HeZhengfa

数据结构代码实现之队列的链表实现（C/C++）

上班闲着无聊，一直想着要开始写博客，但又不知道写什么。最近又回顾了下数据结构的知识，那就从数据结构开始吧。

前言

关于C语言结构体的知识以及队列的特性请读者自行了解，此处不做过多解释，嘻嘻。

同时此篇文章仅仅是关于队列的链表实现。

第一步：结构体编写

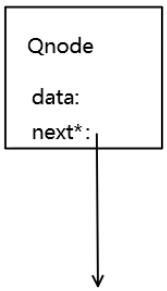
我们首先分析一下队列的特征：先进先出，队尾插入，队头删除，暂时想到的就这么多。

首先，对于链表的节点结构体的内容，我们首先想到的是它有一个值，还有一个指向下

一个节点的指针（链表相关知识请读者自行了解），那么它的结构体可实现如下：

```
typedef struct Qnode{
    int data;
    struct Qnode *next;
};
```

映射到图形，其是这样的结构：



接下来，要让这种节点实现队列的特性，我们可以再建立一个结构体，该结构体有一个指向队头节点的指针和一个指向队尾节点的指针，那么它的实现如下：

```
typedef struct LQueue{
    Qnode *front;
    Qnode *rear;
};
```

其中，front指针指向队头，rear指针指向队尾（注意，该指针是指向Qnode类型的指针）

映射到图形，其是这样的结构：

1. 对汉诺塔递归算法的理解（图解，附完整代码实现）(5528)
2. Ubuntu中Python3虚拟环境的搭建(5256)
3. Python关于函数作为返回值的理解（3分钟就看完了）(4387)
4. 数据结构代码实现之队列的链表实现（C/C++）(2954)
5. 关于Python中包裹传参和解包裹的理解(1191)

评论排行榜

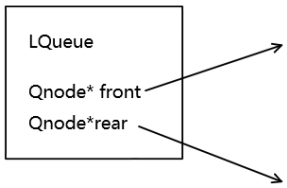
1. Python爬虫例子（笔记，不适合参考，愿意看的可以看看）(2)
2. 数据结构代码实现之队列的链表实现（C/C++）(1)

推荐排行榜

1. Ubuntu中Python3虚拟环境的搭建(1)

最新评论

1. Re:Python爬虫例子（笔记，不适合参考，愿意看的可以看看）
@起帆 我也有点忘记了，你先百度一下quot这个吧。太久远了😂...
--HeZhengfa
2. Re:Python爬虫例子（笔记，不适合参考，愿意看的可以看看）
result['reply_time']=re.findall(.....)中的正则表达式，看不懂，能解释一下吗？谢谢
--起帆
3. Re:数据结构代码实现之队列的链表实现（C/C++）
这个是循环队列还是链队列
--爱热闹的小炸呼



好了，结构体编写工作到这里就完成了，下面开始下一步工作。

第二步，队列的方法分析及实现

一个队列有哪些方法呢，根据前面提到的特性，首先要有插入和删除的方法，我们可以定义插入操作为入队（书上也是这么说的），删除操作为出队，这两个操作应该是队列里最基本的。接下来，初始化队列的方法也是尤其必要的。然后，为了测试方便，还可以定义一个获取队列的长度，队列是否为空，获取队头元素值，获取队尾元素值以及打印队列所有节点数据的方法。下面是对这些方法的实现。

初始化方法：void initQueue(LQueue *q);

方法描述：将创建的队列结构（通过参数传入该方法）的队头和队尾指针都指向一个动态生成的Qnode节点，代码如下：

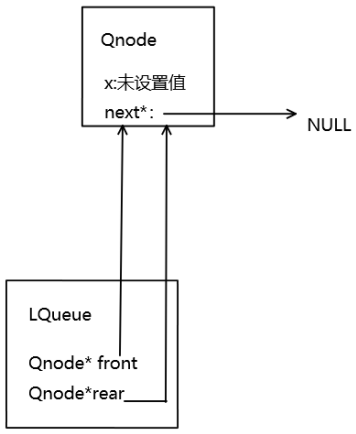
```
1 void initQueue(LQueue *q){
2     q->front = q->rear = (Qnode *)malloc(sizeof(Qnode));
3     q->front->next = NULL;
4 }
```

当创建了一个队列变量，然后调用该方法时：

代码：

```
1     LQueue L;
2     initQueue(&L);
```

内存空间如图：



判断队列是否为空方法：int empty(LQueue *t);

该方法很简单，不做过多描述，代码如下：

```
int empty(LQueue *t){
    return t->front->next == t->rear;
}
```

入队方法：void push(LQueue *t, int x);

方法描述：通过动态生成一个Qnode节点，然后将x赋值给该节点的data值，再将该节点插入到队列中，代码如下：

```
1 void push(LQueue *t, int x){
2     Qnode *s = (Qnode *)malloc(sizeof(Qnode));
3     s->data = x;
4     s->next = NULL;
5     t->rear->next = s;
6     t->rear = s;
7 }
```

代码解释：

第2行：动态生成一个Qnode节点，让指针s指向它；

第3行：将传入的x值赋值给生成的节点（s所指向）的data值；

第4行：将s所指节点的next指针置为空；

第5行：将队列的队尾指针的next指针指向s所指节点；

第6行：再将队尾指针指向s节点，完成push操作。

不懂的读者希望能自行画图帮助理解，其实图一画出来就一目了然了。

出队方法：void pop(LQueue *t);

方法描述：使用该方法时，首先应判断队列是否为空，为空则退出，不进行出队操作。如果队列不空，则首先定义一个Qnode类型指针q，让q指向队头节点的下一个节点（因为队头节点仅作为队头，不存储值），把队头去掉的话，就是头节点啦。然后让队头节点的next指针指向q所指节点的下一个节点，再释放掉q所指节点（q所指节点即为要出队的节点），代码如下：

```
1 void pop(LQueue *t){
2     if(empty(t)){
3         cout << "LQueue is empty,can't pop.\n";
4         return;
5     }
6     Qnode *q = t->front->next;
7     t->front->next = q->next;
8     free(q);
9 }
```

代码解释：

第2-5行：判断队列是否为空，若为空则打印提示消息后退出，不进行出队操作；

第6行：定义一个指针q，使其指向队头节点的next指针所指向的节点；（前面已经解释了，其实就是指向队头节点）

第7行：让队头节点的next指针指向q的next指针所指向的节点；

第8行：释放掉q所指的节点的内存，完成出队操作；

还是那句话，画图！一步一步理解。

获取队头节点的值方法：int getFront(LQueue *t);

该方法很简单，不做过多描述，代码如下：

```
1 int getFront(LQueue *t){
2     return t->front->next->data;
3 }
```

获取队尾节点的值方法：int getRear(LQueue *t);

该方法很简单，不做过多描述，代码如下：

```
1 int getRear(LQueue *t){
2     return t->rear->data;
```

```
3 }
```

获取队列长度的方法：int getSize(LQueue *t);

方法描述：使用一个指向头结点的指针，不断遍历，每遍历一次，计数器加1，当该指针指向空时，遍历完成，返回该计数器，代码如下：

```
1 int getSize(LQueue *t){
2     Qnode *q = t->front->next;
3     int k = 0;
4     while(q){
5         k++;
6         q = q->next;
7     }
8     return k;
9 }
```

代码解释：

第2行：定义一个指向队头节点的指针q;

第3行：定义一个计数器k;

第4-7行：该代码为，当q不指向NULL时，k+1,然后q指向下一个节点，继续循环判断。

第8行：当循环结束时，返回该计数器k，即为队列的长度。

打印队列所有值方法：void printQueue(LQueue *t);

方法描述，定义一个指向Qnode类型的指针，进行遍历，每遍历一个节点，打印该节点，然后继续遍历下一节点，代码如下：

```
1 void printQueue(LQueue *t){
2     Qnode *q = t->front->next;
3     while(q){
4         cout << q->data << " ";
5         q = q->next;
6     }
7     cout << "\n";
8 }
```

该代码比较简单，不做过多解释。

好了，方法至此已全部完成，接下来，就可以通过main函数进行测试了。

第三步：编写main方法测试运行

完整代码如下，亲测可用，希望各位新入坑的朋友多多敲代码练习哦：

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 typedef struct Qnode{
 5     int data;
 6     struct Qnode *next;
 7 };
 8
 9 typedef struct LQueue{
10     Qnode *front;
11     Qnode *rear;
12 };
13
14 void initQueue(LQueue *q){
15     q->front = q->rear = (Qnode *)malloc(sizeof(Qnode));
16     q->front->next = NULL;
```

```
}

int empty(LQueue *t){
    return t->front->next == t->rear;
}

void push(LQueue *t, int x){
    Qnode *s = (Qnode *)malloc(sizeof(Qnode));
    s->data = x;
    s->next = NULL;
    t->rear->next = s;
    t->rear = s;
}

void pop(LQueue *t){
    if(empty(t)){
        cout << "LQueue is empty,can't pop.\n";
        return;
    }
    Qnode *q = t->front->next;
    t->front->next = q->next;
    free(q);
    if(t->rear == NULL)
        t->rear = t->front;
}

int getFront(LQueue *t){
    return t->front->next->data;
}

int getRear(LQueue *t){
    return t->rear->data;
}

int getSize(LQueue *t){
    Qnode *q = t->front->next;
    int k = 0;
    while(q){
        k++;
        q = q->next;
    }
    return k;
}

void printQueue(LQueue *t){
    Qnode *q = t->front->next;
    while(q){
        cout << q->data << " ";
        q = q->next;
    }
    cout << "\n";
}
int main(){
    LQueue L;
    initQueue(&L);
    cout << "Push data to Queue...\n";
    push(&L,2);
    push(&L,5);
    push(&L,4);
    push(&L,3);
    push(&L,6);
    push(&L,8);
    push(&L,10);
    push(&L,11);
    cout << "Push finished.\n";
    cout << "You have pushed such data:";
    printQueue(&L);
    cout << "Pop data out of Queue...\n";
    pop(&L);
    cout << "Pop finished.\n";
    cout << "Now the Queue have such data:";
    printQueue(&L);
```

```
    cout << "Get Queue's front data:" << getFront(&L) << endl;
    cout << "Get Queue's rear data:" << getRear(&L) << endl;
    cout << "Get Queue's size:" << getSize(&L) << endl;
    pop(&L);
    pop(&L);
    pop(&L);
    cout << "After poped 3 times:";
    printQueue(&L);
    cout << "Judge the Queue is null or not(0 means not null,others means null):" << empty
    pop(&L);
    pop(&L);
    pop(&L);
    pop(&L);
    cout << "After poped 4 times:";
    printQueue(&L);
    cout << "Judge the Queue is null or not(0 means not null,others means null):" << empty

    return 0;
}
```

人生中的第一篇博客，写的不好还请海涵~~祝大家生活愉快~~

分类: 基础数据结构

标签: 数据结构, 队列, 链式结构

HeZhengfa
关注 - 1
粉丝 - 2
+加关注

» 下一篇： 对汉诺塔递归算法的理解（图解，附完整代码实现）

posted @ 2018-12-28 13:24 HeZhengfa 阅读(2954) 评论(1) 编辑 收藏 举报

• Three.js 实现脸书元宇宙 3D 动态 Logo

• 关于研发规范化的一些实践和思考

• 2次心态变化和27个问题：机制落地的部分全貌与节奏控制

最新新闻：

• 微软预告：2021年度Windows丑陋毛衣将于11月30日发布（2021-11-23 09:17）

• Twitter为其Birdwatch计划贡献者引入别名机制（2021-11-23 09:11）

• 微软暂时不会为 ARM Mac 开发 Windows 11（2021-11-23 09:06）

• “怀柔一号”卫星引导国际望远镜进行联合观测（2021-11-23 09:00）

• 微软和高通的排他性协议即将到期：联发科/三星等企业要推ARM PC芯片（2021-11-23 08:53）

» 更多新闻...