```cpp
#include <stdio.h>
#include <stdlib.h>

////////////////////////////////////
#define LIST_INIT_SIZE 100
#define LIST_INCREMENT 10
typedef int ElemType;
typedef struct
{
    ElemType *base;
    int length;
    int listsize;
} SqList;



#define STACK_INIT_SIZE      100
#define STACK_INCREMENT_SIZE 10
typedef char SElemType;
typedef struct
{
    SElemType *base;
    SElemType *top;
    int stacksize;
} SqStack;
////////////////////////////////////
#define MAX_QUEUE_SIZE       100
typedef int QElemType;
typedef struct
{
    QElemType   *base;
    int         front;
    int         rear;
} SqQueue;
////////////////////////////////////

enum Status
{
    OVERFLOW = -1,
    ERROR    =  0,
    OK       =  1
};


////////////////////////////////////
//函数声明
Status InitStack(SqStack& S);
Status DestroyStack(SqStack& S);
Status ClearStack(SqStack& S);
```

```cpp
bool StackEmpty(SqStack& S);
Status Push(SqStack& S, SElemType e);
Status Pop(SqStack& S, SElemType& e);
void PirntStack(SqStack& S);

Status InitQueue(SqQueue& Q);
Status DestroyQueue(SqQueue& Q);
Status ClearQueue(SqQueue& Q);
int QueueLenth_M2(SqQueue& Q);
Status EnQueue(SqQueue& Q, QElemType e);
Status DeQueue(SqQueue& Q, QElemType& e);
void PrintQueue(SqQueue& Q);

bool match(char *exp);
void PrintStar(void);

////////////////////////////////////
//主函数
int main()
{
    SqList  l;

    SqQueue Q;
    InitQueue(Q);

    char   chos;
    char   ch[40];


    while(1)
    {
        printf("1.括号匹配检验\n2.整数入队列\n3.退出\n");
        printf("请选择您要运行的程序: ");
        scanf_s("%c", &chos);
        while (chos == '\n')
        {
            scanf_s("%c", &chos);
        }

        switch(chos)
        {
            case '1':
                {
                    printf("请输入需要检验的字符串: ");
                    scanf_s("%s", ch);

                    if (match(ch))
                    {
                        printf("该字符串括号匹配\n");
```

```
 99                      }
100                      else
101                      {
102                          printf("该字符串括号不匹配\n");
103                      }
104                  }break;
105              case '2':
106                  {
107                      int    temp;
108                      int    get;
109
110                      printf("请输入整数的序列(以0结束): ");
111                      scanf_s("%d", &temp);
112                      printf("出队的整数组成的序列是:\n");
113                      PrintStar();
114                      while (temp)
115                      {
116                          if (temp > 0)
117                          {
118                              EnQueue(Q, temp);
119                          }
120                          else
121                          {
122                              if (DeQueue(Q, get))
123                              {
124                                  printf("%d ", get);
125                              }
126                          }
127
128                          scanf_s("%d", &temp);
129                      }
130                      putchar(10);
131                      PrintStar();
132
133                      printf("现在队列中的整数序列为:\n");
134                      PrintQueue(Q);
135                      ClearQueue(Q);
136                  }break;
137              case '3':
138                  {
139                      DestroyQueue(Q);
140                      exit (0);
141                  }break;
142              default:
143                  {
144                      printf("选择错误！\n");
145                  }
146          }
147
```

```
148             putchar(10);
149         }
150     }
151
152     ////////////////////////////////////
153     //函数定义
154
155
156     //初始化栈
157     Status InitStack(SqStack& S)
158     {
159         S.base = (SElemType *) malloc(STACK_INIT_SIZE * sizeof(SElemType));
160         if (!S.base)
161         {
162             exit(OVERFLOW);
163         }
164
165         S.top = S.base;
166         S.stacksize = STACK_INIT_SIZE;
167
168         return OK;
169     }
170
171     //销毁栈
172     Status DestroyStack(SqStack& S)
173     {
174         if (!S.base)
175         {
176             free(S.base);
177             S.stacksize = 0;
178         }
179
180         return OK;
181     }
182
183     //将栈置为空栈
184     Status ClearStack(SqStack& S)
185     {
186         if (!S.base)
187         {
188             return ERROR;
189         }
190
191         S.top = S.base;
192
193         return OK;
194     }
195
196     //栈判空
```

```cpp
bool StackEmpty(SqStack& S)
{
    if (!S.base)
    {
        return false;
    }

    if (S.base == S.top)
    {
        return true;
    }
    else
    {
        return false;
    }
}

//插入元素e为新的栈顶元素
Status Push(SqStack& S, SElemType e)
{
    if (S.top - S.base >= S.stacksize)
    {
        S.base = (SElemType *) realloc (S.base, (S.stacksize +
          STACK_INCREMENT_SIZE) * sizeof(SElemType));
        if (!S.base)
        {
            exit(OVERFLOW);
        }

        S.top = S.base + S.stacksize;
        S.stacksize += STACK_INCREMENT_SIZE;
    }

    *(S.top++) = e;

    return OK;
}

//删除栈顶元素并用e返回其值
Status Pop(SqStack& S, SElemType& e)
{
    if (S.base == S.top)
    {
        return ERROR;
    }

    e = *--S.top;

    return OK;
```

```cpp
}

void PrintStack(SqStack& S)
{
    SElemType *p = S.base;

    PrintStar();
    while (p < S.top)
    {
        printf("%c ", *p);
        ++p;
    }
    putchar(10);
    PrintStar();
}

//括号匹配检验
bool match(char *exp)
{
    bool match = true;

    SqStack S;
    InitStack(S);

    char ch;    //ch存放出栈元素
    int i = 0;  //i是当前字符下标

    while ((exp[i]!='\0') && (match==true))
    {
        if (exp[i] == '(')
        {
            Push(S, exp[i]);
        }
        else if (exp[i] == ')')
        {
            if (!StackEmpty(S))
            {
                Pop(S,ch);
                if (ch != '(')  //情况1
                {
                    match = false;
                }
            }
            else    //情况2
            {
                match = false;
            }
        }

```

```
294             ++i;
295         }
296
297         if(!StackEmpty(S))  //情况3
298         {
299             match = false;
300         }
301         printf("现在栈中存储的所有元素是:\n");
302         PrintStack(S);
303
304         DestroyStack(S);
305
306         return match;
307     }
308
309
310
311
312
313     //初始化顺序队列
314     Status InitQueue(SqQueue& Q)
315     {
316         Q.base = (QElemType *) malloc(MAX_QUEUE_SIZE * sizeof(QElemType));
317         if (!Q.base)
318         {
319             exit(OVERFLOW);
320         }
321         Q.front = Q.rear = 0;
322
323         return OK;
324     }
325
326     //销毁循环队列Q
327     Status DestroyQueue(SqQueue& Q)
328     {
329         if (Q.base == NULL)
330         {
331             return ERROR;
332         }
333
334         free(Q.base);
335         return OK;
336     }
337
338     //将循环队列Q清为空队列
339     Status ClearQueue(SqQueue& Q)
340     {
341         if (Q.base == NULL)
342         {
```

```
343             return ERROR;
344         }
345         else
346         {
347             Q.front = Q.rear;
348         }
349 
350         return OK;
351     }
352 
353     //求循环队列Q长度
354     int QueueLenth_M2(SqQueue& Q)
355     {
356         return (Q.rear - Q.front + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
357     }
358 
359     //插入元素e为Q的新的队尾元素
360     Status EnQueue(SqQueue& Q, QElemType e)
361     {
362         if ((Q.rear + 1) % MAX_QUEUE_SIZE == Q.front)
363         {
364             printf("队列已满，无法添加新的元素!\n");
365             return ERROR;
366         }
367 
368         Q.base[Q.rear] = e;
369         Q.rear = (Q.rear + 1) % MAX_QUEUE_SIZE;
370 
371         return OK;
372     }
373 
374     //若循环队列Q不为空，则删除Q的队头元素并用e返回其值
375     Status DeQueue(SqQueue& Q, QElemType& e)
376     {
377         if(Q.front == Q.rear)
378         {
379             return ERROR;
380         }
381 
382         e = Q.base[Q.front];
383         Q.front = (Q.front + 1) % MAX_QUEUE_SIZE;
384 
385         return OK;
386     }
387 
388     void PrintQueue(SqQueue& Q)
389     {
390         int i = Q.front;
391 
```

```
392     PrintStar();
393     while (i < Q.rear)
394     {
395         printf("%d ", Q.base[i++]);
396         if (i > MAX_QUEUE_SIZE)
397         {
398             i = 0;
399         }
400     }
401     putchar(10);
402     PrintStar();
403 }
404
405 void PrintStar(void)
406 {
407     printf("**************************\n");
408 }
```