作业一: 双链表的设计和实现

张竣凯
3210300361
数学与应用数学

2022 年 10 月 7 日

双向链表也叫双链表，是链表的一种，它的每个数据结点中都有两个指针，分别指向直接后继和直接前驱。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点。一般我们都构造双向循环链表。

——摘自百度百科《双向链表》

1 设计思路

1.1 在 DoubleLinkedList.h 头文件里头编写双链表的类

即：class DoubleLinkedList

1.2 编写该类中所需的变量、结构、子类、以及基本功能等

例如：Node 结构、Iterator 类、begin 函数等

1.3 编写出结构所需的变量

例如：Node 结构中包含数据以及两个指针

1.4 编写出各个子类所需的函数

例如：Iterator 类中的各种运算符重载函数等

1.5 编写出双链表类所需的函数

例如：begin 函数、end 函数、size 函数等

1.6 在 main.cpp 主流程里头编写外部函数 find 以及测试程序

尽可能地使用引用 & 以减少内部复制，以及充分考虑必要的 const 限制以提高安全性

2 额外函数

2.1 printList 函数

```
1  void printList()
2  {
3      for (iterator itr = begin(); itr != end(); itr++)
4      {
5          std::cout << *itr << " ";
6      }
7      std::cout << '\b' << std::endl;
8  }
```

2.2 find 函数

```
1  template <typename DT>
2  typename DoubleLinkedList<DT>::iterator find
3          (DoubleLinkedList<DT>& _list, const DT& _val)
4  {
5      typename DoubleLinkedList<DT>::iterator itr = _list.begin();
6      typename DoubleLinkedList<DT>::iterator end = _list.end();
7      int position = 1;
8      for(itr; itr != end; ++itr)
9      {
10         if(*itr == _val)
11         {
12             std::cout << "The position of value \"" << _val << "\" is "
13                       << position << "." << std::endl;
14             return itr;
15         }
16         position++;
17     }
18     std::cout << "The given value \"" << _val
19               << "\" is not found in the list!" << std::endl;
20     return itr;
21 };
```

2.3 main 函数

```
int main(int argc, char* argv[])
{
    DoubleLinkedList<int> L;
    L.push_back(1);
    L.push_back(2);
    L.push_back(3);
    L.push_back(4);
    L.push_back(5);
    L.printList();
    DoubleLinkedList<int>::iterator iter;
    iter = find(L, 3);
    L.erase(iter);
    L.printList();
    iter = find(L, 3);
    return 0;
}
```

3 测试说明与结果

将 main.cpp 与 DoubleLinkedList.h 两个文件放入项目 list 后，进入终端，找到 list 的路径后，输入 g++ -o test main.cpp 后回车，接着输入./test 后回车，将会出现以下结果：

1 2 3 4 5
The position of value ”3” is 3.
1 2 4 5
The given value ”3” is not found in the list!

4 内存泄漏检查

承接上个部分，继续在终端中输入 valgrind ./test 后回车，将会出现以下结果

==2878==
==2878== HEAP SUMMARY:
==2878== in use at exit: 0 bytes in 0 blocks
==2878== total heap usage: 9 allocs, 9 frees, 73,896 bytes allocated
==2878==
==2878== All heap blocks were freed – no leaks are possible
==2878==
==2878== For lists of detected and suppressed errors, rerun with: -s
==2878== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)