Introduction to OpenFOAM Programming

01 - C++ 数据结构

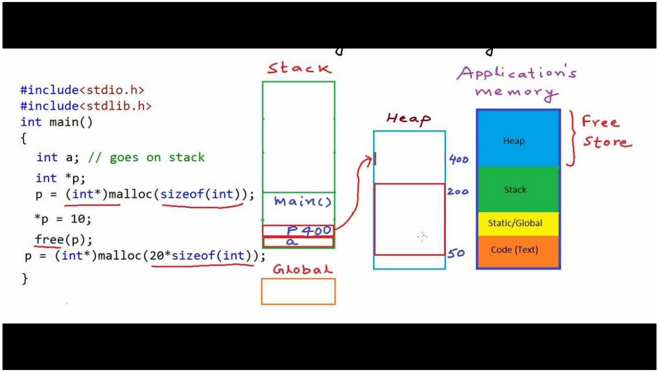
王佳琪

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



```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a; // goes on stack
   int *p;
   p = (int*)malloc(sizeof(int));
   *p = 10;
   free(p);
  p = (int*)malloc(20*sizeof(int));
}
```

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。

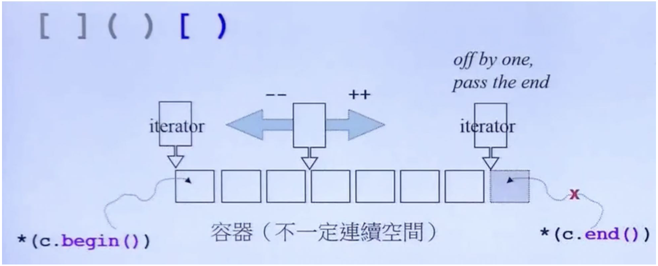


图 2: 迭代器的方向性

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

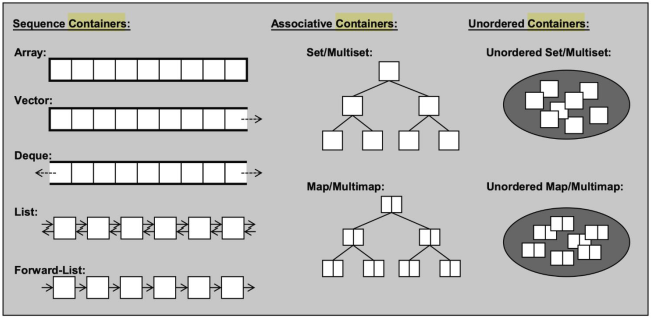


图 3: STL 基本数据结构

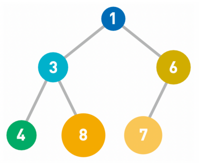
线性数据结构

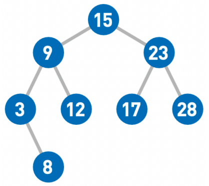
- 在链表中，数据的添加和删除都较为方便，就是访问比较耗费时间。
- 与链表不同，在数组中，访问数据十分简单，而添加和删除数据比较耗工夫。
- 栈也是一种数据呈线性排列的数据结构，只能访问最新添加的数据。
- 虽然与栈有些相似，但队列中添加和删除数据的操作分别是在两端进行的。

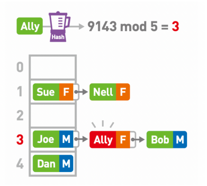
	访问	添加	删除
链表	慢	快	快
数组	快	慢	慢

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。







00-Array/Vector

Example:

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

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

00-main-doc

01-vector::vector

```
//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

```
//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

```cpp
//main: <iostream> <vector>
std::vector<int> myvector (10);   // 10 zero-initialized ints

// assign some values:
for (unsigned i=0; i<myvector.size(); i++)
  myvector.at(i)=i;

std::cout << "myvector␣contains:";
for (unsigned i=0; i<myvector.size(); i++)
  std::cout << '␣' << myvector.at(i);
std::cout << '\n';
```

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

01-gdb 02-doc

04-vector::back

```
//main: <iostream> <vector>
std::vector<int> myvector;
myvector.push_back(10);
while (myvector.back() != 0)
{
  myvector.push_back ( myvector.back() -1 );
}

std::cout << "myvector␣contains:";
for (unsigned i=0; i<myvector.size() ; i++)
  std::cout << '␣' << myvector[i];
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

```
//main: <iostream> <vector>
std::vector<int> myvector;
for (int i=1; i<=5; i++) myvector.push_back(i);

std::cout << "myvector␣contains:";
for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end();
    ++it)
  std::cout << '␣' << *it;
std::cout << '\n';
```

Output: myvector contains: 1 2 3 4 5

01-gdb 02-doc

06-vector::capacity

```
  //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

```
  //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

```
    //main: <iostream> <vector>
    std::vector<int> myvector;
    myvector.push_back (100);
    myvector.push_back (200);
    myvector.push_back (300);

    std::cout << "myvector␣contains:";
    for (unsigned i=0; i<myvector.size(); i++)
      std::cout << '␣' << myvector[i]; std::cout << '\n';

    myvector.clear();
    myvector.push_back (1101);
    myvector.push_back (2202);
    std::cout << "myvector␣contains:";
    for (unsigned i=0; i<myvector.size(); i++)
      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

```
    //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

```
  //main: <iostream> <vector>
   std::vector<int> myvector (5);

   int* p = myvector.data();

   *p = 10;
   ++p;
   *p = 20;
   p[2] = 100;

   std::cout << "myvector contains:";
   for (unsigned i=0; i<myvector.size(); ++i)
     std::cout << ' ' << myvector[i];
   std::cout << '\n';
```

Output: myvector contains: 10 20 0 100 0

01-gdb 02-doc

11-vector::emplace

```
    //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

```
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

```
//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

```
//main: <iostream> <vector>
// set some values (from 1 to 10)
for (int i=1; i<=10; i++) myvector.push_back(i);

// erase the 6th element
myvector.erase (myvector.begin()+5);

// erase the first 3 elements:
myvector.erase (myvector.begin(),myvector.begin()+3);

std::cout << "myvector␣contains:";
for (unsigned i=0; i<myvector.size(); ++i)
  std::cout << '␣' << myvector[i];
std::cout << '\n';
```

Output: myvector contains: 4 5 7 8 9 10

01-gdb 02-doc

15-vector::front

```
//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

```
//main: <iostream> <vector>
std::vector<int> myvector;
int * p;
unsigned int i;

// allocate an array with space for 5 elements using vector's allocator:
p = myvector.get_allocator().allocate(5);
// construct values in-place on the array:
for (i=0; i<5; i++) myvector.get_allocator().construct(&p[i],i);

std::cout << "The␣allocated␣array␣contains:";
for (i=0; i<5; i++) std::cout << '␣' << p[i];
std::cout << '\n';

// destroy and deallocate:
for (i=0; i<5; i++) myvector.get_allocator().destroy(&p[i]);
myvector.get_allocator().deallocate(p,5);
```

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

01-gdb 02-doc

17-vector::insert

```
//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=

```
//main: <iostream> <vector>
std::vector<int> foo (3,0);
std::vector<int> bar (5,0);

bar = foo;
foo = std::vector<int>();

std::cout << "Size of foo: " << int(foo.size()) << '\n';
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[]

```
//main: <iostream> <vector>
std::vector<int> myvector (10);   // 10 zero-initialized elements
std::vector<int >::size_type sz = myvector.size();
// assign some values:
for (unsigned i=0; i<sz; i++) myvector[i]=i;
// reverse vector using operator[]:
for (unsigned i=0; i<sz/2; i++)
{
  int temp;
  temp = myvector[sz-1-i];
  myvector[sz-1-i]=myvector[i];
  myvector[i]=temp;
}
std::cout << "myvector contains:";
for (unsigned i=0; i<sz; i++)
  std::cout << ' ' << myvector[i];
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

```
//main: <iostream> <vector>
std::vector<int> myvector;
int sum (0);
myvector.push_back (100);
myvector.push_back (200);
myvector.push_back (300);

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

```
std::vector<int>::size_type sz; std::vector<int> foo;
sz = foo.capacity();
std::cout << "making␣foo␣grow:\n";
for (int i=0; i<100; ++i) {
  foo.push_back(i);
  if (sz!=foo.capacity()) {
    sz = foo.capacity();
    std::cout << "capacity␣changed:␣" << sz << '\n';
  }
}
std::vector<int> bar; sz = bar.capacity();
bar.reserve(100);   // only difference with foo above
std::cout << "making␣bar␣grow:\n";
for (int i=0; i<100; ++i) {
  bar.push_back(i);
  if (sz!=bar.capacity()) {
    sz = bar.capacity();
    std::cout << "capacity␣changed:␣" << sz << '\n';
  }
}
```

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

```cpp
//main: <iostream> <vector>
std::vector<int> myvector;
// set some initial content:
for (int i=1;i<10;i++) myvector.push_back(i);
myvector.resize(5);
myvector.resize(8,100);
myvector.resize(12);

std::cout << "myvector␣contains:";
for (int i=0;i<myvector.size();i++)
  std::cout << '␣' << myvector[i];
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

```
    //main: <iostream> <vector>
      std::vector<int> myvector (100);
      std::cout << "1. capacity of myvector: " << myvector.capacity() << '\n';

      myvector.resize(10);
      std::cout << "2. capacity of myvector: " << myvector.capacity() << '\n';

      myvector.shrink_to_fit();
      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

```
//main: <iostream> <vector>
std::vector<int> myints;
std::cout << "0. size: " << myints.size() << '\n';

for (int i=0; i<10; i++) myints.push_back(i);
std::cout << "1. size: " << myints.size() << '\n';

myints.insert (myints.end(),10,100);
std::cout << "2. size: " << myints.size() << '\n';

myints.pop_back();
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

```
//main: <iostream> <vector>
std::vector<int> foo (3,100);   // three ints with a value of 100
std::vector<int> bar (5,200);   // five ints with a value of 200

foo.swap(bar);

std::cout << "foo␣contains:";
for (unsigned i=0; i<foo.size(); i++)
  std::cout << '␣' << foo[i];
std::cout << '\n';
std::cout << "bar␣contains:";
for (unsigned i=0; i<bar.size(); i++)
  std::cout << '␣' << bar[i];
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

```
// 为更好的理解链表，建议利用gdb调试，
// 对比vector::vector，仅仅为数据结构发生变化。
  //main: <iostream> <list>
  std::list<int> first;
  std::list<int> second (4,100);
  std::list<int> third (second.begin(),second.end());
  std::list<int> fourth (third);

  int myints[] = {16,2,77,29};
  // end pointer location
  auto pointer = myints + sizeof(myints) / sizeof(int);
  std::list<int> fifth (myints, pointer );

  std::cout << "The␣contents␣of␣fifth␣are:␣";
  for (std::list<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