**目 录**

1 STL 简介

2 顺序性容器

2.1 C++ VECTOR（向量容器）

2.2 C++ LIST（双向链表）

2.3 C++ DEQUE(双向队列)

2.4 三者比较

3 关联容器

3.1 特点

3.2 C++ SETS & MULTISETS

3.3 C++ MAPS & MULTIMAPS

4 容器适配器

4.1 特点

4.2 C++ STACKS（堆栈）

4.3 C++ QUEUES(队列)

4.4 C++ PRIORITY QUEUES(优先队列)

5 迭代器

5.1 解释

5.2 功能特点

6 C++标准库总结

6.1 容器

6.2 算法

6.3 函数对象

6.4 迭代器

6.5 分配器

6.6 数值

1 STL 简介

<http://www.cplusplus.com/reference/stl/更加详细的资料>

C++ STL (Standard Template Library标准模板库) 是通用类模板和算法的集 合，它提供给程序员一些标准的数据结构的实现如 queues(队列), lists(链表), 和 stacks(栈)等.

C++ STL 提供给程序员以下三类数据结构的实现： 标准容器类

顺序性容器

vector 从后面快速的插入与删除，直接访问任何元素

deque 从前面或后面快速的插入与删除，直接访问任何元素

list 双链表，从任何地方快速插入与删除

关联容器

set 快速查找，不允许重复值

multiset 快速查找，允许重复值

map 一对多映射，基于关键字快速查找，不允许重复值

multimap 一对多映射，基于关键字快速查找，允许重复值

容器适配器

stack 后进先出

queue 先进先出

priority\_queue 最高优先级元素总是第一个出列

程序员使用复杂数据结构的最困难的部分已经由STL完成. 如果程序员想使用包 含int数据的stack, 他只要写出如下的代码:

stack<int> myStack;

接下来, 他只要简单的调用 push() 和 pop() 函数来操作栈. 借助 C++ 模板的 威力, 他可以指定任何的数据类型，不仅仅是int类型. STL stack实现了栈的功 能，而不管容纳的是什么数据类型.

2 顺序性容器

2.1 C++ Vector（向量容器）

是一个线性顺序结构。相当于数组，但其大小可以不预先指定，并且自动扩 展。它可以像数组一样被操作，由于它的特性我们完全可以将vector 看作动态数 组。

在创建一个vector 后，它会自动在内存中分配一块连续的内存空间进行数据存储，初始的空间大小可以预先指定也可以由vector 默认指定，这个大小即 capacity （）函数的返回值。当存储的数据超过分配的空间时vector 会重新分配 一块内存块，但这样的分配是很耗时的，在重新分配空间时它会做这样的动作：

首先，vector 会申请一块更大的内存块；

然后，将原来的数据拷贝到新的内存块中；

其次，销毁掉原内存块中的对象（调用对象的析构函数）；

最后，将原来的内存空间释放掉。

如果vector 保存的数据量很大时，这样的操作一定会导致糟糕的性能（这也 是vector 被设计成比较容易拷贝的值类型的原因）。所以说vector 不是在什么情 况下性能都好，只有在预先知道它大小的情况下vector 的性能才是最优的。

vector 的特点：

1. 指定一块如同数组一样的连续存储，但空间可以动态扩展。即它可以像数组 一样操作，并且可以进行动态操作。通常体现在push\_back() pop\_back() 。
2. 随机访问方便，它像数组一样被访问，即支持[ ] 操作符和vector.at()
3. 节省空间，因为它是连续存储，在存储数据的区域都是没有被浪费的，但是 要明确一点vector 大多情况下并不是满存的，在未存储的区域实际是浪费的。
4. 在内部进行插入、删除操作效率非常低，这样的操作基本上是被禁止的。 Vector 被设计成只能在后端进行追加和删除操作，其原因是vector 内部的实现 是按照顺序表的原理。
5. 只能在vector 的最后进行push 和pop ，不能在vector 的头进行push 和pop 。
6. 当动态添加的数据超过vector 默认分配的大小时要进行内存的重新分配、拷 贝与释放，这个操作非常消耗性能。 所以要vector 达到最优的性能，最好在创 建vector 时就指定其空间大小。

Vectors 包含着一系列连续存储的元素,其行为和数组类似。访问Vector中的 任意元素或从末尾添加元素都可以在常量级时间复杂度内完成，而查找特定值的 元素所处的位置或是在Vector中插入元素则是线性时间复杂度。

1. Constructors 构造函数

vector<int> v1; //构造一个空的

vector vector<int> v1( 5, 42 ); //构造了一个包含5个值为42的元素的Vector

1. Operators 对vector进行赋值或比较

C++ Vectors能够使用标准运算符: ==, !=, <=, >=, <, 和 >.

要访问vector中的某特定位置的元素可以使用 [] 操作符.

两个vectors被认为是相等的,如果:

1. 它们具有相同的容量
2. 所有相同位置的元素相等.

vectors之间大小的比较是按照词典规则.

3.assign() 对Vector中的元素赋值

语法：

void assign( input\_iterator start, input\_iterator end );

// 将区间[start, end)的元素赋到当前

vector void assign( size\_type num, const TYPE &val );

// 赋num个值为val的元素到vector中,这个函数将会清除掉为vector赋值以前的内容.

1. at() 返回指定位置的元素

语法：

TYPE at( size\_type loc );//差不多等同v[i];但比v[i]安全; 5.back() 返回最末一个元素

1. begin() 返回第一个元素的迭代器
2. capacity() 返回vector所能容纳的元素数量(在不重新分配内存的情况下）
3. clear() 清空所有元素
4. empty() 判断Vector是否为空（返回true时为空）
5. end() 返回最末元素的迭代器(译注:实指向最末元素的下一个位置)
6. erase() 删除指定元素 语法： iterator erase( iterator loc );//删除loc处的元素 iterator erase( iterator start, iterator end );//删除start和end之间的元素
7. front() 返回第一个元素的引用
8. get\_allocator() 返回vector的内存分配器
9. insert() 插入元素到Vector中 语法： iterator insert( iterator loc, const TYPE &val ); //在指定位置loc前插入值为val的元素,返回指向这个元素的迭代器, void insert( iterator loc, size\_type num, const TYPE &val ); //在指定位置loc前插入num个值为val的元素 void insert( iterator loc, input\_iterator start, input\_iterator end ); //在指定位置loc前插入区间[start, end)的所有元素
10. max\_size() 返回Vector所能容纳元素的最大数量（上限）
11. pop\_back() 移除最后一个元素
12. push\_back() 在Vector最后添加一个元素
13. rbegin() 返回Vector尾部的逆迭代器
14. rend() 返回Vector起始的逆迭代器
15. reserve() 设置Vector最小的元素容纳数量 //为当前vector预留至少共容纳size个元素的空间
16. resize() 改变Vector元素数量的大小 语法: void resize( size\_type size, TYPE val ); //改变当前vector的大小为size,且对新创建的元素赋值val
17. size() 返回Vector元素数量的大小
18. swap() 交换两个Vector 语法： void swap( vector &from );

2.2 C++ List（双向链表）

是一个线性链表结构，它的数据由若干个节点构成，每一个节点都包括一个 信息块（即实际存储的数据）、一个前驱指针和一个后驱指针。它无需分配指定 的内存大小且可以任意伸缩，这是因为它存储在非连续的内存空间中，并且由指针将有序的元素链接起来。 由于其结构的原因，list 随机检索的性能非常的不好，因为它不像vector 那 样直接找到元素的地址，而是要从头一个一个的顺序查找，这样目标元素越靠后， 它的检索时间就越长。检索时间与目标元素的位置成正比。 虽然随机检索的速度不够快，但是它可以迅速地在任何节点进行插入和删除 操作。因为list 的每个节点保存着它在链表中的位置，插入或删除一个元素仅对 最多三个元素有所影响，不像vector 会对操作点之后的所有元素的存储地址都有 所影响，这一点是vector 不可比拟的。

list 的特点：

(1) 不使用连续的内存空间这样可以随意地进行动态操作；

(2) 可以在内部任何位置快速地插入或删除，当然也可以在两端进行push和pop 。

(3) 不能进行内部的随机访问，即不支持[ ] 操作符和vector.at() ；

Lists将元素按顺序储存在链表中，与向量(vectors)相比，它允许快速的插入 和删除，但是随机访问却比较慢.

1.assign() 给list赋值 语法: void assign( input\_iterator start, input\_iterator end ); //以迭代器start和end指示的范围为list赋值 void assign( size\_type num, const TYPE &val ); //赋值num个以val为值的元素。

2.back() 返回最后一个元素的引用

3.begin() 返回指向第一个元素的迭代器

4.clear() 删除所有元素

5.empty() 如果list是空的则返回true

1. end() 返回末尾的迭代器

7.erase() 删除一个元素 语法： iterator erase( iterator loc );//删除loc处的元素 iterator erase( iterator start, iterator end ); //删除start和end之间的元素

1. front() 返回第一个元素的引用

9.get\_allocator() 返回list的配置器

10.insert() 插入一个元素到list中 语法： iterator insert( iterator loc, const TYPE &val ); //在指定位置loc前插入值为val的元素,返回指向这个元素的迭代器, void insert( iterator loc, size\_type num, const TYPE &val ); //定位置loc前插入num个值为val的元素 void insert( iterator loc, input\_iterator start, input\_iterator end ); //在指定位置loc前插入区间[start, end)的所有元素

1. max\_size() 返回list能容纳的最大元素数量

12.merge() 合并两个list 语法:

void merge( list &lst );//把自己和lst链表连接在一起 void merge( list &lst, Comp compfunction ); //指定compfunction，则将指定函数作为比较的依据。

1. pop\_back() 删除最后一个元素
2. pop\_front() 删除第一个元素
3. push\_back() 在list的末尾添加一个元素
4. push\_front() 在list的头部添加一个元素
5. rbegin() 返回指向第一个元素的逆向迭代器
6. remove() 从list删除元素 语法: void remove( const TYPE &val ); //删除链表中所有值为val的元素
7. remove\_if() 按指定条件删除元素
8. rend() 指向list末尾的逆向迭代器
9. resize() 改变list的大小 语法: void resize( size\_type num, TYPE val ); //把list的大小改变到num。被加入的多余的元素都被赋值为val22.
10. reverse() 把list的元素倒转
11. size() 返回list中的元素个数
12. sort() 给list排序 语法: void sort();//为链表排序，默认是升序 void sort( Comp compfunction );//采用指定函数compfunction来判定两个元素的大小。
13. splice() 合并两个list 语法: void splice( iterator pos, list &lst );//把lst连接到pos的位置 void splice( iterator pos, list &lst, iterator del );//插入lst中del所指元素到现链表的pos上 void splice( iterator pos, list &lst, iterator start, iterator end );//用start和end指定范围。
14. swap() 交换两个list 语法： void swap( list &lst );// 交换lst和现链表中的元素

27.unique() 删除list中重复的元素 语法: void unique();//删除链表中所有重复的元素 void unique( BinPred pr );// 指定pr，则使用pr来判定是否删除。

2.3 C++ Deque(双向队列)

是一种优化了的、对序列两端元素进行添加和删除操作的基本序列容器。它 允许较为快速地随机访问，但它不像vector 把所有的对象保存在一块连续的内存 块，而是采用多个连续的存储块，并且在一个映射结构中保存对这些块及其顺序 的跟踪。向deque 两端添加或删除元素的开销很小。它不需要重新分配空间，所

以向末端增加元素比vector 更有效。 实际上，deque 是对vector 和list 优缺点的结合，它是处于两者之间的一种 容器。

deque 的特点：

(1) 随机访问方便，即支持[ ] 操作符和vector.at() ，但性能没有vector 好；

1. 可以在内部进行插入和删除操作，但性能不及list ；

(3) 可以在两端进行push 、pop ；

(4) 相对于verctor 占用更多的内存。

双向队列和向量很相似，但是它允许在容器头部快速插入和删除（就像在尾部一 样）。

1.Constructors 创建一个新双向队列 语法: deque();//创建一个空双向队列 deque( size\_type size );// 创建一个大小为size的双向队列 deque( size\_type num, const TYPE &val ); //放置num个val的拷贝到队列中 deque( const deque &from );// 从from创建一个内容一样的双向队列 deque( input\_iterator start, input\_iterator end ); // start 和 end - 创建一个队列，保存从start到end的元素。

1. Operators 比较和赋值双向队列 //可以使用[]操作符访问双向队列中单个的元素
2. assign() 设置双向队列的值 语法: void assign( input\_iterator start, input\_iterator end); //start和end指示的范围为双向队列赋值 void assign( Size num, const TYPE &val );//设置成num个val。
3. at() 返回指定的元素 语法: reference at( size\_type pos ); 返回一个引用，指向双向队列中位置pos上的元素
4. back() 返回最后一个元素 语法: reference back();//返回一个引用，指向双向队列中最后一个元素
5. begin() 返回指向第一个元素的迭代器 语法: iterator begin();//返回一个迭代器，指向双向队列的第一个元素
6. clear() 删除所有元素
7. empty() 返回真如果双向队列为空
8. end() 返回指向尾部的迭代器
9. erase() 删除一个元素 语法: iterator erase( iterator pos ); //删除pos位置上的元素 iterator erase( iterator start, iterator end ); //删除start和end之间的所有元素 //返回指向被删除元素的后一个元素

11.front() 返回第一个元素的引用

1. get\_allocator() 返回双向队列的配置器
2. insert() 插入一个元素到双向队列中 语法: iterator insert( iterator pos, size\_type num, const TYPE &val ); //pos前插入num个val值 void insert( iterator pos, input\_iterator start, input\_iterator end ); //插入从start到end范围内的元素到pos前面
3. max\_size() 返回双向队列能容纳的最大元素个数
4. pop\_back() 删除尾部的元素
5. pop\_front() 删除头部的元素
6. push\_back() 在尾部加入一个元素
7. push\_front() 在头部加入一个元素
8. rbegin() 返回指向尾部的逆向迭代器
9. rend() 返回指向头部的逆向迭代器
10. resize() 改变双向队列的大小
11. size() 返回双向队列中元素的个数

23.swap() 和另一个双向队列交换元素 语法： void swap( deque &target );// 交换target和现双向队列中元素

2.4 三者比较

vector 是一段连续的内存块，而deque 是多个连续的内存块， list 是所有 数据元素分开保存，可以是任何两个元素没有连续。 vector 的查询性能最好，并且在末端增加数据也很好，除非它重新申请内存 段；适合高效地随机存储。 list 是一个链表，任何一个元素都可以是不连续的，但它都有两个指向上一 元素和下一元素的指针。所以它对插入、删除元素性能是最好的，而查询性能非 常差；适合大量地插入和删除操作而不关心随机存取的需求。 deque 是介于两者之间，它兼顾了数组和链表的优点，它是分块的链表和多 个数组的联合。所以它有被list好的查询性能，有被vector好的插入、删除性能。 如果你需要随即存取又关心两端数据的插入和删除，那么deque是最佳之选。

3 关联容器

3.1 特点

set, multiset, map, multimap 是一种非线性的树结构，具体的说采用的是一种 比较高效的特殊的平衡检索二叉树—— 红黑树结构。（至于什么是红黑树，我也 不太理解，只能理解到它是一种二叉树结构） 因为关联容器的这四种容器类都使用同一原理，所以他们核心的算法是一致 的，但是它们在应用上又有一些差别，先描述一下它们之间的差别。

set 又称集合，实际上就是一组元素的集合，但其中所包含的元素的值是唯 一的，且是按一定顺序排列的，集合中的每个元素被称作集合中的实例。因为其 内部是通过链表的方式来组织，所以在插入的时候比vector 快，但在查找和末尾 添加上比vector 慢。

multiset 是多重集合，其实现方式和set 是相似的，只是它不要求集合中的 元素是唯一的，也就是说集合中的同一个元素可以出现多次。

map 提供一种“键- 值”关系的一对一的数据存储能力。其“键”在容器中 不可重复，且按一定顺序排列（其实我们可以将set 也看成是一种键- 值关系的 存储，只是它只有键没有值。它是map 的一种特殊形式）。由于其是按链表的方 式存储，它也继承了链表的优缺点。

multimap 和map 的原理基本相似，它允许“键”在容器中可以不唯一。

关联容器的特点是明显的，相对于顺序容器，有以下几个主要特点：

1、其内部实现是采用非线性的二叉树结构，具体的说是红黑树的结构原理 实现的；

2、set 和map 保证了元素的唯一性，mulset 和mulmap 扩展了这一属性， 可以允许元素不唯一；

1. 元素是有序的集合，默认在插入的时候按升序排列。

基于以上特点，

1. 关联容器对元素的插入和删除操作比vector 要快，因为vector 是顺序存 储，而关联容器是链式存储；比list 要慢，是因为即使它们同是链式结构，但list 是线性的，而关联容器是二叉树结构，其改变一个元素涉及到其它元素的变动比 list 要多，并且它是排序的，每次插入和删除都需要对元素重新排序；

2、关联容器对元素的检索操作比vector 慢，但是比list 要快很多。vector 是 顺序的连续存储，当然是比不上的，但相对链式的list 要快很多是因为list 是逐 个搜索，它搜索的时间是跟容器的大小成正比，而关联容器 查找的复杂度基本 是Log(N) ，比如如果有1000 个记录，最多查找10 次，1,000,000 个记录，最多 查找20 次。容器越大，关联容器相对list 的优越性就越能体现；

3、在使用上set 区别于vector,deque,list 的最大特点就是set 是内部排序的， 这在查询上虽然逊色于vector ，但是却大大的强于list 。

4、在使用上map 的功能是不可取代的，它保存了“键- 值”关系的数据， 而这种键值关系采用了类数组的方式。数组是用数字类型的下标来索引元素的位 置，而map 是用字符型关键字来索引元素的位置。在使用上map 也提供了一种 类数组操作的方式，即它可以通过下标来检索数据，这是其他容器做不到的，当 然也包括set 。（ STL 中只有vector 和map 可以通过类数组的方式操作元素，即 如同ele[1] 方式）

3.2 C++ Sets & MultiSets

集合(Set)是一种包含已排序对象的关联容器。多元集合(MultiSets)和集合 (Sets)相像，只不过支持重复对象,其用法与set基本相同。

1. begin() 返回指向第一个元素的迭代器
2. clear() 清除所有元素

3.count() 返回某个值元素的个数

4.empty() 如果集合为空，返回true

5.end() 返回指向最后一个元素的迭代器

6.equal\_range() 返回第一个>=关键字的迭代器和>关键字的迭代器 语法： pair <iterator,iterator>equal\_range( const key\_type &key ); //key是用于排序的关键字 Set<int> ctr; 例如： Pair<set<int>::iterator,set<int>::iterarot>p; For(i=0;i<=5;i++) ctr.insert(i); P=ctr.equal\_range(2); 那么\*p.first==2;\*p.second==3;

7.erase() 删除集合中的元素 语法: iterator erase( iterator i ); //删除i位置元素 iterator erase( iterator start, iterator end ); //删除从start开始到end(end为第一个不被删除的值)结束的元素 size\_type erase( const key\_type &key ); //删除等于key值的所有元素（返回被删除的元素的个数） //前两个返回第一个不被删除的双向定位器,不存在返回末尾 //第三个返回删除个数

8.find() 返回一个指向被查找到元素的迭代器 语法: iterator find( const key\_type &key ); //查找等于key值的元素，并返回指向该元素的迭代器; //如果没有找到,返回指向集合最后一个元素的迭代器

9.get\_allocator() 返回集合的分配器

10.insert() 在集合中插入元素 语法: iterator insert( iterator i, const TYPE &val ); //在迭代器i前插入val void insert( input\_iterator start, input\_iterator end ); //将迭代器start开始到end（end不被插入）结束返回内的元素插入到集合中 pair insert( const TYPE &val ); //插入val元素，返回指向该元素的迭代器和一个布尔值来说明val是否成功被插入 //应该注意的是在集合(Sets中不能插入两个相同的元素)

11.lower\_bound() 返回指向大于（或等于）某值的第一个元素的迭代器 语法: iterator lower\_bound( const key\_type &key ); //返回一个指向大于或者等于key值的第一个元素的迭代器

12.key\_comp() 返回一个用于元素间值比较的函数 语法: key\_compare key\_comp(); //返回一个用于元素间值比较的函数对象

13.max\_size() 返回集合能容纳的元素的最大限值

14.rbegin() 返回指向集合中最后一个元素的反向迭代器 示例： Set<int> ctr; Set<int>::reverse\_iterator rcp; For(rcp=ctr.rbegin();rcp!=ctr.rend();rcp++) Cout<<\*rcp<<” ”;

15.rend() 返回指向集合中第一个元素的反向迭代器

16.size() 集合中元素的数目

17.swap() 交换两个集合变量 语法: void swap( set &object ); //交换当前集合和object集合中的元素

18.upper\_bound() 返回大于某个值元素的迭代器 语法: iterator upwer\_bound( const key\_type &key ); //返回一个指向大于key值的第一个元素的迭代器

19.value\_comp() 返回一个用于比较元素间的值的函数 语法: iterator upper\_bound( const key\_type &key );//返回一个用于比较元素间的值的函数对象

3.3 C++ Maps & MultiMaps

C++ Maps是一种关联式容器，包含“关键字/值”对。

C++ Multimaps和maps很相似，但是MultiMaps允许重复的元素。

1. begin() 返回指向map头部的迭代器
2. clear() 删除所有元素
3. count() 返回指定元素出现的次数 语法: size\_type count( const KEY\_TYPE &key ); //返回map中键值等于key的元素的个数
4. empty() 如果map为空则返回true

5.end() 返回指向map末尾的迭代器

6.equal\_range() 返回特殊条目的迭代器对 语法: pair equal\_range( const KEY\_TYPE &key ); 返回两个迭代器,指向第一个键值为key的元素和指向最后一个键值为key的元素

7.erase() 删除一个元素 语法: void erase( iterator i ); //删除i元素 void erase( iterator start, iterator end ); //删除从start开始到end（不包括end）结束的元素 size\_type erase( const key\_type &key ); //删除等于key值的所有元素（返回被删除的元素的个数）

8.find() 查找一个元素 语法: iterator find( const key\_type &key );

//查找等于key值的元素，并返回指向该元素的迭代器; //如果没有找到,返回指向集合最后一个元素的迭代器. 9.get\_allocator() 返回map的配置器

1. insert() 插入元素 语法: iterator insert( iterator pos, const pair<KEY\_TYPE,VALUE\_TYPE> &val ); //插入val到pos的后面，然后返回一个指向这个元素的迭代器 void insert( input\_iterator start, input\_iterator end ); //插入start到end的元素到map中 pair<iterator, bool> insert( const pair<KEY\_TYPE,VALUE\_TYPE> &val ); //只有在val不存在时插入val。返回指向被插入元素的迭代器和描述是否插入的bool值
2. key\_comp() 返回比较元素key的函数 语法: key\_compare key\_comp(); //返回一个用于元素间值比较的函数对象
3. lower\_bound() 返回键值>=给定元素的第一个位置 语法: iterator lower\_bound( const key\_type &key ); //返回一个指向大于或者等于key值的第一个元素的迭代器
4. max\_size() 返回可以容纳的最大元素个数
5. rbegin() 返回一个指向map尾部的逆向迭代器

15.rend() 返回一个指向map头部的逆向迭代器

16.size() 返回map中元素的个数

17.swap() 交换两个map 语法: void swap( map &obj ); //swap()交换obj和现map中的元素

18.upper\_bound() 返回键值>给定元素的第一个位置 语法: iterator upwer\_bound( const key\_type &key ); //返回一个指向大于key值的第一个元素的迭代器

19.value\_comp() 返回比较元素value的函数 语法: value\_compare value\_comp(); //返回一个用于比较元素value的函数

4 容器适配器

4.1 特点

STL 中包含三种适配器：栈stack 、队列queue 和优先级priority\_queue 。 适配器是容器的接口，它本身不能直接保存元素，它保存元素的机制是调用另一种顺序容器去实现，即可以把适配器看作“它保存一个容器，这个容器再保 存所有元素”。

STL 中提供的三种适配器可以由某一种顺序容器去实现。默认下stack 和 queue 基于deque 容器实现，priority\_queue 则基于vector 容器实现。当然在创 建一个适配器时也可以指定具体的实现容器，创建适配器时在第二个参数上指定 具体的顺序容器可以覆盖适配器的默认实现。

由于适配器的特点，一个适配器不是可以由任一个顺序容器都可以实现的。

栈stack 的特点是后进先出，所以它关联的基本容器可以是任意一种顺序容 器，因为这些容器类型结构都可以提供栈的操作有求，它们都提供了push\_back 、 pop\_back 和back 操作。

队列queue 的特点是先进先出，适配器要求其关联的基础容器必须提供 pop\_front 操作，因此其不能建立在vector 容器上。

4.2 C++ Stacks（堆栈）

C++ Stack（堆栈） 是一个容器类的改编，为程序员提供了堆栈的全部功能，— —也就是说实现了一个先进后出（FILO）的数据结构。

1. empty() 堆栈为空则返回真
2. pop() 移除栈顶元素
3. push() 在栈顶增加元素
4. size() 返回栈中元素数目
5. top() 返回栈顶元素

4.3 C++ Queues(队列)

C++队列是一种容器适配器，它给予程序员一种先进先出(FIFO)的数据结构。

1. back() 返回一个引用，指向最后一个元素
2. empty() 如果队列空则返回真
3. front() 返回第一个元素
4. pop() 删除第一个元素
5. push() 在末尾加入一个元素
6. size() 返回队列中元素的个数

4.4 C++ Priority Queues(优先队列)

C++优先队列类似队列，但是在这个数据结构中的元素按照一定的断言排列有 序。

1. empty() 如果优先队列为空，则返回真

2.pop() 删除第一个元素

3.push() 加入一个元素

4.size() 返回优先队列中拥有的元素的个数

5.top() 返回优先队列中有最高优先级的元素

5 迭代器

5.1 解释

迭代器是一种对象，它能够用来遍历STL容器中的部分或全部元素，每个迭 代器对象代表容器中的确定的地址。迭代器修改了常规指针的接口，所谓迭代器 是一种概念上的抽象：那些行为上象迭代器的东西都可以叫做迭代器。然而迭代 器有很多不同的能力，它可以把抽象容器和通用算法有机的统一起来。

迭代器提供一些基本操作符：\*、++、==、！ =、=。这些操作和C/C++“操作 array元素”时的指针接口一致。不同之处在于，迭代器是个所谓的smart pointers， 具有遍历复杂数据结构的能力。其下层运行机制取决于其所遍历的数据结构。因 此，每一种容器型别都必须提供自己的迭代器。事实上每一种容器都将其迭代器 以嵌套的方式定义于内部。因此各种迭代器的接口相同，型别却不同。这直接导 出了泛型程序设计的概念：所有操作行为都使用相同接口，虽然它们的型别不同。

5.2 功能特点

迭代器使开发人员不必整个实现类接口。只需提供一个迭代器，即可遍历类 中的数据结构，可被用来访问一个容器类的所包函的全部元素，其行为像一个指 针，但是只可被进行增加(++)或减少(--)操作。举一个例子，你可用一个迭代器 来实现对vector容器中所含元素的遍历。 如下代码对vector容器对象生成和使用了迭代器：

vector<int> the\_vector; vector<int>::iterator the\_iterator; for( int i=0; i < 10; i++ ) the\_vector.push\_back(i); int total = 0; the\_iterator = the\_vector.begin(); while( the\_iterator != the\_vector.end() ) { total += \*the\_iterator; the\_iterator++; }cout << "Total=" << total << endl;

提示：通过对一个迭代器的解引用操作（\*），可以访问到容器所包含的元素。

6 C++标准库总结

6.1 容器

6.1.1 序列

vector========================<vector>

list==========================<list>

deque========================<deque>

6.1.2 序列适配器

stack:top,push,pop===========<stack>

queue:front,back,push,pop=====<queue>

priority\_queue:top,push,pop====<queue>

6.1.3 关联容器

map==========================<map>

multimap======================<map>

set===========================<set>

multiset=======================<set>

6.1.4 拟容器

string=========================<string>

valarray=======================<valarray>

bitset=========================<bitset>

6.2 算法

http://www.cplusplus.com/reference/algorithm/详细

6.2.1 非修改性序列操作

<algorithm>

for\_each()===================对序列中每个元素执行操作

find()======================在序列中找某个值的第一个出现

find\_if()====================在序列中找符合某谓词的第一个元素

find\_first\_of()================在序列中找另一序列里的值

adjust\_find()=================找出相邻的一对值

count()=====================在序列中统计某个值出现的次数

count\_if()===================在序列中统计与某谓词匹配的次数

mismatch()==================找使两序列相异的第一个元素

equal()=====================如果两个序列对应元素都相同则为真

search()====================找出一序列作为子序列的第一个出现位置

find\_end()==================找出一序列作为子序列的最后一个出现位置

search\_n()==================找出一序列作为子序列的第 n 个出现位置

6.2.2 修改性的序列操作

<algorithm>

transform()====================将操作应用于序列中的每个元素

copy()=======================从序列的第一个元素起进行复制

copy\_backward()===============从序列的最后元素起进行复制

swap()=======================交换两个元素

iter\_swap()====================交换由迭代器所指的两个元素

swap\_ranges()==================交换两个序列中的元素

replace()======================用一个给定值替换一些元素

replace\_if()===================替换满足谓词的一些元素

replace\_copy()=================复制序列时用一个给定值替换元素

replace\_copy\_if()==============复制序列时替换满足谓词的元素

fill()=========================用一个给定值取代所有元素

fill\_n()=======================用一个给定值取代前 n 个元素

generate()=====================用一个操作的结果取代所有元素

generate\_n()===================用一个操作的结果取代前 n 个元素

remove()=======================删除具有给定值的元素

remove\_if()====================删除满足谓词的元素

remove\_copy()==================复制序列时删除给定值的元素

remove\_copy\_if()===============复制序列时删除满足谓词的元素

unique()=======================删除相邻的重复元素

unique\_copy()==================复制序列时删除相邻的重复元素

reexample()======================反转元素的次序

reexample\_copy()=================复制序列时反转元素的次序

rotate()=======================循环移动元素

rotate\_copy()==================复制序列时循环移动元素

random\_shuffle()===============采用均匀分布随机移动元素

6.2.3 序列排序

<algorithm>

sort()=========================以很好的平均次序排序

stable\_sort()==================排序且维持相同元素原有的顺序

partial\_sort()=================将序列的前一部分排好序

partial\_sort\_copy()============复制的同时将序列的前一部分排好序

nth\_element()==================将第 n 个元素放到它的正确位置

lower\_bound()==================找到某个值的第一个出现

upper\_bound()==================找到大于某个值的第一个出现

equal\_range()==================找出具有给定值的一个子序列

binary\_search()================在排好序的序列中确定给定元素是否存在

merge()========================归并两个排好序的序列

inplace\_merge()================归并两个接续的排好序的序列

partition()====================将满足某谓词的元素都放到前面

stable\_partition()===========将满足某谓词的元素都放到前面且维持原顺序

6.2.4 集合算法

<algorithm>

include()======================如果一个序列是另一个的子序列则为真

set\_union()====================构造一个已排序的并集

set\_intersection()=============构造一个已排序的交集

set\_difference()构造一个已排序序列，包含在第一个序列但不在第二个序列的

set\_symmetric\_difference()构造一个已排序序列，包括所有只在两个序列之一中

6.2.5 堆操作

<algorithm>

make\_heap()====================将序列高速得能够作为堆使用

push\_heap()====================向堆中加入一个元素

pop\_heap()=====================从堆中去除元素

sort\_heap()====================对堆排序

6.2.6 最大和最小

<algorithm>

min()==========================两个值中较小的

max()==========================两个值中较大的

min\_element()==================序列中的最小元素

max\_element()==================序列中的最大元素

lexicographic\_compare()========两个序列中按字典序的第一个在前

6.2.7 排列

<algorithm>

next\_permutation()=============按字典序的下一个排列

prev\_permutation()=============按字典序的前一个排列

6.2.8 通用数值算法

<numeric>

accumulate()=======积累在一个序列中运算的结果(向量的元素求各的 推广)

inner\_product()============积累在两个序列中运算的结果(内积)

partial\_sum()=============通过在序列上的运算产生序列(增量变化)

adjacent\_difference()===通过在序列上的运算产生序列(与 partial\_sum 相反)

6.2.9 C 风格算法

<cstdlib>

qsort()=======快速排序，元素不能有用户定义的构造，拷贝赋 值和析构函数

bsearch()====二分法查找，元素不能有用户定义的构造，拷贝 赋值和析构函数

6.3 函数对象

6.3.1 基类

template<class Arg, class Res> struct unary\_function

emplate<class Arg, class Arg2, class Res> struct binary\_function

6.3.2 谓词

返回 bool 的函数对象。

<functional>

equal\_to=======================二元，arg1 == arg2

not\_equal\_to===================二元，arg1 != arg2

greater========================二元，arg1 > arg2

less===========================二元，arg1 < arg2

greater\_equal==================二元，arg1 >= arg2

less\_equal=====================二元，arg1 <= arg2

logical\_and====================二元，arg1 && arg2

logical\_or=====================二元，arg1 || arg2

logical\_not====================一元，!arg

6.3.3 算术函数对象

<functional>

plus===========================二元，arg1 + arg2

minus==========================二元，arg1 - arg2

multiplies=====================二元，arg1 \* arg2

divides========================二元，arg1 / arg2

modulus========================二元，arg1 % arg2

negate=========================一元，-arg

6.3.4 约束器，适配器和否定器

<functional>

bind2nd(y)

binder2nd==================以 y 作为第二个参数调用二元函数

bind1st(x)

binder1st==================以 x 作为第一个参数调用二元函数

mem\_fun()

mem\_fun\_t==================通过指针调用 0 元成员函数

mem\_fun1\_t=================通过指针调用一元成员函数

const\_mem\_fun\_t============通过指针调用 0 元 const 成员函数

const\_mem\_fun1\_t===========通过指针调用一元 const 成员函数

mem\_fun\_ref()

mem\_fun\_ref\_t==============通过引用调用 0 元成员函数

mem\_fun1\_ref\_t=============通过引用调用一元成员函数

const\_mem\_fun\_ref\_t========通过引用调用 0 元 const 成员函数

const\_mem\_fun1\_ref\_t=======通过引用调用一元 const 成员函数

ptr\_fun()

pointer\_to\_unary\_function==调用一元函数指针

ptr\_fun()

pointer\_to\_binary\_function=调用二元函数指针

not1()

unary\_negate===============否定一元谓词

not2()

binary\_negate==============否定二元谓词

6.4 迭代器

6.4.1 分类

Output: \*p= , ++

Input: =\*p , -> , ++ , == , !=

Forward: \*p= , =\*p , -> , ++ , == , !=

Bidirectional: \*p= , =\*p -> , [] , ++ , -- , == , !=

Random: += , -= , \*p= , =\*p -> , [] , ++ , -- , + , - , == , != , < , > , <= , >=

6.4.2 插入器

template<class Cont> back\_insert\_iterator<Cont> back\_inserter(Cont& c);

template<class Cont> front\_insert\_iterator<Cont> front\_inserter(Cont& c);

template<class Cont, class Out> insert\_iterator<Cont> back\_inserter(Cont& c, Out p);

6.4.3 反向迭代器

reexample\_iterator===============rbegin(), rend()

6.4.4 流迭代器

ostream\_iterator===============用于向 ostream 写入

istream\_iterator===============用于向 istream 读出

ostreambuf\_iterator============用于向流缓冲区写入

istreambuf\_iterator============用于向流缓冲区读出

6.5 分配器

<memory>

template<class T> class std::allocator

6.6 数值

6.6.1 数值的限制

<limits>

numeric\_limits<>

<climits>

CHAR\_BIT

INT\_MAX

...

<cfloat>

DBL\_MIN\_EXP

FLT\_RADIX

LDBL\_MAX

...

6.6.2 标准数学函数

<cmath>

double abs(double)=============绝对值(不在 C 中)，同 fabs()

double fabs(double)============绝对值

double ceil(double d)==========不小于 d 的最小整数

double floor(double d)=========不大于 d 的最大整数

double sqrt(double d)==========d 在平方根，d 必须非负

double pow(double d, double e)=d 的 e 次幂

double pow(double d, int i)====d 的 i 次幂

double cos(double)=============余弦

double sin(double)=============正弦

double tan(double)=============正切

double acos(double)============反余弦

double asin(double)============反正弦

double atan(double)============反正切

double atan2(double x,double y) //atan(x/y)

double sinh(double)============双曲正弦

double cosh(double)============双曲余弦

double tanh(double)============双曲正切

double exp(double)=============指数，以 e 为底

double log(double d)===========自动对数(以 e 为底),d 必须大于 0

double log10(double d)=========10 底对数，d 必须大于 0

double modf(double d,double\*p)=返回 d 的小数部分，整数部分存入\*p

double frexp(double d, int\* p)找出[0.5,1)中的 x,y,使 d=x\*pow(2,y),返回 x 并将 y 存入\*p

double fmod(double d,double m)=浮点数余数，符号与 d 相同

double ldexp(double d, int i)==d\*pow(2,i)

<cstdlib>

int abs(int)===================绝对值

long abs(long)=================绝对值(不在 C 中)

long labs(long)================绝对值

struct div\_t { implementation\_defined quot, rem; }

struct ldiv\_t { implementation\_defined quot, rem; }

div\_t div(int n, int d)========用 d 除 n，返回(商，余数)

ldiv\_t div(long n, long d)=====用 d 除 n，返回(商，余数)(不在 C 中)

ldiv\_t ldiv(long n, long d)====用 d 除 n，返回(商，余数) 6.6.3 向量算术 <valarray>

valarray 6.6.4 复数算术

<complex>

template<class T> class std::complex;

6.6.5 通用数值算法

见 6.2.8

C++string类常用函数

string类的构造函数：

string(const char \*s); //用c字符串s初始化

string(int n,char c); //用n个字符c初始化

此外，string类还支持默认构造函数和复制构造函数，如string s1；string s2="hello"；都是正确的写法。当构造的string太长而无法表达时会抛出length\_error异常

string类的字符操作：

const char &operator[](int n)const;

const char &at(int n)const;

char &operator[](int n);

char &at(int n);

operator[]和at()均返回当前字符串中第n个字符的位置，但at函数提供范围检查，当越界时会抛出out\_of\_range异常，下标运算符[]不提供检查访问。

const char \*data()const;//返回一个非null终止的c字符数组

const char \*c\_str()const;//返回一个以null终止的c字符串

int copy(char \*s, int n, int pos = 0) const;//把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目

string的特性描述:

int capacity()const; //返回当前容量（即string中不必增加内存即可存放的元素个数）

int max\_size()const; //返回string对象中可存放的最大字符串的长度

int size()const; //返回当前字符串的大小

int length()const; //返回当前字符串的长度

bool empty()const; //当前字符串是否为空

void resize(int len,char c);//把字符串当前大小置为len，并用字符c填充不足的部分

string类的输入输出操作:

string类重载运算符operator>>用于输入，同样重载运算符operator<<用于输出操作。

函数getline(istream &in,string &s);用于从输入流in中读取字符串到s中，以换行符'\n'分开。

string的赋值：

string &operator=(const string &s);//把字符串s赋给当前字符串

string &assign(const char \*s);//用c类型字符串s赋值

string &assign(const char \*s,int n);//用c字符串s开始的n个字符赋值

string &assign(const string &s);//把字符串s赋给当前字符串

string &assign(int n,char c);//用n个字符c赋值给当前字符串

string &assign(const string &s,int start,int n);//把字符串s中从start开始的n个字符赋给当前字符串

string &assign(const\_iterator first,const\_itertor last);//把first和last迭代器之间的部分赋给字符串

string的连接：

string &operator+=(const string &s);//把字符串s连接到当前字符串的结尾

string &append(const char \*s); //把c类型字符串s连接到当前字符串结尾

string &append(const char \*s,int n);//把c类型字符串s的前n个字符连接到当前字符串结尾

string &append(const string &s); //同operator+=()

string &append(const string &s,int pos,int n);//把字符串s中从pos开始的n个字符连接到当前字符串的结尾

string &append(int n,char c); //在当前字符串结尾添加n个字符c

string &append(const\_iterator first,const\_iterator last);//把迭代器first和last之间的部分连接到当前字符串的结尾

string的比较：

bool operator==(const string &s1,const string &s2)const;//比较两个字符串是否相等

运算符">","<",">=","<=","!="均被重载用于字符串的比较；

int compare(const string &s) const;//比较当前字符串和s的大小

int compare(int pos, int n,const string &s)const;//比较当前字符串从pos开始的n个字符组成的字符串与s的大小

int compare(int pos, int n,const string &s,int pos2,int n2)const;//比较当前字符串从pos开始的n个字符组成的字符串与s中pos2开始的n2个字符组成的字符串的大小

int compare(const char \*s) const;

int compare(int pos, int n,const char \*s) const;

int compare(int pos, int n,const char \*s, int pos2) const;

compare函数在>时返回1，<时返回-1，==时返回0

string的子串：

string substr(int pos = 0,int n = npos) const;//返回pos开始的n个字符组成的字符串

string的交换：

void swap(string &s2); //交换当前字符串与s2的值

string类的查找函数：

int find(char c, int pos = 0) const;//从pos开始查找字符c在当前字符串的位置

int find(const char \*s, int pos = 0) const;//从pos开始查找字符串s在当前串中的位置

int find(const char \*s, int pos, int n) const;//从pos开始查找字符串s中前n个字符在当前串中的位置

int find(const string &s, int pos = 0) const;//从pos开始查找字符串s在当前串中的位置

//查找成功时返回所在位置，失败返回string::npos的值

int rfind(char c, int pos = npos) const;//从pos开始从后向前查找字符c在当前串中的位置

int rfind(const char \*s, int pos = npos) const;

int rfind(const char \*s, int pos, int n = npos) const;

int rfind(const string &s,int pos = npos) const;

//从pos开始从后向前查找字符串s中前n个字符组成的字符串在当前串中的位置，成功返回所在位置，失败时返回string::npos的值

int find\_first\_of(char c, int pos = 0) const;//从pos开始查找字符c第一次出现的位置

int find\_first\_of(const char \*s, int pos = 0) const;

int find\_first\_of(const char \*s, int pos, int n) const;

int find\_first\_of(const string &s,int pos = 0) const;

//从pos开始查找当前串中第一个在s的前n个字符组成的数组里的字符的位置。查找失败返回string::npos

int find\_first\_not\_of(char c, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos,int n) const;

int find\_first\_not\_of(const string &s,int pos = 0) const;

//从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos

int find\_last\_of(char c, int pos = npos) const;

int find\_last\_of(const char \*s, int pos = npos) const;

int find\_last\_of(const char \*s, int pos, int n = npos) const;

int find\_last\_of(const string &s,int pos = npos) const;

int find\_last\_not\_of(char c, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos, int n) const;

int find\_last\_not\_of(const string &s,int pos = npos) const;

//find\_last\_of和find\_last\_not\_of与find\_first\_of和find\_first\_not\_of相似，只不过是从后向前查找

string类的替换函数：

string &replace(int p0, int n0,const char \*s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const char \*s, int n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符

string &replace(int p0, int n0,const string &s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const string &s, int pos, int n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符

string &replace(int p0, int n0,int n, char c);//删除p0开始的n0个字符，然后在p0处插入n个字符c

string &replace(iterator first0, iterator last0,const char \*s);//把[first0，last0）之间的部分替换为字符串s

string &replace(iterator first0, iterator last0,const char \*s, int n);//把[first0，last0）之间的部分替换为s的前n个字符

string &replace(iterator first0, iterator last0,const string &s);//把[first0，last0）之间的部分替换为串s

string &replace(iterator first0, iterator last0,int n, char c);//把[first0，last0）之间的部分替换为n个字符c

string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串

string类的插入函数：

string &insert(int p0, const char \*s);

string &insert(int p0, const char \*s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);

//前4个函数在p0位置插入字符串s中pos开始的前n个字符

string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c

iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置

void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在it处插入n个字符c

string类的删除函数

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置

iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置

string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串

string类的迭代器处理：

string类提供了向前和向后遍历的迭代器iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。

用string::iterator或string::const\_iterator声明迭代器变量，const\_iterator不允许改变迭代的内容。常用迭代器函数有：

const\_iterator begin()const;

iterator begin(); //返回string的起始位置

const\_iterator end()const;

iterator end(); //返回string的最后一个字符后面的位置

const\_iterator rbegin()const;

iterator rbegin(); //返回string的最后一个字符的位置

const\_iterator rend()const;

iterator rend(); //返回string第一个字符位置的前面

rbegin和rend用于从后向前的迭代访问，通过设置迭代器string::reverse\_iterator,string::const\_reverse\_iterator实现

字符串流处理：

通过定义ostringstream和istringstream变量实现，<sstream>头文件中

例如：

string input("hello,this is a test");

istringstream is(input);

string s1,s2,s3,s4;

is>>s1>>s2>>s3>>s4;//s1="hello,this",s2="is",s3="a",s4="test"

ostringstream os;

os<<s1<<s2<<s3<<s4;

cout<<os.str();

C库函数手册

分类函数,所在函数库为ctype.h

int isalpha(int ch) 若ch是字母('A'-'Z','a'-'z')返回非0值,否则返回0

int isalnum(int ch) 若ch是字母('A'-'Z','a'-'z')或数字('0'-'9') 返回1,否则返回0

int isascii(int ch) 若ch是字符(ASCII码中的0-127)返回非0值,否则返回0

int iscntrl(int ch) 若ch是作废字符(0x7F)或普通控制字符(0x00-0x1F) 返回1,否则返回0

int isdigit(int ch) 若ch是数字('0'-'9')返回非0值,否则返回0

int isgraph(int ch) 若ch是可打印字符(不含空格)(0x21-0x7E)返回1,否则返回0

int islower(int ch) 若ch是小写字母('a'-'z')返回非0值,否则返回0

int isprint(int ch) 若ch是可打印字符(含空格)(0x20-0x7E)返回非0值,否则返回0

int ispunct(int ch) 若ch是标点字符(0x00-0x1F)返回非0值,否则返回0

int isspace(int ch) 若ch是空格(' '),水平制表符('\t'),回车符('\r'),

走纸换行('\f'),垂直制表符('\v'),换行符('\n')

返回非0值,否则返回0

int isupper(int ch) 若ch是大写字母('A'-'Z')返回非0值,否则返回0

int isxdigit(int ch) 若ch是16进制数('0'-'9','A'-'F','a'-'f')返回非0值,

否则返回0

int tolower(int ch) 若ch是大写字母('A'-'Z')返回相应的小写字母('a'-'z')

int toupper(int ch) 若ch是小写字母('a'-'z')返回相应的大写字母('A'-'Z')

数学函数,所在函数库为math.h、stdlib.h、string.h、float.h

int abs(int i) 返回整型参数i的绝对值

double cabs(struct complex znum) 返回复数znum的绝对值

double fabs(double x) 返回双精度参数x的绝对值

long labs(long n) 返回长整型参数n的绝对值

double exp(double x) 返回指数函数ex的值

double frexp(double value,int \*eptr) 返回value=x\*2n中x的值,n存贮在eptr中

double ldexp(double value,int exp); 返回value\*2exp的值

double log(double x) 返回logex的值

double log10(double x) 返回log10x的值

double pow(double x,double y) 返回xy的值

double pow10(int p) 返回10p的值

double sqrt(double x) 返回+√x的值

double acos(double x) 返回x的反余弦cos-1(x)值,x为弧度

double asin(double x) 返回x的反正弦sin-1(x)值,x为弧度

double atan(double x) 返回x的反正切tan-1(x)值,x为弧度

double atan2(double y,double x) 返回y/x的反正切tan-1(x)值,y的x为弧度

double cos(double x) 返回x的余弦cos(x)值,x为弧度

double sin(double x) 返回x的正弦sin(x)值,x为弧度

double tan(double x) 返回x的正切tan(x)值,x为弧度

double cosh(double x) 返回x的双曲余弦cosh(x)值,x为弧度

double sinh(double x) 返回x的双曲正弦sinh(x)值,x为弧度

double tanh(double x) 返回x的双曲正切tanh(x)值,x为弧度

double hypot(double x,double y) 返回直角三角形斜边的长度(z),

x和y为直角边的长度,z2=x2+y2

double ceil(double x) 返回不小于x的最小整数

double floor(double x) 返回不大于x的最大整数

void srand(unsigned seed) 初始化随机数发生器

int rand() 产生一个随机数并返回这个数

double poly(double x,int n,double c[])从参数产生一个多项式

double modf(double value,double \*iptr)将双精度数value分解成尾数和阶

double fmod(double x,double y) 返回x/y的余数

double frexp(double value,int \*eptr) 将双精度数value分成尾数和阶

unsigned int \_clear87() 清除浮点状态字并返回原来的浮点状态

void \_fpreset() 重新初使化浮点数学程序包

unsigned int \_status87() 返回浮点状态字

转换子程序,函数库为math.h、stdlib.h、ctype.h、float.h

char \*ecvt(double value,int ndigit,int \*decpt,int \*sign)

将浮点数value转换成字符串并返回该字符串

char \*fcvt(double value,int ndigit,int \*decpt,int \*sign)

将浮点数value转换成字符串并返回该字符串

char \*gcvt(double value,int ndigit,char \*buf)

将数value转换成字符串并存于buf中,并返回buf的指针

char \*ultoa(unsigned long value,char \*string,int radix)

将无符号整型数value转换成字符串并返回该字符串,radix为转换时所用基数

char \*ltoa(long value,char \*string,int radix)

将长整型数value转换成字符串并返回该字符串,radix为转换时所用基数

char \*itoa(int value,char \*string,int radix)

将整数value转换成字符串存入string,radix为转换时所用基数

double atof(char \*nptr) 将字符串nptr转换成双精度数,并返回这个数,错误返回0

int atoi(char \*nptr) 将字符串nptr转换成整型数, 并返回这个数,错误返回0

long atol(char \*nptr) 将字符串nptr转换成长整型数,并返回这个数,错误返回0

double strtod(char \*str,char \*\*endptr)将字符串str转换成双精度数,并返回这个数,

long strtol(char \*str,char \*\*endptr,int base)

将字符串str转换成长整型数,并返回这个数,

int toascii(int c) 返回c相应的ASCII

int tolower(int ch) 若ch是大写字母('A'-'Z')返回相应的小写字母('a'-'z')

int \_tolower(int ch) 返回ch相应的小写字母('a'-'z')

int toupper(int ch) 若ch是小写字母('a'-'z')返回相应的大写字母('A'-'Z')

int \_toupper(int ch) 返回ch相应的大写字母('A'-'Z')

输入输出子程序,函数库为io.h、conio.h、stat.h、dos.h、stdio.h、signal.h

int kbhit() 本函数返回最近所敲的按键

int fgetchar() 从控制台(键盘)读一个字符，显示在屏幕上

int getch() 从控制台(键盘)读一个字符，不显示在屏幕上

int putch() 向控制台(键盘)写一个字符

int getchar() 从控制台(键盘)读一个字符，显示在屏幕上

int putchar() 向控制台(键盘)写一个字符

int getche() 从控制台(键盘)读一个字符，显示在屏幕上

int ungetch(int c) 把字符c退回给控制台(键盘)

char \*cgets(char \*string) 从控制台(键盘)读入字符串存于string中

int scanf(char \*format[,argument…])

从控制台读入一个字符串,分别对各个参数进行赋值,使用BIOS进行输出

int vscanf(char \*format,Valist param)从控制台读入一个字符串,分别对各个参数进行赋值,使用BIOS进行输出,参数从Valist param中取得

int cscanf(char \*format[,argument…])

从控制台读入一个字符串,分别对各个参数进行 赋值,直接对控制台作操作,比如显示器在显示时字符时即为直接写频方式显示

int sscanf(char \*string,char \*format[,argument,…])

通过字符串string,分别对各个参数进行赋值

int vsscanf(char \*string,char \*format,Vlist param)通过字符串string,分别对各个

参数进行赋值,参数从Vlist param中取得

int puts(char \*string) 发关一个字符串string给控制台(显示器),

使用BIOS进行输出

void cputs(char \*string) 发送一个字符串string给控制台(显示器),

直接对控制台作操作,比如显示器即为直接写频方式显示

int printf(char \*format[,argument,…]) 发送格式化字符串输出给控制台(显示器) 使用BIOS进行输出

int vprintf(char \*format,Valist param) 发送格式化字符串输出给控制台(显示器)使用BIOS进行输出,参数从Valist param中取得

int cprintf(char \*format[,argument,…]) 发送格式化字符串输出给控制台(显示器),直接对控制台作操作,比如显示器即为直接写频方式显示

int vcprintf(char \*format,Valist param)发送格式化字符串输出给控制台(显示器),直接对控制台作操作,比如显示器即为直接写频方式显示,参数从Valist param中取得

int sprintf(char \*string,char \*format[,argument,…])

将字符串string的内容重新写为格式化后的字符串

FILE \*freopen(char \*filename,char \*type,FILE \*stream)

int getc(FILE \*stream) 从流stream中读一个字符，并返回这个字符

int putc(int ch,FILE \*stream)向流stream写入一个字符ch

int getw(FILE \*stream) 从流stream读入一个整数，错误返回EOF

int putw(int w,FILE \*stream)向流stream写入一个整数

int ungetc(char c,FILE \*stream)

把字符c退回给流stream，下一次读进的字符将是c

int fgetc(FILE \*stream) 从流stream处读一个字符，并返回这个字符

int fputc(int ch,FILE \*stream) 将字符ch写入流stream中

char \*fgets(char \*string,int n,FILE \*stream)

从流stream中读n个字符存入string中

int fputs(char \*string,FILE \*stream) 将字符串string写入流stream中

int fread(void \*ptr,int size,int nitems,FILE \*stream) 从流stream中读入nitems

个长度为size的字符串存入ptr中

int fwrite(void \*ptr,int size,int nitems,FILE \*stream) 向流stream中写入nitems

个长度为size的字符串,字符串在ptr中

int fscanf(FILE \*stream,char \*format[,argument,…]) 以格式化形式从流stream中读入一个字符串

int vfscanf(FILE \*stream,char \*format,Valist param) 以格式化形式从流stream中 读入一个字符串,参数从Valist param中取得

int fprintf(FILE \*stream,char \*format[,argument,…]) 以格式化形式将一个字符串写给指定的流stream

int vfprintf(FILE \*stream,char \*format,Valist param) 以格式化形式将一个字符 串写给指定的流stream,参数从Valist param中取得

int fseek(FILE \*stream,long offset,int fromwhere) 函数把文件指针移到fromwhere

所指位置的向后offset个字节处,fromwhere可以为以下值:SEEK\_SET 文件开关 SEEK\_CUR 当前位置 SEEK\_END 文件尾

long ftell(FILE \*stream) 函数返回定位在stream中的当前文件指针位置,以字节表示

int rewind(FILE \*stream) 将当前文件指针stream移到文件开头

int feof(FILE \*stream) 检测流stream上的文件指针是否在结束位置

int fileno(FILE \*stream) 取流stream上的文件处理，并返回文件处理

int ferror(FILE \*stream) 检测流stream上是否有读写错误，如有错误就返回1

void clearerr(FILE \*stream) 清除流stream上的读写错误

void setbuf(FILE \*stream,char \*buf) 给流stream指定一个缓冲区buf

int fclose(FILE \*stream) 关闭一个流，可以是文件或设备(例如LPT1)

int fcloseall() 关闭所有除stdin或stdout外的流

int fflush(FILE \*stream) 关闭一个流，并对缓冲区作处理,即对读的流，将流内内容读入缓冲区；对写的流，将缓冲区内内容写入流。成功返回0

int fflushall() 关闭所有流，并对流各自的缓冲区作处理,即对读的流，将流内内容读入缓冲区；对写的流，将缓冲区内内容写入流。成功返回0

操作函数,所在函数库为string.h、mem.h

mem…操作存贮数组

void \*memccpy(void \*destin,void \*source,unsigned char ch,unsigned n)

void \*memchr(void \*s,char ch,unsigned n)

void \*memcmp(void \*s1,void \*s2,unsigned n)

int memicmp(void \*s1,void \*s2,unsigned n)

void \*memmove(void \*destin,void \*source,unsigned n)

void \*memcpy(void \*destin,void \*source,unsigned n)

void \*memset(void \*s,char ch,unsigned n)

这些函数,mem…系列的所有成员均操作存贮数组.在所有这些函数中,数组是n字节长.

memcpy从source复制一个n字节的块到destin.如果源块和目标块重迭,则选择复制方向,以例正确地复制覆盖的字节.

memmove与memcpy相同.

memset将s的所有字节置于字节ch中.s数组的长度由n给出.

memcmp比较正好是n字节长的两个字符串s1和s2.些函数按无符号字符比较字节,因此,memcmp("0xFF","\x7F",1)返回值大于0.

memicmp比较s1和s2的前n个字节,不管字符大写或小写.

memccpy从source复制字节到destin.复制一结束就发生下列任一情况:

(1)字符ch首选复制到destin.

(2)n个字节已复制到destin.

memchr对字符ch检索s数组的前n个字节.

返回值:memmove和memcpy返回destin

memset返回s的值

memcmp和memicmp─┬─若s1<s2返回值小于0

├─若s1=s2返回值等于0

└─若s1>s2返回值大于0

memccpy若复制了ch,则返回直接跟随ch的在destin中的字节的一个指针否则返回NULL,memchr返回在s中首先出现ch的一个指针;如果在s数组中不出现ch,就返回NULL.

void movemem(void \*source,void \*destin,unsigned len)

本函数从source处复制一块长len字节的数据到destin.若源地址和目标地址字符串 重迭,则选择复制方向,以便正确的复制数据.

str…字符串操作函数

char stpcpy(char \*dest,const char \*src)

将字符串src复制到dest

char strcat(char \*dest,const char \*src)

将字符串src添加到dest末尾

char strchr(const char \*s,int c)

检索并返回字符c在字符串s中第一次出现的位置

int strcmp(const char \*s1,const char \*s2)

比较字符串s1与s2的大小,并返回s1-s2

char strcpy(char \*dest,const char \*src)

将字符串src复制到dest

size\_t strcspn(const char \*s1,const char \*s2)

扫描s1,返回在s1中有,在s2中也有的字符个数

char strdup(const char \*s)

将字符串s复制到最近建立的单元

int stricmp(const char \*s1,const char \*s2)

比较字符串s1和s2,并返回s1-s2

size\_t strlen(const char \*s)

返回字符串s的长度

char strlwr(char \*s)

将字符串s中的大写字母全部转换成小写字母,并返回转换后的字符串

char strncat(char \*dest,const char \*src,size\_t maxlen)

将字符串src中最多maxlen个字符复制到字符串dest中

int strncmp(const char \*s1,const char \*s2,size\_t maxlen)

比较字符串s1与s2中的前maxlen个字符

char strncpy(char \*dest,const char \*src,size\_t maxlen)

复制src中的前maxlen个字符到dest中

int strnicmp(const char \*s1,const char \*s2,size\_t maxlen)

比较字符串s1与s2中的前maxlen个字符

char strnset(char \*s,int ch,size\_t n)

将字符串s的前n个字符置于ch中

char strpbrk(const char \*s1,const char \*s2)

扫描字符串s1,并返回在s1和s2中均有的字符个数

char strrchr(const char \*s,int c)

扫描最后出现一个给定字符c的一个字符串s

char strrev(char \*s)

将字符串s中的字符全部颠倒顺序重新排列,并返回排列后的字符串

char strset(char \*s,int ch)

将一个字符串s中的所有字符置于一个给定的字符ch

size\_t strspn(const char \*s1,const char \*s2)

扫描字符串s1,并返回在s1和s2中均有的字符个数

char strstr(const char \*s1,const char \*s2)

扫描字符串s2,并返回第一次出现s1的位置

char strtok(char \*s1,const char \*s2)

检索字符串s1,该字符串s1是由字符串s2中定义的定界符所分隔

char strupr(char \*s)

将字符串s中的小写字母全部转换成大写字母,并返回转换后的字符串

存贮分配子程序,所在函数库为dos.h、alloc.h、malloc.h、stdlib.h、process.h

void \*malloc(unsigned size)

分配size个字节的内存空间,并返回所分配内存的指针

void free(void \*ptr)释放先前所分配的内存,所要释放的内存的指针为ptr

void \*realloc(void \*ptr,unsigned newsize)改变已分配内存的大小,ptr为已分配有内存区域的指针,newsize为新的长度,返回分配好的内存指针.