Listing 1: `li_queue.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef int item_t;
5  typedef struct qu_n_t {item_t    item;
6                         struct qu_n_t *next; } qu_node_t;
7
8  typedef struct {qu_node_t *remove;
9                  qu_node_t *insert; } queue_t;
10 typedef qu_node_t node_t;
11
12 #define BLOCKSIZE 256
13
14 node_t *currentblock = NULL;
15 int     size_left;
16 node_t *free_list = NULL;
17
18 node_t *get_node()
19 { node_t *tmp;
20   if( free_list != NULL )
21   {  tmp = free_list;
22      free_list = free_list -> next;
23   }
24   else
25   {  if( currentblock == NULL || size_left == 0)
26      {  currentblock =
27                 (node_t *) malloc( BLOCKSIZE * sizeof(node_t) );
28         size_left = BLOCKSIZE;
29      }
30      tmp = currentblock++;
31      size_left -= 1;
32   }
33   return( tmp );
34 }
35
36
37 void return_node(node_t *node)
38 {  node->next = free_list;
39    free_list = node;
40 }
41
42 queue_t *create_queue()
43 {   queue_t *qu;
44     qu = (queue_t *) malloc( sizeof(queue_t) );
```

```
45      qu->remove = qu->insert = NULL;
46      return( qu );
47  }
48
49  int queue_empty(queue_t *qu)
50  {   return( qu->insert == NULL );
51  }
52
53  void enqueue( item_t x, queue_t *qu)
54  {   qu_node_t *tmp;
55      tmp = get_node();
56      tmp->item = x;
57      tmp->next = NULL; /* end marker */
58      if ( qu->insert != NULL ) /* queue nonempty */
59      {   qu->insert->next = tmp;
60          qu->insert = tmp;
61      }
62      else /* insert in empty queue */
63      {   qu->remove = qu->insert = tmp;
64      }
65  }
66
67  item_t dequeue(queue_t *qu)
68  {   qu_node_t *tmp; item_t tmp_item;
69      tmp = qu->remove; tmp_item = tmp->item;
70      qu->remove = tmp->next;
71      if( qu->remove == NULL ) /* reached end */
72          qu->insert = NULL; /* make queue empty */
73      return_node(tmp);
74      return( tmp_item );
75  }
76
77  item_t front_element(queue_t *qu)
78  {   return( qu->remove->item );
79  }
80
81  void remove_queue(queue_t *qu)
82  {   qu_node_t *tmp;
83      while( qu->remove != NULL)
84      { tmp = qu->remove;
85        qu->remove = tmp->next;
86        return_node(tmp);
87      }
88      free( qu );
89  }
```

```
90
91  int main()
92  {   queue_t *qu;
93      char nextop;
94      qu = create_queue();
95      printf("Made List-Based Queue\n");
96      while( (nextop = getchar())!= 'q' )
97      { if( nextop == 'e' )
98        { int insitem;
99          scanf(" %d", &insitem);
100         enqueue( insitem, qu );
101         printf(" enqueued %d. The current front item is %d\n", insitem,
102             front_element(qu) );
103       }
104       if( nextop == 'd' )
105       { int de_item;
106         getchar();
107         de_item = dequeue(qu);
108         printf("  dequeued item %d", de_item);
109         if( queue_empty(qu) )
110           printf(" the queue is now empty\n");
111         else
112           printf(" the front element is now %d\n",  front_element(qu) );
113
114       }
115       if( nextop == '?' )
116       { getchar();
117         if( queue_empty(qu) )
118           printf("the queue is empty\n");
119         else
120           printf("the front element is %d\n", front_element(qu) );
121       }
122
123     }
124     remove_queue(qu);
125     printf(" removed queue\n");
126     return(0);
127 }
```