Introduction to OpenFOAM Programming

01 - C++ 数据结构 + 算法

王佳琪

上海交通大学

2022 年 1 月

动态内存分配

- 栈分配：分配发生在连续的内存块上。演示
- 堆分配：在执行程序员编写的指令期间分配内存，new/delete。演示

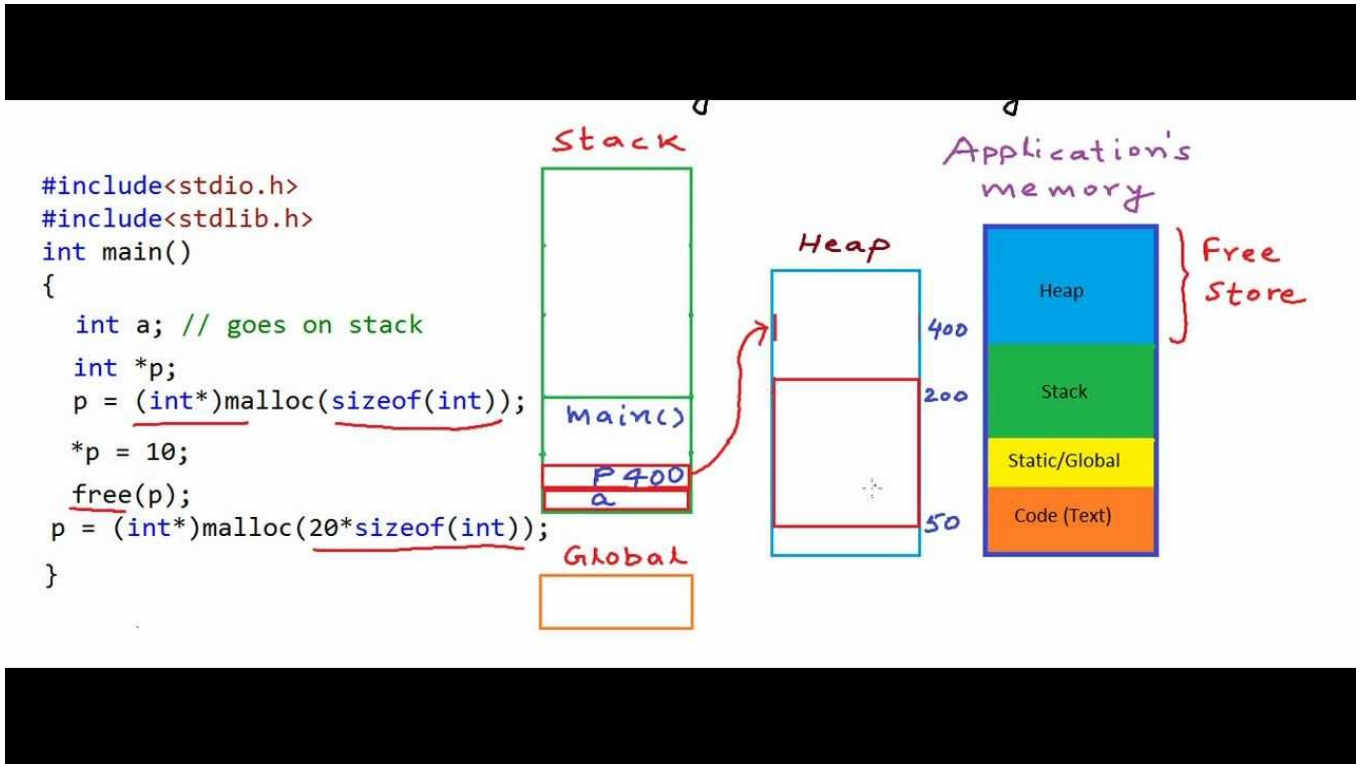


图 1: 动态内存分配机制

迭代器

- 迭代器被用来遍历对象集合中的元素。

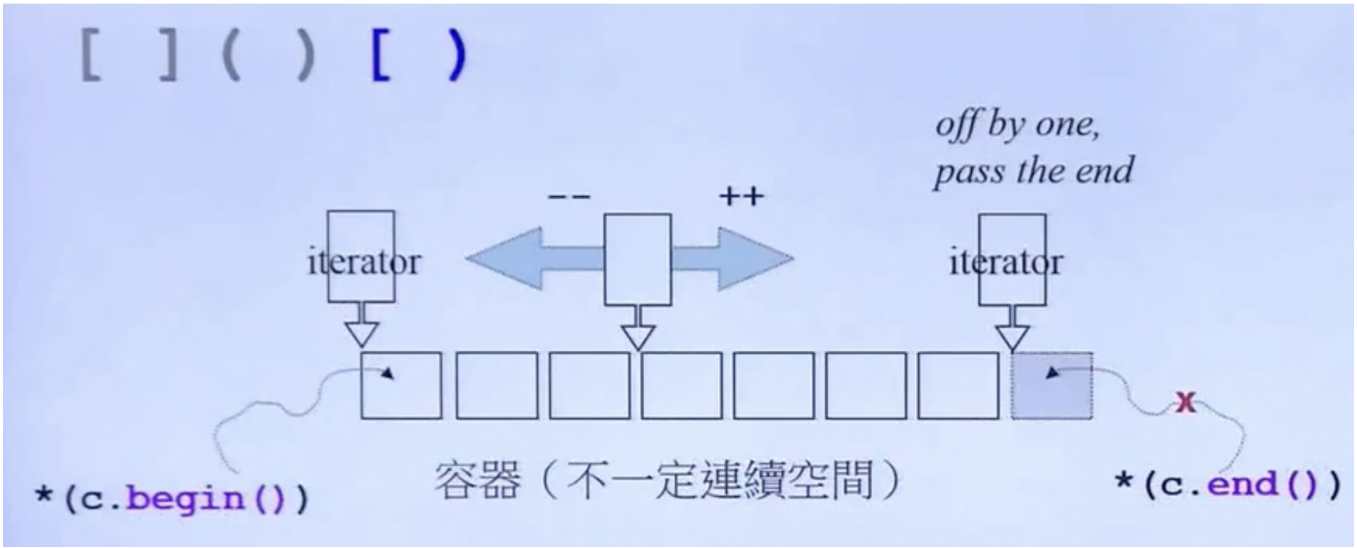


图 2: 迭代器的方向性

图解数据结构

- 线性数据结构：数组-Array/Vector、链表-List、栈-Stack、队列-Deque
- 非线性数据结构：树 (Map/Set)、堆-Heap、图-Graph、散列表-Hashing

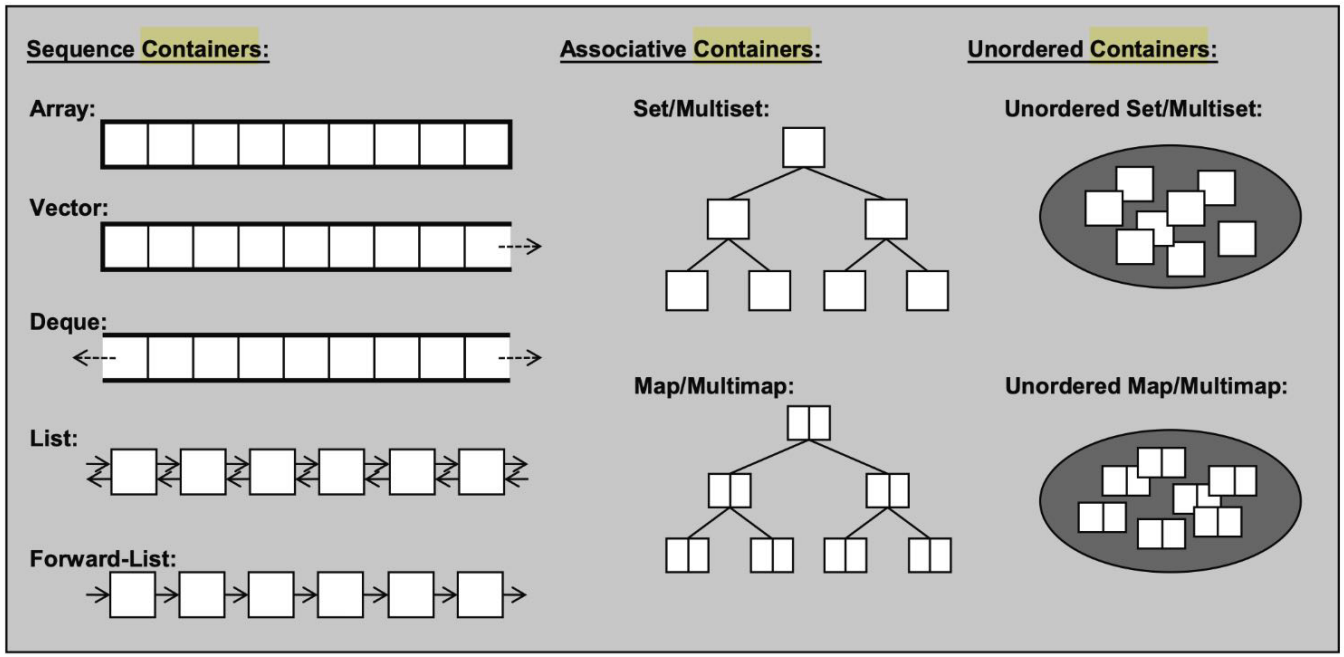


图 3: STL 基本数据结构

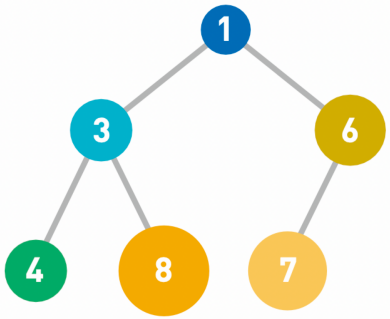
线性数据结构

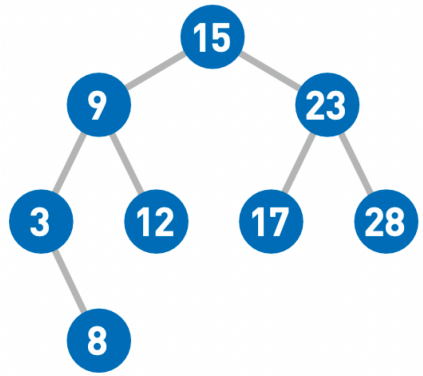
- 在链表中，数据的添加和删除都较为方便，就是访问比较耗费时间。
- 与链表不同，在数组中，访问数据十分简单，而添加和删除数据比较耗工夫。
- 栈也是一种数据呈线性排列的数据结构，只能访问最新添加的数据。
- 虽然与栈有些相似，但队列中添加和删除数据的操作分别是在两端进行的。

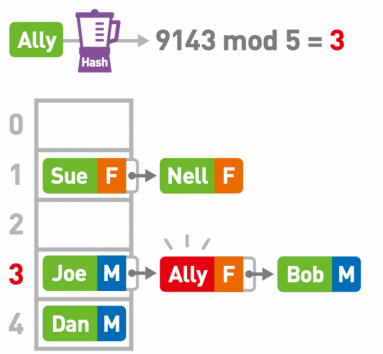
	访问	添加	删除
链表	慢	快	快
数组	快	慢	慢

非线性数据结构

- 堆是一种图的树形结构，被用于实现“优先队列”。优先队列是一种数据结构，可以自由添加数据，但取出数据时要从最小值开始按顺序取出。在 map 和 set 中体现。
- 二叉查找树 (又叫作二叉搜索树或二叉排序树)。在 map 和 set 中体现。
- 哈希表使用“哈希函数”，可以使数据的查询效率得到显著提升。结合数组和链表。







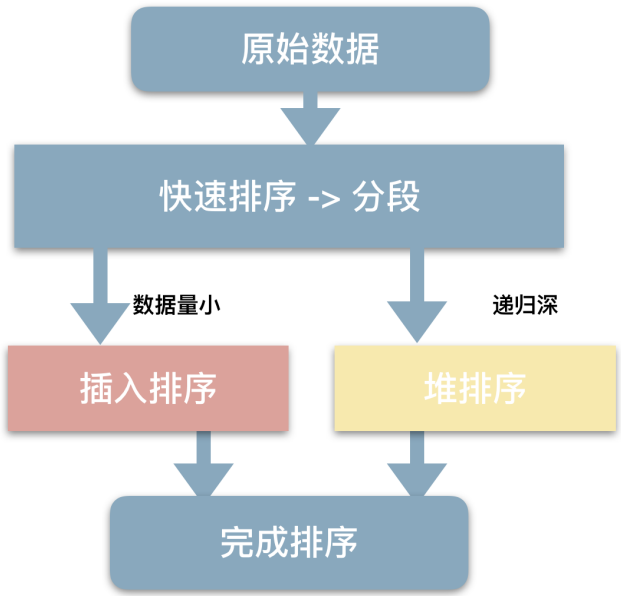
排序

- 哪些 STL 容器需要用到 sort 算法?
  - 首 先，关系型容器拥有自动排序功能，因为底层采用 RB-Tree，所以不需要用到 sort 算法。
  - 其 次，序列式容器中的 stack、queue 和 priority-queue 都有特定的出入口，不允许用户对元素排序。
  - 剩 下的 vector、deque、list，适用 sort 算法。
  - 常 见排序算法: bubbleSort、selectionSort、insertSort、heapSort、mergeSort、quickSort

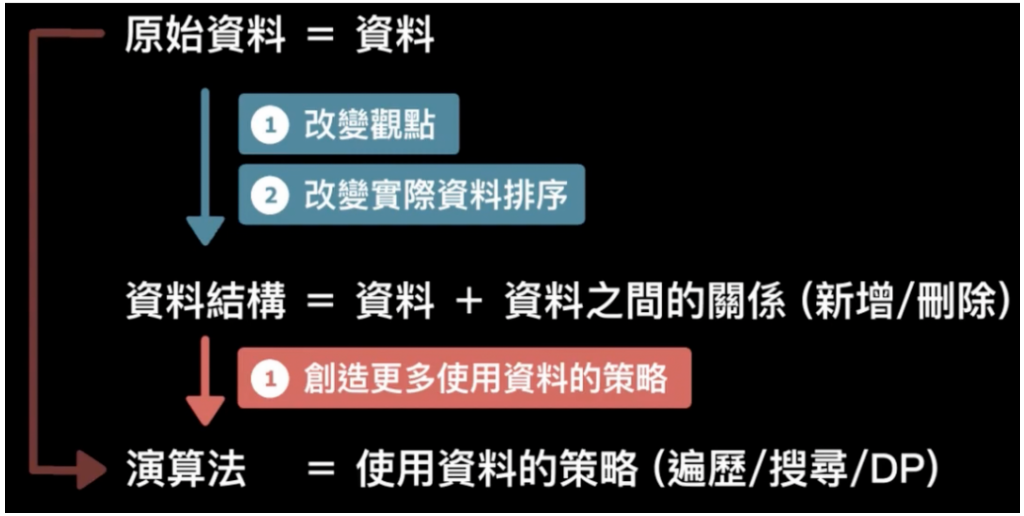
02-doc

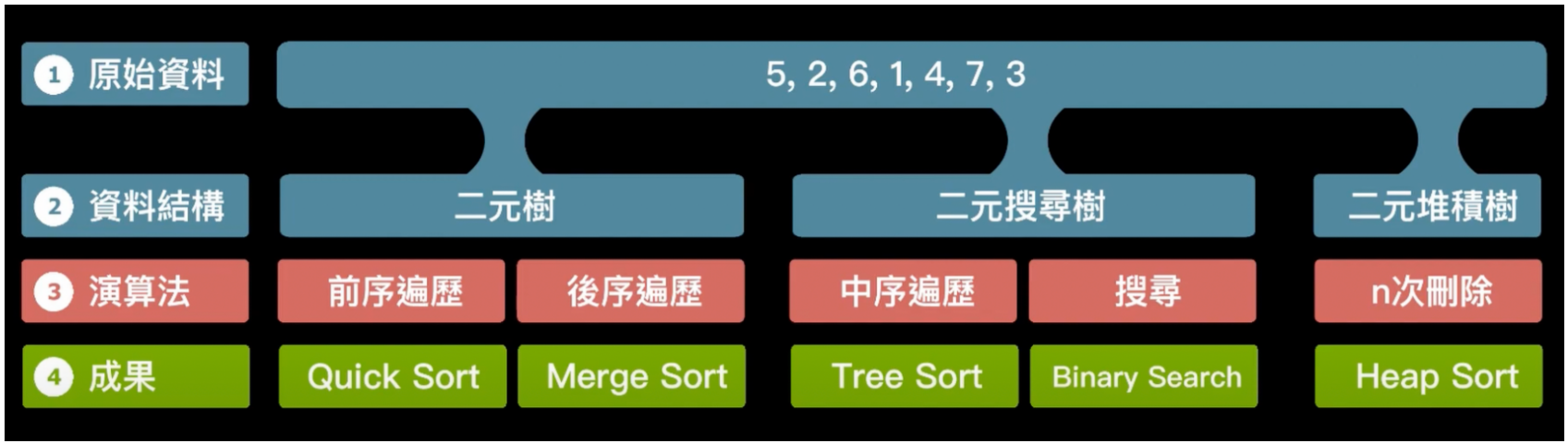
排序

- 实现逻辑
  - S TL 的 sort 算法，数据量大时采用快速排序算法，分段归并排序。一旦分段后的数据量小于某个门槛（16），为避免 QuickSort 快排的递归调用带来过大的额外负荷，就改用插入排序。如果递归层次过深，还会改用堆排序。



图解算法


原始資料 = 資料
1 改變觀點
2 改變實際資料排序
資料結構 = 資料 + 資料之間的關係 (新增/刪除)
1 創造更多使用資料的策略
演算法 = 使用資料的策略 (遍歷/搜尋/DP)


1 原始資料
5, 2, 6, 1, 4, 7, 3
2 資料結構
二元樹
二元搜尋樹
二元堆積樹
3 演算法
前序遍歷
後序遍歷
中序遍歷
搜尋
n次刪除
4 成果
Quick Sort
Merge Sort
Tree Sort
Binary Search
Heap Sort

00-Array/Vector

Example:

```
1 int array[5]; // 初始化
2 array[0] = 2; // 元素赋值
3 array[1] = 3;
4 array[2] = 1;
5 array[3] = 0;
6 array[4] = 2;
```

```
1 vector<int> array; // 初始化可变数组
2 array.push_back(2); // 向尾部添加元素
```

00-main-doc

01-vector::vector

```cpp
//main: <iostream> <vector>
std::vector<int> first;
std::vector<int> second (4,100);
std::vector<int> third (second.begin(),second.end());
std::vector<int> fourth (third);
int myints[] = {16,2,77,29};

auto pointer = myints + sizeof(myints) / sizeof(int); // second.end()
std::vector<int> fifth (myints, pointer);

std::cout << "The contents of fifth are:";
for (std::vector<int>::iterator it = fifth.begin(); it != fifth.end(); ++it)
  std::cout << ' ' << *it;
std::cout << '\n';
```

Output: The contents of fifth are: 16 2 77 29

01-gdb 02-doc

02-vector::assign

```cpp
//main: <iostream> <vector>
  std::vector<int> first;
  std::vector<int> second;
  std::vector<int> third;
  first.assign (7,100);             // 7 ints with a value of 100
  std::vector<int>::iterator it;
  it=first.begin()+1;
  second.assign (it,first.end()-1); // the 5 central values of first

  int myints[] = {1776,7,4};
  third.assign (myints,myints+3);   // assigning from array.
  std::cout << "Size of first: " << int (first.size()) << '\n';
  std::cout << "Size of second: " << int (second.size()) << '\n';
  std::cout << "Size of third: " << int (third.size()) << '\n';
```

Output: Size of first: 7 Size of second: 5 Size of third: 3

01-gdb 02-doc

03-vector::at

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector (10);   // 10 zero-initialized ints
3
4  // assign some values:
5  for (unsigned i=0; i<myvector.size(); i++)
6    myvector.at(i)=i;
7
8  std::cout << "myvector␣contains:";
9  for (unsigned i=0; i<myvector.size(); i++)
10   std::cout << '␣' << myvector.at(i);
11 std::cout << '\n';
```

Output: myvector contains: 0 1 2 3 4 5 6 7 8 9

01-gdb 02-doc

04-vector::back

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector;
3  myvector.push_back(10);
4  while (myvector.back() != 0)
5  {
6    myvector.push_back ( myvector.back() -1 );
7  }
8
9  std::cout << "myvector␣contains:";
10 for (unsigned i=0; i<myvector.size() ; i++)
11   std::cout << '␣' << myvector[i];
12 std::cout << '\n';
```

Output: myvector contains: 10 9 8 7 6 5 4 3 2 1 0

01-gdb 02-doc

05-vector::begin/end

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector;
3  for (int i=1; i<=5; i++) myvector.push_back(i);
4
5  std::cout << "myvector contains:";
6  for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end();
       ++it)
7    std::cout << ' ' << *it;
8  std::cout << '\n';
```

Output: myvector contains: 1 2 3 4 5

01-gdb 02-doc

06-vector::capacity

```cpp
//main: <iostream> <vector>
  std::vector<int> myvector;

  // set some content in the vector:
  for (int i=0; i<100; i++) myvector.push_back(i);

  std::cout << "size: " << (int) myvector.size() << '\n';
  std::cout << "capacity: " << (int) myvector.capacity() << '\n';
  std::cout << "max_size: " << (int) myvector.max_size() << '\n';
```

Output: size: 100 capacity: 128 max_size: 1073741823

01-gdb 02-doc

07-vector::cend

```cpp
//main: <iostream> <vector>
  std::vector<int> myvector = {10,20,30,40,50};

  std::cout << "myvector␣contains:";

  for (auto it = myvector.cbegin(); it != myvector.cend(); ++it)
    std::cout << '␣' << *it;
  std::cout << '\n';
```

Output: myvector contains: 10 20 30 40 50

01-gdb 02-doc

08-vector::clear

```
1  //main: <iostream> <vector>
2    std::vector<int> myvector;
3    myvector.push_back (100);
4    myvector.push_back (200);
5    myvector.push_back (300);
6
7    std::cout << "myvector␣contains:";
8    for (unsigned i=0; i<myvector.size(); i++)
9      std::cout << '␣' << myvector[i]; std::cout << '\n';
10
11   myvector.clear();
12   myvector.push_back (1101);
13   myvector.push_back (2202);
14   std::cout << "myvector␣contains:";
15   for (unsigned i=0; i<myvector.size(); i++)
16     std::cout << '␣' << myvector[i]; std::cout << '\n';
```

Output: myvector contains: 100 200 300 myvector contains: 1101 2202

01-gdb 02-doc

09-vector::crbegin/crend/rbegin/rend

```cpp
//main: <iostream> <vector>
  std::vector<int> myvector = {1,2,3,4,5};

  std::cout << "myvector backwards:";
  for (auto rit = myvector.crbegin(); rit != myvector.crend(); ++rit)
    std::cout << ' ' << *rit;
  std::cout << '\n';
```

Output: myvector backwards: 5 4 3 2 1

01-gdb 02-doc

10-vector::data

```
1  //main: <iostream> <vector>
2    std::vector<int> myvector (5);
3
4    int* p = myvector.data();
5
6    *p = 10;
7    ++p;
8    *p = 20;
9    p[2] = 100;
10
11   std::cout << "myvector␣contains:";
12   for (unsigned i=0; i<myvector.size(); ++i)
13     std::cout << '␣' << myvector[i];
14   std::cout << '\n';
```

Output: myvector contains: 10 20 0 100 0

01-gdb 02-doc

11-vector::emplace

```cpp
//main: <iostream> <vector>
  std::vector<int> myvector = {10,20,30};

  auto it = myvector.emplace ( myvector.begin()+1, 100 );
  myvector.emplace ( it, 200 );
  myvector.emplace ( myvector.end(), 300 );

  std::cout << "myvector␣contains:";
  for (auto& x: myvector)
    std::cout << '␣' << x;
  std::cout << '\n';
```

Output: myvector contains: 10 200 100 20 30 300

01-gdb 02-doc

12-vector::emplace_back

```cpp
std::vector<int> myvector = {10,20,30};

myvector.emplace_back (100);
myvector.emplace_back (200);

std::cout << "myvector␣contains:";
for (auto& x: myvector)
  std::cout << '␣' << x;
std::cout << '\n';
```

Output: myvector contains: 10 20 30 100 200

01-gdb 02-doc

13-vector::empty

```cpp
//main: <iostream> <vector>
std::vector<int> myvector;
int sum (0);

for (int i=1;i<=10;i++) myvector.push_back(i);

while (!myvector.empty())
{
    sum += myvector.back();
    myvector.pop_back();
}

std::cout << "total: " << sum << '\n';
```

Output: total: 55

01-gdb 02-doc

14-vector::erase

```
1  //main: <iostream> <vector>
2  // set some values (from 1 to 10)
3  for (int i=1; i<=10; i++) myvector.push_back(i);
4
5  // erase the 6th element
6  myvector.erase (myvector.begin()+5);
7
8  // erase the first 3 elements:
9  myvector.erase (myvector.begin(),myvector.begin()+3);
10
11 std::cout << "myvector␣contains:";
12 for (unsigned i=0; i<myvector.size(); ++i)
13   std::cout << '␣' << myvector[i];
14 std::cout << '\n';
```

Output: myvector contains: 4 5 7 8 9 10

01-gdb 02-doc

15-vector::front

```cpp
//main: <iostream> <vector>
std::vector<int> myvector;

myvector.push_back(78);
myvector.push_back(16);

// now front equals 78, and back 16

myvector.front() -= myvector.back();

std::cout << "myvector.front() is now " << myvector.front() << '\n';
```

Output: myvector.front() is now 62

01-gdb 02-doc

16-vector::get_allocator

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector;
3  int * p;
4  unsigned int i;
5
6  // allocate an array with space for 5 elements using vector's allocator:
7  p = myvector.get_allocator().allocate(5);
8  // construct values in-place on the array:
9  for (i=0; i<5; i++) myvector.get_allocator().construct(&p[i],i);
10
11 std::cout << "The allocated array contains:";
12 for (i=0; i<5; i++) std::cout << ' ' << p[i];
13 std::cout << '\n';
14
15 // destroy and deallocate:
16 for (i=0; i<5; i++) myvector.get_allocator().destroy(&p[i]);
17 myvector.get_allocator().deallocate(p,5);
```

Output: The allocated array contains: 0 1 2 3 4

01-gdb 02-doc

17-vector::insert

```cpp
//main: <iostream> <vector>
std::vector<int> myvector (3,100);
std::vector<int>::iterator it;

it = myvector.begin();
it = myvector.insert ( it , 200 );
myvector.insert (it,2,300);

it = myvector.begin();   // "it" no longer valid, get a new one
std::vector<int> anothervector (2,400);
myvector.insert (it+2,anothervector.begin(),anothervector.end());
int myarray [] = { 501,502,503 };
myvector.insert (myvector.begin(), myarray, myarray+3);
std::cout << "myvector contains:";
for (it=myvector.begin(); it<myvector.end(); it++)
  std::cout << ' ' << *it;
std::cout << '\n';
```

Output: myvector contains: 501 502 503 300 300 400 400 200 100 100 100

01-gdb 02-doc

18-vector::operator=

```
1  //main: <iostream> <vector>
2  std::vector<int> foo (3,0);
3  std::vector<int> bar (5,0);
4
5  bar = foo;
6  foo = std::vector<int>();
7
8  std::cout << "Size of foo: " << int(foo.size()) << '\n';
9  std::cout << "Size of bar: " << int(bar.size()) << '\n';
```

Output: Size of foo: 0 Size of bar: 3

01-gdb 02-doc

19-vector::operator[]

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector (10);   // 10 zero-initialized elements
3  std::vector<int>::size_type sz = myvector.size();
4  // assign some values:
5  for (unsigned i=0; i<sz; i++) myvector[i]=i;
6  // reverse vector using operator[]:
7  for (unsigned i=0; i<sz/2; i++)
8  {
9    int temp;
10   temp = myvector[sz-1-i];
11   myvector[sz-1-i]=myvector[i];
12   myvector[i]=temp;
13 }
14 std::cout << "myvector␣contains:";
15 for (unsigned i=0; i<sz; i++)
16   std::cout << '␣' << myvector[i];
17 std::cout << '\n';
```

Output: myvector contains: 9 8 7 6 5 4 3 2 1 0

01-gdb 02-doc

20-vector::pop_back/push_back

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector;
3  int sum (0);
4  myvector.push_back (100);
5  myvector.push_back (200);
6  myvector.push_back (300);
7
8  while (!myvector.empty())
9  {
10   sum+=myvector.back();
11   myvector.pop_back();
12 }
13 std::cout << "The elements of myvector add up to" << sum << '\n';
```

Output: The elements of myvector add up to 600

01-gdb 02-doc

21-vector::reserve

```
1  std::vector<int>::size_type sz; std::vector<int> foo;
2  sz = foo.capacity();
3  std::cout << "making foo grow:\n";
4  for (int i=0; i<100; ++i) {
5    foo.push_back(i);
6    if (sz!=foo.capacity()) {
7      sz = foo.capacity();
8      std::cout << "capacity changed: " << sz << '\n';
9    }
10 }
11 std::vector<int> bar; sz = bar.capacity();
12 bar.reserve(100);   // only difference with foo above
13 std::cout << "making bar grow:\n";
14 for (int i=0; i<100; ++i) {
15   bar.push_back(i);
16   if (sz!=bar.capacity()) {
17     sz = bar.capacity();
18     std::cout << "capacity changed: " << sz << '\n';
19   }
20 }
```

Output: making foo grow: capacity changed: 1 capacity changed: 2 capacity changed: 4 capacity changed: 8 capacity changed: 16 capacity changed: 32 capacity changed: 64 capacity changed: 128 making bar grow: capacity changed: 100

01-gdb 02-doc

22-vector::resize

```
1  //main: <iostream> <vector>
2  std::vector<int> myvector;
3  // set some initial content:
4  for (int i=1;i<10;i++) myvector.push_back(i);
5  myvector.resize(5);
6  myvector.resize(8,100);
7  myvector.resize(12);
8
9  std::cout << "myvector␣contains:";
10 for (int i=0;i<myvector.size();i++)
11   std::cout << '␣' << myvector[i];
12 std::cout << '\n';
```

Output: myvector contains: 1 2 3 4 5 100 100 100 0 0 0 0

01-gdb 02-doc

23-vector::shrink_to_fit

```
1  //main: <iostream> <vector>
2    std::vector<int> myvector (100);
3    std::cout << "1. capacity of myvector: " << myvector.capacity() << '\n';
4
5    myvector.resize(10);
6    std::cout << "2. capacity of myvector: " << myvector.capacity() << '\n';
7
8    myvector.shrink_to_fit();
9    std::cout << "3. capacity of myvector: " << myvector.capacity() << '\n';
```

Output: 1 capacity of myvector: 100 2 capacity of myvector: 100 3 capacity of myvector: 10

01-gdb 02-doc

24-vector::size

```
1  //main: <iostream> <vector>
2  std::vector<int> myints;
3  std::cout << "0. size: " << myints.size() << '\n';
4
5  for (int i=0; i<10; i++) myints.push_back(i);
6  std::cout << "1. size: " << myints.size() << '\n';
7
8  myints.insert (myints.end(),10,100);
9  std::cout << "2. size: " << myints.size() << '\n';
10
11 myints.pop_back();
12 std::cout << "3. size: " << myints.size() << '\n';
```

Output: 0. size: 0 1. size: 10 2. size: 20 3. size: 19

01-gdb 02-doc

25-vector::swap

```
1  //main: <iostream> <vector>
2  std::vector<int> foo (3,100);   // three ints with a value of 100
3  std::vector<int> bar (5,200);   // five ints with a value of 200
4
5  foo.swap(bar);
6
7  std::cout << "foo␣contains:";
8  for (unsigned i=0; i<foo.size(); i++)
9    std::cout << '␣' << foo[i];
10 std::cout << '\n';
11 std::cout << "bar␣contains:";
12 for (unsigned i=0; i<bar.size(); i++)
13   std::cout << '␣' << bar[i];
14 std::cout << '\n';
```

Output: foo contains: 200 200 200 200 200 bar contains: 100 100 100

01-gdb 02-doc

00-List

相比 vector 和 array 容器，List 容器依据其链表的特点，多出以下成员函数：emplace_front、pop_front、merge、reverse、remove、remove_if、sort、splice、unique

01-list::list

```
1  // 为更好的理解链表，建议利用gdb调试，
2  // 对比vector::vector，仅仅为数据结构发生变化。
3    //main: <iostream> <list>
4    std::list<int> first;
5    std::list<int> second (4,100);
6    std::list<int> third (second.begin(),second.end());
7    std::list<int> fourth (third);
8
9    int myints[] = {16,2,77,29};
10   // end pointer location
11   auto pointer = myints + sizeof(myints) / sizeof(int);
12   std::list<int> fifth (myints, pointer );
13
14   std::cout << "The contents of fifth are: ";
15   for (std::list<int>::iterator it = fifth.begin(); it != fifth.end(); it++)
16     std::cout << *it << ' ';
17   std::cout << '\n';
```

Output: The contents of fifth are: 16 2 77 29

01-gdb 02-doc

映射是关联容器，它按照特定顺序存储由键值和映射值组合形成的元素。

在映射中，键值通常用于对元素进行排序和唯一标识，而映射值存储与该键关联的内容。键和映射值的类型可能不同，并在成员类型 value_type 中组合在一起，这是一个结合了两者的对类型：

```
1    typedef pair<const Key, T> value_type;
```

01-map::map

```
1  bool fncomp (char lhs, char rhs) {return lhs<rhs;}
2  struct classcomp {
3    bool operator() (const char& lhs, const char& rhs) const
4    {return lhs<rhs;}
5  };
6  // main: <iostream> <map>
7    std::map<char,int> first;
8    first['a']=10;
9    first['b']=30;
10   first['c']=50;
11   first['d']=70;
12   std::map<char,int> second (first.begin(),first.end());
13   std::map<char,int> third (second);
14   std::map<char,int,classcomp> fourth;                 // class as
         Compare
15   bool(*fn_pt)(char,char) = fncomp;
16   std::map<char,int,bool(*)(char,char)> fifth (fn_pt); // function
         pointer as Compare
```

Output: XXX

01-gdb 02-doc

02-map::at

```cpp
// main: <iostream> <string> <map>
std::map<std::string, int> mymap = {
                { "alpha", 0 },
                { "beta", 0 },
                { "gamma", 0 } };

mymap.at("alpha") = 10;
mymap.at("beta") = 20;
mymap.at("gamma") = 30;

for (auto& x: mymap) {
  std::cout << x.first << ": " << x.second << '\n';
}
```

Output: alpha: 10 beta: 20 gamma: 30

01-gdb 02-doc

03-map::begin

```
1  // main: <iostream> <string> <map>
2  std::map<char,int> mymap;
3
4  mymap['b'] = 100;
5  mymap['a'] = 200;
6  mymap['c'] = 300;
7
8  // show content:
9  for (std::map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
10   std::cout << it->first << " => " << it->second << '\n';
```

Output: a => 200 b => 100 c => 300

01-gdb 02-doc