```
#include <iostream>
using  namespace  std  ;

struct  elemenet
{
    int                data  ;
    struct elemenet *  next  ;
};

typedef  struct elemenet Element  ;
typedef  Element *       Stack    ;
typedef  Element *       Queue    ;

Stack  create_stack()
{
    return  NULL  ;
}

bool  empty_stack(Stack stk)
{
    if(stk == NULL)
        return  true  ;
    else
        return  false ;
}

int  depth_stack(Stack stk)
{
    Stack  ptr    =  stk  ;
    int    count  =  0    ;

    while(ptr != NULL)
    {
        count++  ;
        ptr  =  ptr->next  ;
    }

    return  count  ;
}

Stack  push(Stack stk, int val)
{
    Stack  ptr  =  new  Element  ;
    ptr->data   =  val  ;

    ptr->next  =  stk  ;
    stk        =  ptr  ;

    return  stk  ;
}

Stack  pop(Stack stk, int &val)
{
    if(!empty_stack(stk))
    {
        Stack  ptr  =  stk  ;

        val  =  stk->data  ;
        stk  =  stk->next  ;
        delete  ptr        ;

        return  stk  ;
    }
    else
    {
        cout  <<  "Error, the stack is already empty."  <<  endl  ;
        return  NULL  ;
    }
}

int  top_stack(Stack stk)
{
    if(empty_stack(stk))
    {
        cout  <<  "Error, Stack is empty."  <<  endl  ;
        return  0  ;
    }
```

```
78
79      return  stk->data  ;
80  }
81
82  Stack  replace_top_stack(Stack stk, int val)
83  {
84      if(empty_stack(stk))
85      {
86          cout  <<  "Error, Stack is empty."  <<  endl  ;
87          return  NULL  ;
88      }
89
90      stk->data  =  val  ;
91
92      return  stk  ;
93  }
94
95  void  display_stack(Stack stk)
96  {
97      Stack  ptr    =  stk  ;
98
99      if(empty_stack(stk))
100     {
101         cout  <<  "Stack is empty."  <<  endl  ;
102         return  ;
103     }
104
105     while(ptr != NULL)
106     {
107         cout  <<  ptr->data  <<  " "  ;
108         ptr  =  ptr->next  ;
109     }
110     cout  <<  endl  ;
111
112     return  ;
113 }
114
115 Queue  create_queue()
116 {
117     return  NULL  ;
118 }
119
120 bool  empty_queue(Queue que)
121 {
122     if(que == NULL)
123         return  true  ;
124     else
125         return  false ;
126 }
127
128 int  length_queue(Queue que)
129 {
130     Queue  ptr    =  que  ;
131     int    count  =  0    ;
132
133     while(ptr != NULL)
134     {
135         count++  ;
136         ptr  =  ptr->next  ;
137     }
138
139     return  count  ;
140 }
141
142 Queue  enqueue(Queue que, int val)
143 {
144     Queue  elm  =  new  Element  ;
145     elm->data   =  val   ;
146     elm->next   =  NULL  ;
147
148     if(empty_queue(que))
149     {
150         que  =  elm  ;
151         return  que  ;
152     }
153
154     Queue  ptr  =  que  ;
```

```
155
156     while(ptr->next != NULL)
157         ptr  =  ptr->next  ;
158
159     ptr->next  =  elm  ;
160
161     return  que  ;
162 }
163
164 Queue  dequeue(Queue que, int &val)
165 {
166     if(!empty_queue(que))
167     {
168         Queue  ptr  =  que  ;
169
170         val  =  que->data  ;
171         que  =  que->next  ;
172         delete  ptr        ;
173
174         return  que  ;
175     }
176     else
177     {
178         cout  <<  "Error, the queue is already empty."  <<  endl  ;
179         return  NULL  ;
180     }
181 }
182
183 int  tail_queue(Queue que)
184 {
185     if(empty_queue(que))
186     {
187         cout  <<  "Error, Queue is empty."  <<  endl  ;
188         return  0  ;
189     }
190
191     return  que->data  ;
192 }
193
194 int  head_queue(Queue que)
195 {
196     if(empty_queue(que))
197     {
198         cout  <<  "Error, Queue is empty."  <<  endl  ;
199         return  0  ;
200     }
201
202     Queue  ptr  =  que  ;
203
204     while(ptr->next != NULL)
205         ptr  =  ptr->next  ;
206
207     return  ptr->data  ;
208 }
209
210 void  display_queue(Queue que)
211 {
212     Queue  ptr    =  que  ;
213
214     if(empty_queue(que))
215     {
216         cout  <<  "Queue is empty."  <<  endl  ;
217         return  ;
218     }
219
220     while(ptr != NULL)
221     {
222         cout  <<  ptr->data  <<  " "  ;
223         ptr  =  ptr->next  ;
224     }
225     cout  <<  endl  ;
226
227     return  ;
228 }
229
```

```
230 Queue  reverse_queue(Queue &que)
231 {
232     Queue  rst  =  create_queue()  ;
233     Stack  stk  =  create_stack()  ;
234 
235             int  val  ;
236 
237     for(int i = 0 ; i < length_queue(que) ; i++)
238     {
239         que  =  dequeue(que, val)  ;
240         que  =  enqueue(que, val)  ;
241         stk  =  push(stk, val)     ;
242     }
243 
244     while(!empty_stack(stk))
245     {
246         stk  =  pop(stk, val)      ;
247         rst  =  enqueue(rst, val)  ;
248     }
249 
250     return  rst  ;
251 }
252 
253 Stack  reverse_stack(Stack &stk)
254 {
255     Stack  rst  =  create_stack()  ;
256     Stack  tmp  =  create_stack()  ;
257 
258     int  val  ;
259 
260     while(!empty_stack(stk))
261     {
262         stk  =  pop(stk, val)   ;
263 
264         rst  =  push(rst, val)  ;
265         tmp  =  push(tmp, val)  ;
266     }
267 
268     while(!empty_stack(tmp))
269     {
270         tmp  =  pop(tmp, val)   ;
271 
272         stk  =  push(stk, val)  ;
273     }
274 
275     return  rst  ;
276 }
277 
278 int  main()
279 {
280     Queue  que  =  create_queue()  ;
281 
282     que  =  enqueue(que, 1)  ;
283     que  =  enqueue(que, 2)  ;
284     que  =  enqueue(que, 3)  ;
285 
286     display_queue(que)  ;
287 
288     Queue  q2  =  reverse_queue(que)  ;
289     display_queue(q2)  ;
290 
291     Queue  stk  =  create_queue()  ;
292 
293     stk  =  enqueue(stk, 1)  ;
294     stk  =  enqueue(stk, 2)  ;
295     stk  =  enqueue(stk, 3)  ;
296 
297     display_queue(stk)  ;
298 
299     Stack  s2  =  reverse_stack(stk)  ;
300 
301     display_stack(s2)  ;
302 
303     return  0  ;
304 }
```