**单循环列表**

1. **实验目的和要求**

用单循环链表实现一个队列，并只能用一个指针表示队列，则最好情况下入队和出队的时间性能分别是多少。

设计并实现上述队列类。

**二、实验内容**

1. 实验准备
   1. 理论知识介绍
      1. 时间复杂度计算: 通常采用下属办法来估计算法在给定输入下的计算量：
         1. 根据问题的特点合理地选择一种或几种操作作为“标准操作”，将标准操作作为一个抽象的运算单位。
         2. 确定每个算法在给定输入下共执行了多少次标准操作，并将它作为算法的计算量。
         3. 利用求和定理和求积定理相结合可以计算出最终的时间复杂度
      2. 队列
         1. 队列基本概念：队列可以看成是一种特殊的线性表。在这种线性表上，插入限定在表的一端，删除限定再表的另一端。允许进行插入的一端称为队尾，允许进行删除的一段称为队头。位于队头的元素称为队头元素，位于对位的元素称为对位元素。新插入的匀速是对位元素，即将被删除的是队头元素。队列中的元素的个数称为队列长度。若队列中没有元素，则成为空队列。队列也被称为先进先出的线性表。
         2. 队列的顺序实现
            1. 队列的顺序实现称为顺序队列。与顺序表相似，队列的顺序是先也是有一个一维数组组成，队列的元素依次存放在数组中。在栈的顺序实现中，可将栈底位置设在数组两段的任意一段，因为栈的操作仅在栈底进行，因而不会引起大量数据的移动。而在队列的顺序实现中，队头队尾设在哪里是很值得讨论的，这对队列的性能有很大的影响。
            2. 循环队列：普通的顺序队列的实现存在空间利用率的问题。通常的情况下，当rear指针指向数组的末尾的时候，此时这个队列已经被占满，不能再进行入队的操作，但是实际上数组的空间并没有被完全占满，在front指针前面的空间实际上还是空的，所以说一般的方法就不能有效地进行空间的利用，但是循环队列不同，循环队列的实现可以将整个数组的空间充分利用。解决这个问题很简单，可以把数组的头尾看成是相连的，当rear指针指向数组的末尾的时候我们就可以将下一个将要入队的元素放在下标为0的空间中，这种方案被称为循环队列。
         3. 队列的链接实现：

队列的链接实现称为链接队列。由于队列的操作是现在队列的梁端进行的，不会对队列中的其他元素进行操作，也不需要找某一元素直接前驱的操作，因此不需要双链表，用单链表就足够了。单链表在表头和表尾的插入都很简单，但是在表尾删除非常麻烦，所以一般将单链表的表头作为队头，单链表的表尾作为队尾。

1. 实验项目
   1. 分析
      1. 这次的实现中单循环链表的指针只有一个，所以这个指针的位置的选择需要有一定的考虑，不同的指针的位置的选择可能有不同的效果。
      2. 如果将表头的选取为指针的位置（假设链表中的元素为1,2,3…n）,那么新插入一个元素n+1需要将表尾元素的指针指向新的结点的位置，但是由于我们只有表头的指针，所以要改变表尾的指针的位置需要将整个单链表遍历一遍，时间复杂度为O(n)。要删除一个结点（出队），那么我们需要将表尾结点指向第二个元素所在的结点，同理时间复杂度也为O(n)。
      3. 如果我们将表尾选取为指针的位置，我们就能避免以上的麻烦，因为如果我们将表尾选取为指针的位置，我们通过一次运算就能得到表头的位置，同时我们还保留着表尾的位置，这样我们在入队和出队的运算中就能避免对整个链表进行遍历，从而将复杂度降为O(1)。
   2. 方案
      1. 利用递归函数转换进制。
      2. 递归函数需要有一个终止条件（当）来进行。
      3. 需要将输出的语句放在递归语句的后面来实现栈的原理。
   3. 测试数据

int main() {

linkQueue<int> linkqueue;

for (int i = 1; i < 10; i++) {

linkqueue.enQueue(i);

cout << linkqueue.getRear() << endl;

}

for (int i = 1; i < 9; i++) {

linkqueue.deQueue();

cout << linkqueue.getHead() << endl;

}

return 0;

}

* 1. 测试结果

1

2

3

4

5

6

7

8

9

2

3

4

5

6

7

8

9

* 1. 出错情况

原来的这个类的析构函数是模仿书上的单链表的析构函数来进行实现的，代码如下：

template <class elemType>

linkQueue < elemType > :: ~linkQueue() {

node \* tmp;

while (rear != NULL) {

tmp = rear;

rear = rear->next;

delete tmp;

}

}

但是如果是这样的代码的话程序在运行的时候会出现内存泄漏的错误，在运行的最后的部分rear指针指向的地址为0xdddddddd。

因此在delete tmp;这个语句的时候会出错，为了纠正这个错误，我将判断语句修改成如下的语句：

template <class elemType>

linkQueue < elemType > :: ~linkQueue() {

node \* tmp;

while (rear->next != rear) {

tmp = rear;

rear = rear->next;

delete tmp;

}

}

这样的话最后一步就不会执行，就不会出现内存泄漏的错误。

**三、实验小结**

1. 重点

队列的单循环链表的实现。

1. 难点

单循环链表中指针位置的选取。单循环链表的具体类的实现。

1. 解决方法

单循环链表的指针的位置应该选在单循环链表中的表尾的元素，这样和将指针的位置指向表头能大大减少时间复杂度。

1. 收获、体会

队列的实现有很多种方法，通常的方法我们都需要两个空间来进行整个队列的数据的储存（头指针和尾指针，头指针和数组长度），在这次的单循环链表的实验中，我们节省了一个空间，减少了空间的复杂度，通常情况下就会增加时间的复杂度。我们确实增加了时间的复杂度，但是复杂度和原来相比还是在一个量级的（O(1)），实现了较好的效果。

指针位置的选择在这里有很大的作用，如果将指针选在头部的位置的话时间段复杂度就会增加到O(n)。

同时在处理动态的结构的时候很容易发生内存泄漏的情况，要时刻注意将不用的指针及时删除。