

测试报告

葛骏

2022-09-28

测试报告

编译 & 测试环境

我的环境是 Ubuntu 22.04 LTS, kernel 5.15.0

使用的编译器是 clang++

文件中附带了 makefile 请根据内容修改编译器。

基本功能 curd

- 增加 (pushBack, insert)
- 删除 (popBack, erase)
- 修改 (修改某一个特定值)
- 查询 (打印)

上机实验添加代码

- 删除 (remove): 通过数据删除
- 删除全部 (removeAll): 通过数据全部删除

测试代码:

输出截图

其他功能

- 排序 直接调用 STL 的 sort 函数
- 是否有序

```
clang++ main.cpp -o vector.out
insert 10 random numbers
Now data:618 127 417 966 547 54 142 20 90 506
size: 10
capacity: 16

insert 2333 at index 3
Now data:618 127 417 2333 966 547 54 142 20 90 506
size: 11
capacity: 16

erase the element at index 4
Now data:618 127 417 2333 547 54 142 20 90 506
size: 10
capacity: 16

set the element at index 5 to 666
Now data:618 127 417 2333 547 666 142 20 90 506
size: 10
capacity: 16

Is the vector sorted? No
sort the vector
Now data:20 90 127 142 417 506 547 618 666 2333
size: 10
capacity: 16

Is the vector sorted? Yes
clear the vector
Now data:
size: 0
capacity: 0

insert 1, 2, 2, 3
Now data:1 2 2 3
size: 4
capacity: 4

remove 1
Now data:2 2 3
size: 3
capacity: 4

remove all 2
Now data:3
size: 1
capacity: 4
```

Figure 1: 1.png
2

基本思路

1. 内存管理

默认开的是 0 空间。在插入时检查，如果 $size = capacity$

直接 `double capacity`，并且“移动”(复制)原来的数据到新的空间。

2. 数据连续性

`popBack`, `pushBack` 不需要考虑数据连续性，而 `insert`, `erase` 需要考虑数据连续性问题。

`insert`, `erase` 时实际上开辟了新的空间，释放了旧的空间。

3. 数据访问和修改 实现了一个 `reference` 类，实际上是对指针的封装。包括但不限于下列好处：

1. 防止对指针的错误操作，导致指针被释放或其他不可预期的问题。
2. 能重构 `=`, `<<` 运算符，操作更加人性化（直接返回指针，需要 `*v[0]` 获取值）

关于进一步优化的思考

1. 移动数据造成的运算开销

是否存在更优化的解决方法，进行 $O(1)$ 复杂度的数据迁移？

是否能将 `vector` 和 `linked list` 结合起来（使用指针保证数据链接，不保证数据的连续性，代价是查询开销增大）

2. 内存开销的优化

按照本算法，内存开销的增长率是巨大的 ($O(2^n)$)，是否存在优化空间。

(STL 中的 `vector` 提供了 `shrink_to_fit` 的方法允许手动释放没用的内存，是否存在自动释放内存的方式?)