C++ Lists（链表）

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
2.	back()	返回最后一个元素
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
6.	end()	返回末尾的迭代器
7.	erase()	删除一个元素
8.	front()	返回第一个元素
9.	get_allocator()	返回 list 的配置器
10.	insert()	插入一个元素到 list 中
11.	max_size()	返回 list 能容纳的最大元素数量
12.	merge()	合并两个 list
13.	pop_back()	删除最后一个元素
14.	pop_front()	删除第一个元素
15.	push_back()	在 list 的末尾添加一个元素
16.	push_front()	在 list 的头部添加一个元素
17.	rbegin()	返回指向第一个元素的逆向迭代器
18.	remove()	从 list 删除元素
19.	remove_if()	按指定条件删除元素
20.	rend()	指向 list 末尾的逆向迭代器
21.	resize()	改变 list 的大小
22.	reverse()	把 list 的元素倒转
23.	size()	返回 list 中的元素个数
24.	sort()	给 list 排序
25.	splice()	合并两个 list
26.	swap()	交换两个 list
27.	unique()	删除 list 中重复的元素

1. 赋值(assign)

语法:

```
void assign( input_iterator start, input_iterator end );
void assign( size_type num, const TYPE &val );
```

assign()函数以迭代器 start 和 end 指示的范围为 list 赋值或者为 list 赋值 num 个以 val 为值的元素。

2. back

语法:

```
reference back();
```

back()函数返回一个引用，指向 list 的最后一个元素。

3. begin

语法:

```
iterator begin();
```

begin()函数返回一个迭代器，指向 list 的第一个元素。

4. clear

语法:

```
void clear();
```

clear()函数删除 list 的所有元素。

5. empty

语法:

```
bool empty();
```

empty()函数返回真(true)如果链表为空，否则返回假。

6. end

语法:

```
iterator end();
```

end()函数返回一个迭代器，指向链表的末尾。

7. erase

语法:

```
iterator erase( iterator pos );
iterator erase( iterator start, iterator end );
```

erase()函数删除以 pos 指示位置的元素，或者删除 *start* 和 *end* 之间的元素。返回值是一个迭代器，指向最后一个被删除元素的下一个元素。

8. front

语法:

```
reference front();
```

front()函数返回一个引用，指向链表的第一个元素。

9. get_allocator

语法:

```
allocator_type get_allocator();
```

get_allocator()函数返回链表的配置器。

10. insert

语法:

```
iterator insert( iterator pos, const TYPE &val );
void insert( iterator pos, size_type num, const TYPE &val );
void insert( iterator pos, input_iterator start, input_iterator end );
```

insert()插入元素 val 到位置 pos，或者插入 num 个元素 val 到 pos 之前，或者插入 start 到 end 之间的元素到 pos 的位置。返回值是一个迭代器，指向被插入的元素。

11. max_size

语法:

```
size_type max_size();
```

max_size()函数返回链表能够储存的元素数目。

12. merge

语法:

```
void merge( list &lst );
void merge( list &lst, Comp compfunction );
```

merge()函数把自己和 lst 链表连接在一起，产生一个整齐排列的组合链表。如果指定 compfunction，则将指定函数作为比较的依据。

13. pop_back

语法:

```
void pop_back();
```

pop_back()函数删除链表的最后一个元素。

14. pop_front

语法:

```
void pop_front();
```

pop_front()函数删除链表的第一个元素。

15. push_back

语法:

```
void push_back( const TYPE &val );
```

push_back()将 val 连接到链表的最后。

16. push_front

Syntax:

```
void push_front( const TYPE &val );
```

push_front()函数将 val 连接到链表的头部。

17. rbegin

语法:

```
reverse_iterator rbegin();
```

rbegin()函数返回一个逆向迭代器，指向链表的末尾。

18. remove

语法:

```
void remove( const TYPE &val );
```

remove()函数删除链表中所有值为 val 的元素。

19. remove_if

语法:

```
void remove_if( UnPred pr );
```

remove_if()以一元谓词 pr 为判断元素的依据,遍历整个链表。如果 pr 返回 true 则删除该元素。

20. rend

语法:

```
reverse_iterator rend();
```

rend()函数迭代器指向链表的头部。

21. resize

语法:

```
void resize( size_type num, TYPE val );
```

resize()函数把 list 的大小改变到 *num*。被加入的多余的元素都被赋值为 val

22. reverse

语法:

```
void reverse();
```

reverse()函数把 list 所有元素倒转。

23. size

语法:

```
size_type size();
```

size()函数返回 list 中元素的数量。

24. 排序(sort)

语法:

```
void sort();
void sort( Comp compfunction );
```

sort()函数为链表排序，默认是升序。如果指定 compfunction 的话，就采用指定函数来判定两个元素的大小。

25. splice

语法:

```
void splice( iterator pos, list &lst );
void splice( iterator pos, list &lst, iterator del );
void splice( iterator pos, list &lst, iterator start, iterator end );
```

splice()函数把 lst 连接到 pos 的位置。如果指定其他参数，则插入 lst 中 del 所指元素到现链表的 pos 上，或者用 start 和 end 指定范围。

26. swap

语法:

```
void swap( list &lst );
```

swap()函数交换 lst 和现链表中的元素。

27. unique

语法:

```
void unique();
void unique( BinPred pr );
```

unique()函数删除链表中所有重复的元素。如果指定 pr，则使用 pr 来判定是否删除。