**计算机科学与工程学院实验报告5**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验课程名称** | | **实验五 基于STL数据结构及应用**  **——STL的一维向量类vector** | | | **实验成绩** |  |
| **专业** | **计算机科学与技术** | | **班级** | **\*** | **指导教师签字** |  |
| **姓名** | **\*** | | **姓名** | **\*** | **姓名** |  |
| **学号** | **\*** | | **学号** | **\*** | **学号** |  |
| **附录实验程序及结果：**  **实验程序：**        **结果：**    **源代码：**  namespace **pxl**  {       template<class **T**>       class **vector**       {         public:              typedef **T**\* **iterator**;            typedef const **T**\* **const\_iterator**;      **vector**()                 :**\_start**(nullptr)                 , **\_finish**(nullptr)                 , **\_endofstorage**(nullptr)            {}    *//传统写法的拷贝构造*  */\*   vector(const vector<T>& v)*  *{*  *\_start = new T[v.capacity()];*  *\_finish = \_start + v.size();*  *\_endofstorage = \_start + v.capacity();*  *memcpy(\_start, v.\_start, v.size() \* sizeof(T));*  *//这里的memcpy和reserve一样*  *}\*/*    *//迭代器区间构造*  *//一个类模板的成员函数，又可以是一个函数模板*            template <class **InputIterator**> *//为什么叫inputiterator,迭代器区间的引擎规范*  *//函数模板的模板参数要传送迭代器区间，存在命名规范*  *//input\_ 只写迭代器    output\_ iterator只读迭代器 forward\_ 单向迭代器（单链表）   bidirectional\_双向迭代器*  *//randomaccess\_ 随机迭代器*    **vector**(**InputIterator** first, **InputIterator** last)                 :**\_start**(nullptr)                 , **\_finish**(nullptr)                 , **\_endofstorage**(nullptr)            {                 while (first != last)                 {  **push\_back**(\*first);                      ++first;                 }            }              void **swap**(**vector**<**T**>& v)            {  **std**::**swap**(\_start, v.\_start);  **std**::**swap**(\_finish, v.\_start);  **std**::**swap**(\_endofstorage, v.\_endofstorage);            }              现代写法的拷贝构造  **vector**(const **vector**<**T**>& v)                 :**\_start**(nullptr)                 , **\_finish**(nullptr)                 , **\_endofstorage**(nullptr)            {                 vector<T> **tmp**(v.**begin**(), v.**end**());    *//this->swap(tmp);*  **swap**(tmp);              }    *//opreator*  **vector**<**T**>& operator = (**vector**<**T**> v)            {  **swap**(v);                 return \*this;            }      **~vector**()            {                 if (\_start)                 {                      delete[] \_start;                     \_start = \_finish = \_endofstorage = nullptr;                 }            }    **iterator** **begin**()            {                 return \_start;            }  **iterator** **end**()            {                 return \_finish;            }    **const\_iterator** **begin**() const            {                 return \_start;            }  **const\_iterator** **end**() const            {                 return \_finish;            }                const **T**& operator[](size\_t i) const            {  **assert**(i < **size**());                      return \_start[i];            }    **T**& operator[](size\_t i)            {  **assert**(i < **size**());                      return \_start[i];            }              size\_t **size**() const            {                 return \_finish - \_start;            }              size\_t **capacity**() const            {                 return \_endofstorage - \_start;            }              void **push\_back**(const **T**& x)            {                 if (\_finish == \_endofstorage)                 {  */\*   size\_t newCapacity =( capacity() == 0 ? 4 : capacity() \* 2);*  *size\_t sz = size();*  *T\* tmp = new T[newCapacity];*  *if (\_start)*  *{*  *memcpy(tmp, \_start, sizeof(T) \* size());*  *delete[] \_start;*  *}*  *\_start = tmp;*  *\_finish = \_start + sz;*  *\_endofstorage = \_start + newCapacity;\*/*    **reserve**(**capacity**() == 0 ? 4 : **capacity**() \* 2);                 }                 \*\_finish = x;                 ++\_finish;            }              void **reserve**(size\_t n)            {                 if (n > **capacity**())                 {                      size\_t sz = **size**();                      T\* tmp = new T[n];                      if (\_start)                      {  *//使memcpy对自定义类型又浅拷贝的问题*  *//memcpy(tmp, \_start, sizeof(T) \* size());*                             for (size\_t i = 0; i < sz; i++)                           {  *//T是int,一个一个拷贝没问题*  *//T是自定义类型 ，一个一个拷贝调用的是T的深拷贝赋值=*                                tmp[i] = \_start[i];                           }                           delete[] \_start;                      }                      \_start = tmp;                      \_finish = \_start + sz;                      \_endofstorage = \_start + n;                 }            }    *//const引用，延长匿名对象的生命周期,匿名对象传缺省值，T这个类型生成的匿名对象，*  *//模板出现以后可以认为内置类型也有构造函数*            void **resize**(size\_t n, const **T**& val = **T**())            {                 if (n < **size**())                 {                      \_finish = \_start + n;                 }                 else                 {                      if (n > **capacity**())                      {  **reserve**(n);                      }                        while (\_finish != \_start + n)                      {                           \*\_finish = val;                           ++\_finish;                      }                 }              }              void **pop\_back**()            {  **assert**(\_finish > \_start);                 --\_finish;            }    *//vector的迭代器失效主要发生在insert和erase*  *//只要使用迭代器访问访问的容器，都可能迭代器失效*  *//string失效的场景与vector完全一样，但是string很少失效，因为它的插入与删除使用下标*  **iterator** **insert**(**iterator** pos, const **T**& x)            {  **assert**(pos >= \_start);  **assert**(pos <= \_finish); *//=是为了尾插*  *//满了就扩容*                 if (\_finish == \_endofstorage)                 {  *//扩容会导致pos失效，扩容需要更新一下pos*                      size\_t len = pos - \_start;  **reserve**(**capacity**() == 0 ? 4 : capacity \* 2);                      pos = \_start + len;                 }                   iterator end = \_finish - 1;                 while (end >= pos)                 {                      \*(end + 1) = \*(end);                      --end;                 }                 \*pos = x;                 ++\_finish;                   return pos;            }    **iterator** **erase**(**iterator** pos)            {  **assert**(pos >= \_start);  **assert**(pos < \_finish);                 iterator begin = pos + 1;                 while (begin < \_finish)                 {                      \*(begin - 1) = \*begin;                      ++begin;                 }                 --\_finish;                   return pos;            }           private:            iterator \_start;            iterator \_finish;            iterator \_endofstorage;           };           void **test**()       {  **vector**<int> v;            v.**push\_back**(1);            v.**push\_back**(2);            v.**push\_back**(3);            v.**push\_back**(4);            v.**push\_back**(5);  **vector**<int> **v2**(v);              for (**size\_t** i = 0; i < v.**size**(); i++)            {                 cout << v[i];            }            cout << endl;            for (auto e : v)            {                 cout << e;            }       }  }  #include<iostream>  #include<vector>  using namespace std;  int main()  {  vector<int> v(2, 1);  for (auto x : v)  {  cout << x << " ";  }  cout << endl;    v.push\_back(3);  v.push\_back(3);  v.push\_back(3);  v.push\_back(3);  v.push\_back(3);  for (auto x : v)  {  cout << x << " ";  }  cout << endl;    return 0;  } | | | | | | |
| **实验课程总结**  通过本次实验课程内容的学习，我了解了stl vector的相关知识。主要学习的是如何实现vector以及vector的一些函数应用。本次实验内容提升了我对vector和stl相关知识的了解，加强了有关的应用。 | | | | | | |

**批改时间：**