<int> a;
3      cout<<"测试simVector:"<<endl;
4
5      a.insert(0, 1);
6      a.insert(1, 2);
7      a.insert(0, 3);
8      a.insert(0, 4);
9      a.insert(0, 5);
10
11     a.erase(2);
12
13     a.push_back(10);
14
15     cout<<"元素个数:"<<a.size()<<endl;
16     cout<<"容量:"<<a.capacity()<<endl;
17
18     cout<<"表中的元素为： ";
19     for(int i=0; i<a.size(); ++i){
20         cout<<a[i]<<" ";
21     }
22     cout<<endl;
23     return 0;
24 }
```

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，之后的每一行都为字符串元素，共20000个元素。

sequential_list_exp.txt 的前几行如下：

1	20000
2	wkoIqIEA
3	avHUcm
4	Cexi
5	QDtA
6	GHAaW0vL
7

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 UUulaoaoax IUk qPwfD

再次从第2022位开始的10个元素：

VROcTz II 张三 UUulaoaoax IUk qPwfD GZvhWZJQ ttZxWHFIhm 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：

```
1 | vector<int> a;  
2 | ...  
3 | a.insert(a.begin()+5, 0);  
4 |
```