

1. 动态数组补充

```
#include <utility>
using namespace std;

int main()
{
    initializer_list<int> il = { 1, 3, 5, 7, 9, 2, 3, 2 };
    il.size();
    for (auto val : il)
    {
        cout << val << endl;
    }
}
```

Figure 1-1

收初始化列表,但是只支持相同类型

1.1 CVector.h

```
1  #pragma once
2
3  #include <assert.h>
4  #include <memory>
5  #include <utility>
6
7  // 动态数组类
8  template<typename T>
9  class CVector
10 {
11 public:
12     CVector();
13     CVector(std::initializer_list<T> il);
14     CVector(size_t nSize);
15     CVector(const CVector& obj);
16     CVector(CVector&& obj);
17     CVector& operator=(const CVector& obj);
18     virtual ~CVector();
19
20     void PushHead(const T& val); // 头插
21     void PushTail(const T& val); // 尾插
22     void Insert(size_t nIdx, const T& val); // 指定位置插入
23
24     void PopHead(); // 头删
25     void PopTail(); // 尾删
26     void Delete(size_t nIdx); // 指定位置删除
```

```

27
28     T& operator[](size_t nIndex); // 修改
29
30     int Find(size_t val) const; // 查询, 返回下标
31
32     void Sort(); // 排序, 默认从小到大
33
34     bool IsEmpty() const; // 是否为空
35
36     size_t GetCount() const; // 获取元素个数
37
38     void Clear(); // 清空
39 private:
40     void Reset();
41
42 private:
43     T* m_pBuff; // 存储元素的缓冲区
44     size_t m_nBuffLen; // 缓冲区大小
45     size_t m_nCount; // 类型T的元素个数
46 };
47
48 template<typename T>
49 void CVector<T>::Reset()
50 {
51     m_pBuff = nullptr;
52     m_nBuffLen = 0 * sizeof(T) * m_nCount;
53     m_nCount = 0;
54 }
55
56 template<typename T>
57 CVector<T>::CVector()
58 {
59     Reset();
60 }
61 template<typename T>
62 CVector<T>::CVector(std::initializer_list<T> il)
63 {
64     Reset();
65     for (auto val : il) // 调拷贝构造
66     {
67         PushTail(val);
68     }
69 }
70
71 template<typename T>
72 CVector<T>::CVector(size_t nSize) // 多少个T。也就是Count
73 {
74     m_pBuff = new T[nSize];
75     if (m_pBuff == nullptr)
76     {
77         return;
78     }
79     m_nBuffLen = nSize * sizeof(T) * m_nCount;
80     m_nCount = nSize;
81 }
82
83 template<typename T>
84 CVector<T>::CVector(const CVector& obj)

```

```

85 {
86     *this = obj; // 调用=运算符重载
87 }
88
89 template<typename T>
90 CVector<T>::CVector(CVector<T>&& obj)
91 {
92     m_pBuff = obj.m_pBuff;
93     m_nBuffLen = obj.m_nBuffLen;
94     m_nCount = m_nCount;
95
96     obj.Reset();
97 }
98
99 template<typename T>
100 CVector<T>& CVector<T>::operator=(const CVector<T>& obj)
101 {
102     if (*this != obj)
103     {
104         if (obj.IsEmpty())
105         {
106             Clear();
107             return *this;
108         }
109         else
110         {
111             if (m_nBuffLen < obj.m_nBuffLen)
112             {
113                 T* pNewBuff = nullptr;
114                 pNewBuff = new T[obj.m_nCount];
115                 if (pNewBuff != nullptr)
116                 {
117                     return *this;
118                 }
119                 // memcpy(pNewBuff, obj.m_pBuff, obj.m_nBuffLen * sizeof(T));
120                 for (size_t i = 0; i < obj.m_nCount; i++)
121                 {
122                     m_pBuff[i] = obj.m_pBuff[i];
123                 }
124                 m_pBuff = pNewBuff;
125                 m_nBuffLen = obj.m_nBuffLen;
126                 m_nCount = obj.m_nCount;
127             }
128             else
129             {
130                 // memcpy(m_pBuff, obj.m_pBuff, obj.m_nBuffLen * sizeof(T));
131                 for (size_t i = 0; i < obj.m_nCount; i++)
132                 {
133                     m_pBuff[i] = obj.m_pBuff[i];
134                 }
135                 m_nBuffLen = obj.m_nBuffLen;
136                 m_nCount = obj.m_nCount;
137             }
138         }
139     }
140     return *this;
141 }
142

```

```

143     template<typename T>
144     CVector<T>::~~CVector()
145     {
146         Clear();
147     }
148
149     template<typename T>
150     void CVector<T>::PushHeard(const T& val)
151     {
152         Insert(0, val);
153     }
154
155     template<typename T>
156     void CVector<T>::PushTail(const T& val)
157     {
158         Insert(m_nCount, val);
159     }
160
161     template<typename T>
162     void CVector<T>::Insert(size_t nIdx, const T& val)
163     {
164         // 检查
165         assert(nIdx ≤ m_nCount);
166         // 判断内存是否为空
167         if (m_pBuff == nullptr)
168         {
169             m_pBuff = new T(val);
170             m_nBuffLen = 1 * sizeof(T) * m_nCount;
171             m_nCount = 1;
172             return;
173         }
174
175         size_t nNewLen = m_nCount * 2; // 只是两倍Count的空间
176         T* pNewBuff = new T[nNewLen];
177         if (pNewBuff == nullptr)
178         {
179             return;
180         }
181         // 拷贝原来的数据
182         // memcpy(pNewBuff, m_pBuff, m_nBuffLen * sizeof(T)); // 浅拷贝
183         for (size_t i = 0; i < m_nCount; i++)
184         {
185             pNewBuff[i] = m_pBuff[i]; // 调用=运算符重载
186         }
187         // 删除原来的
188         m_nCount == 1 ? delete m_pBuff : delete[] m_pBuff;
189         m_pBuff = pNewBuff;
190         m_nBuffLen = nNewLen;
191
192         // 移动数据
193         // memcpy(m_pBuff + nIdx + 1, m_pBuff + nIdx, (m_nCount - nIdx) * sizeof(T));
194         for (size_t i = m_nCount; i > nIdx; --i)
195         {
196             m_pBuff[i] = m_pBuff[i - 1];
197         }
198         // 数据赋值
199         m_pBuff[nIdx] = val;
200         // 更新元素个数

```

```

201     m_nCount++;
202 }
203
204 template<typename T>
205 void CVector<T>::PopHead()
206 {
207     Delete(0);
208 }
209
210 template<typename T>
211 void CVector<T>::PopTail()
212 {
213     Delete(m_nCount - 1);
214 }
215
216 template<typename T>
217 void CVector<T>::Delete(size_t nIdx)
218 {
219     assert(nIdx < m_nCount);
220     if (m_pBuff == nullptr)
221     {
222         return;
223     }
224     else
225     {
226         //memcpy(m_pBuff + nIdx, m_pBuff + nIdx + 1, (m_nBuffLen - nIdx) *
sizeof(T));
227         // m_nBuffLen -= sizeof(T);容量不应该减少
228         for (size_t i = nIdx; i < m_nCount - 1; i++)
229         {
230             m_pBuff[i] = m_pBuff[i + 1];
231         }
232         m_nCount--;
233     }
234 }
235
236 template<typename T>
237 T& CVector<T>::operator[](size_t nIdx)
238 {
239     assert(nIdx < m_nCount);
240     return m_pBuff[nIdx];
241 }
242
243 template<typename T>
244 int CVector<T>::Find(size_t val) const
245 {
246     for (size_t i = 0; i < m_nCount; i++)
247     {
248         if (m_pBuff[i] == val)
249         {
250             return i;
251         }
252     }
253     return -1;
254 }
255
256 template<typename T>
257 void CVector<T>::Sort()

```

```

258 {
259 }
260
261 template<typename T>
262 bool CVector<T>::IsEmpty() const
263 {
264     return m_nCount == 0;
265 }
266
267 template<typename T>
268 size_t CVector<T>::GetCount() const
269 {
270     return m_nCount;
271 }
272
273 template<typename T>
274 void CVector<T>::Clear()
275 {
276     if (m_pBuff != nullptr)
277     {
278         m_nCount = 1 ? delete m_pBuff : delete[] m_pBuff;
279     }
280     Reset();
281 }

```

Fence 1-1

1.2 动态数组.cpp

```

1 // 动态数组.cpp
2
3 #include <iostream>
4 #include "CVector.h"
5 using namespace std;
6
7 class CA
8 {
9 public:
10     CA() :m_p(nullptr) {}
11     CA(int n) :m_p(new int(n)) {}
12     CA(const CA& obj) :m_p(new int(*obj.m_p)) {}
13     CA& operator=(const CA& obj)
14     {
15         if (m_p != nullptr)
16         {
17             delete m_p;
18         }
19         m_p = new int(*obj.m_p);
20         return *this;
21     }
22     ~CA()
23     {
24         if (m_p != nullptr)
25         {
26             delete m_p;
27         }
28     }
29 private:
30     int* m_p;

```

```

31 };
32
33 int main()
34 {
35     // 动态数组,元素是CA类型的对象
36     // 先创建3个对象的一个C++自带的动态数组,
37     // 再迭代器这个动态数组, 每个对象尾插到自己的动态数组vet
38
39     CVector<CA> vet({ CA(4), CA(6), CA{8} });
40
41     vet.PopHead();
42
43
44     CVector<int> vec;
45     vec.Insert(0, 4);
46     vec.Insert(1, 5);
47     // vec.Insert(5, 7);
48     vec.Insert(2, 9);
49     vec.Insert(1, 2);
50     vec.Insert(3, 5);
51
52     vec.PopHead();
53     vec.Delete(2);
54
55     vec[1] = 7;
56     int nRes = vec.Find(7);
57
58     return 0;
59 }

```

Fence 1-2

2. STL中的Vector、迭代器

```

1 // STL中的Vector.cpp
2
3 #include <iostream>
4 #include <vector>
5 using namespace std;
6
7 int main()
8 {
9     vector<int> vct({6, 9, 8, 2, 41, 45, 65});
10    vct.push_back(4);
11    vct.push_back(2);
12
13    vct[2] = 8;
14
15    vct.front();
16    vct.back();
17
18    vct.pop_back();
19    vct.pop_back();
20
21    for (auto val : vct)
22    {
23        cout << val << endl;
24    }

```

