xjyxp01

随笔 - 168 文章 - 0 评论 - 3 阅读 - 58780

昵称： xjyxp01
园龄： 2年9个月
粉丝： 3
关注： 17
+加关注

< 2022年3月 >

日	一	二	三	四	五	六
27	28	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

最新随笔
1.IO口多路复用
2.C++ getline函数用法详解
3.C++中stringstream的使用方法和样例
4.C++中的 istringstream 的用法

常用数据结构STL

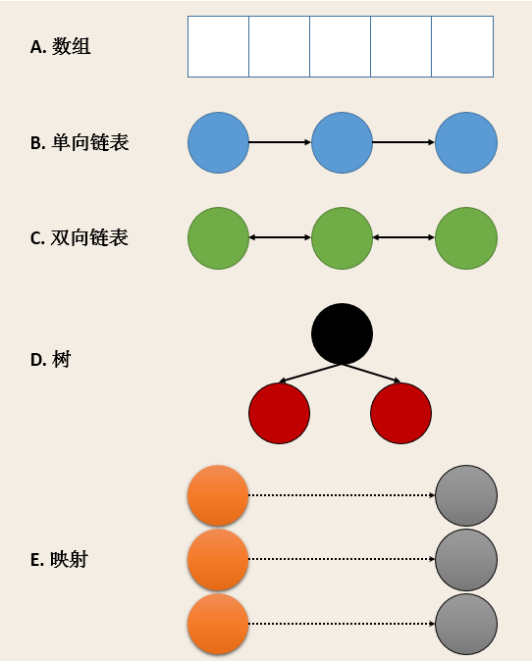
目录

简介

程序员的世界里有一个经典的公式： 数据结构+算法=程序。

所以数据结构及算法的重要性就不用在此赘述了，下面直接进入正题。

在物理层面有以下五种常见的数据结构：



一、数组

1. 静态数组

在编译期确定数组大小，在运行期无法改变数组大小，所以称之为静态数组。

C++ 中的 array 由这种结构实现

```
int main ()
{
    array<int, 10> a = {0,1,2,3,4,5,6,7,8,9};
    a[0] = 22;
    cout<<a.at(0)<<endl;
    cout<<a.back()<<endl;
</span><span style="color: #0000ff;">return</span> <span style="color: #800080;">0</span><span style="color: #000000;">;


}
```

2. 动态数组

在运行期可动态改变数组大小，所以称之为动态数组。C++ 中的动态数组有两个，分别是 vector 和 deque。

2.1. vector

矢量，只能在末尾增删元素

数组大小的增长策略：每次增加的长度为原来的1倍（有些编译器增加0.5倍）。

这样可以保证增加元素的平均时间复杂度为O(1)。

```
int main ()
{
```

2. 学习经验总结 | C++后台开发/云计算方向，offer收割机的学习路线(1)


```
    vector<int> vv = {1,2,3,4};
    vv.push_back(12);// 在末尾添加元素
    vv.pop_back();// 在末尾删除元素
    vv.at(3);// 读取第三个元素
    vv[3];// 读写第三个元素
    vv.insert(vv.begin()+3, 12);// 将元素插到第三个位置上
```

```
 <span style="color: #0000ff;">return</span> <span style="color:
 #800080;">0</span><span style="color: #000000;">;
```

```
}
```

2.2. priority_queue

优先队列，默认由 vector 实现，也可由 deque 实现。它保证顶部元素始终是最大值，可用于实现**堆排序。**

```
int main ()
{
    priority_queue<int> pp;
    pp.push(12);
    pp.push(10);
    pp.push(11);
    pp.top();// 读取顶部元素
    pp.pop();// 弹出顶部元素
```

```
 <span style="color: #0000ff;">return</span> <span style="color:
 #800080;">0</span><span style="color: #000000;">;
```

```
}
```

2.3. deque

双端队列，可以在开头或末尾增删元素

```
int main ()
{
    deque<int> dd;
    dd.push_front(12);// 在开头添加元素
    dd.push_back(10);// 在结尾添加元素
    dd.insert(dd.begin()+1, 3);// 在位置1插入元素
    dd.front();// 读取开头元素
    dd.back();// 读取结尾元素
    dd[1]; // 读取第一个元素
    dd.pop_front();// 弹出开头元素
    dd.pop_back();// 弹出末尾元素
```

```
 <span style="color: #0000ff;">return</span> <span style="color:
 #800080;">0</span><span style="color: #000000;">;
```

```
}
```

2.4. stack

栈，默认由 deque 实现，也可由 list 或 vector 实现。是一种**先进后出**的数据结构

```
int main ()
{
    stack<int> ss;
    ss.push(12);// 添加元素
    ss.top();// 读取栈顶元素
    ss.pop();// 弹出栈顶元素
 <span style="color: #0000ff;">return</span> <span style="color:
 #800080;">0</span><span style="color: #000000;">;


}
```

2.5. queue

队列，默认由 deque实现，也可由 list 实现。是一种**先进先出**的数据结构

```
int main ()
{
    queue<int> qq;
    qq.push(12);// 添加元素
    qq.front();// 读取队首元素
    qq.pop();// 弹出队首元素
 <span style="color: #0000ff;">return</span> <span style="color:
 #800080;">0</span><span style="color: #000000;">;


}
```

二、单向链表

B. 单向链表 

forward_list

只能从头到尾顺序遍历，不能逆序遍历，即没有 rbegin() 接口

```
int main ()
{
    forward_list<int> fl;
    fl.push_front(12);// 在开头添加元素
    fl.insert_after(fl.begin(), 11);
    fl.pop_front();// 在开头删除元素
    fl.remove(11);// 删除元素
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

三、双向链表

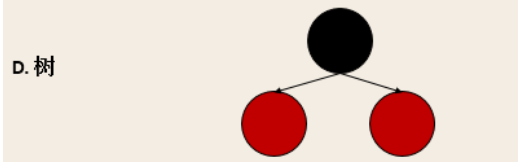
list

```
int main ()
{
    list<int> ll;
    ll.push_back(12);// 在末尾添加元素
    ll.push_front(10);// 在开头添加元素
    ll.back();// 读取末尾元素
    ll.front();// 读取开头元素
    ll.push_back(12);
    ll.unique();// 删除重复元素
    cout<<ll.size()<<endl;
    ll.pop_front();// 在末尾删除元素
    ll.pop_back();// 在开头删除元素
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

四、树



常见的树有二叉树、二叉搜索树、二叉平衡树、红黑树等。

C++ 中的 set multiset map multimap 是用二叉搜索树实现的，这种数据结构支持二分搜索，所以增删改查的复杂度都是O(logn)。

1. set

类似数学中的集合，set 中不能包含重复的元素，元素是排好序的，且不能被修改。

```
int main ()
{
```

```
    set<int, less<int>> ss;// 由小到大排序
    ss.insert(12);
    ss.insert(10);
    for(auto itr=ss.cbegin(); itr!=ss.cend(); itr++)cout<<*itr<<endl;// 输出 10 12
    ss.erase(ss.cbegin());// 擦除首元素
    ss.count(13);// 元素 13 的个数，0 或 1
    ss.find(10);// 查找元素 12，返回迭代器，若没找到返回 ss.end()
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

2. multiset

与 set 类似，但可以包含重复元素

```
int main ()
{
    multiset<int, less<int>> ms;// 由小到大排序
    ms.insert(12);
    ms.insert(10);
    ms.insert(10);
    for(auto itr=ms.cbegin(); itr!=ms.cend(); itr++)cout<<*itr<<endl;// 输出 10 10 1
    cout<<""<<endl;
    auto pp = ms.equal_range(10);
    for(auto itr=pp.first; itr!=pp.second; itr++)cout<<*itr<<endl;// 输出 10 10
    ms.lower_bound(10);// = pp.first
    ms.upper_bound(10);// = pp.second
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

3. map

元素由 (key,value) 对组成，接口与 set 类似，在插入与遍历元素时有些区别

```
int main ()
{
    map<int, int> mm;
    mm[1] = 1;// 插入元素 (1,1)
    mm.insert(make_pair(2,2));// 插入元素 (2,2)
    for(auto itr=mm.cbegin(); itr!=mm.cend(); itr++)
        cout<<"("<<itr->first<<","<<itr->second<<")"<<endl;// 输出 (1,1) (2,2)
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

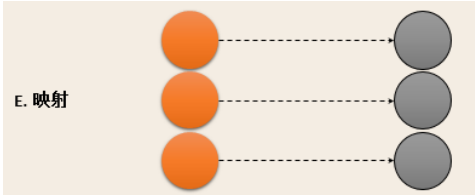
4. multimap

```
int main ()
{
    multimap<int, int> mm;
    mm.insert(make_pair(1,10));// 插入元素 (1,10)
    mm.insert(make_pair(1,11));// 插入元素 (1,11)
    mm.insert(make_pair(2,2));// 插入元素 (2,2)
    for(auto itr=mm.cbegin(); itr!=mm.cend(); itr++)// 遍历所有元素
        cout<<"("<<itr->first<<","<<itr->second<<")"<<endl;// 输出 (1,10) (1,11) (2,
    for(auto itr=mm.lower_bound(1); itr!=mm.upper_bound(1); itr++)// 遍历 key=1 的元素
        cout<<"("<<itr->first<<","<<itr->second<<")"<<endl;// 输出 (1,10) (1,11)
    mm.erase(1);// 删除所有 key=1 的元素
    mm.erase(mm.cbegin());// 删除第一个元素
```

```
<span style="color: #0000ff;">return</span> <span style="color:
#800080;">0</span><span style="color: #000000;">;
```

```
}
```

五、映射



映射类似数学中的函数，每一个 key 对应一个 value，写成函数表达式为：value=f(key)，其中 f 被称为哈希函数。

C++11 中的 unordered_set unordered_multiset unordered_map unordered_multimap 是用映射实现的，这种数据结构可以在O(1)的时间复杂度下访问单个元素，效率高于二叉搜索树(O(logn))，但是遍历元素的效率比二叉搜索树低。

1. unordered_set

接口与 set 类似，不在赘述

2. unordered_multiset

接口与 multiset 类似，不在赘述

3. unordered_map

接口与 map 类似，不在赘述

4. unordered_multimap

接口与 multimap 类似，不在赘述

分类: C++中STL学习

分类: C++, 数据结构, 算法

标签: 数据结构, 算法, C++