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

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

////////////////////////////////////////////////
//树的二叉链表存储表示
typedef char TelemType;
typedef struct BiTNode
{
    TelemType data;
    BiTNode *lchild, *rchild;
    int tag;                //访问标志域
} BiTNode, *BiTree;

////////////////////////////////////////////////
//栈的顺序存储表示
#define STACK_INIT_SIZE      100
#define STACK_INCREMENT_SIZE 10
typedef BiTree SElemType;
typedef struct
{
    SElemType *base;
    SElemType *top;
    int stacksize;
} SqStack;

////////////////////////////////////////////////
//函数声明
Status CreateBiTree(BiTree& T);
Status DestroyBiTree(BiTree& T);
void   PreOrderTraverse(BiTree root, void (*visit) (BiTree));
void   PostOrderTraverse(BiTree root, void (*visit) (BiTree));
void   PrintNodeData(BiTree root);
void   InitNodeTag(BiTree root);
void   DestroyNode(BiTree root);

Status      InitStack(SqStack& S);
Status      DestroyStack(SqStack& S);
Status      ClearStack(SqStack& S);
bool        StackEmpty(SqStack& S);
Status      Push(SqStack& S, SElemType e);
SElemType   GetTop(SqStack& S);
SElemType   Pop(SqStack& S);
```

```
50  void        PrintStack(SqStack& S);
51
52  void   PrintStar(void);
53  ////////////////////////////////////////////////
54  int main()
55  {
56      BiTree T = NULL;
57      char chos;
58
59      while(1)
60      {
61          printf("1.Creat a binary tree:\n2.Traverse binary tree(postorder)\n3.Quit
              \n");
62
63          printf("So choose what you want to do now: ");
64          scanf_s("%c", &chos);
65          while(chos == 10)
66          {
67              scanf_s("%c", &chos);
68          }
69
70          putchar(10);
71
72          switch(chos)
73          {
74              case '1':
75                  {
76                      if (T != NULL)
77                      {
78                          printf("You have creat one already.\nDo you want to creat
                          another one?\n");
79                          printf("1.Yes      2.No\n");
80                          printf("So give me your choice: ");
81                          scanf_s("%c", &chos);
82                          while(chos == 10)
83                          {
84                              scanf_s("%c", &chos);
85                          }
86
87                          switch(chos)
88                          {
89                              case '1': DestroyBiTree(T), putchar(10); break;
90                              case '2': goto NG; break;
91                              default : printf("CHOOSE ERROR!\n");
92                          }
93                      }
94
95                      printf("You will create a binary tree.\n");
96                      printf("Please input the extended preorder sequence of this
```

```
                    tree: ");
97                  CreateBiTree(T);
98                  printf("Congratulations! The binary tree has been created!
                    \n");
99
100                 NG:;
101             }break;
102
103         case '2':
104             {
105                 if (T == NULL)
106                 {
107                     printf("You have to creat a tree first.\n"); break;
108                 }
109
110                 printf("Its postorder sequence is :");
111                 PostOrderTraverse(T, PrintNodeData);
112             }break;
113         case '3':
114             {
115                 if (T != NULL)
116                 {
117                     DestroyBiTree(T);
118                     T = NULL;
119                 }
120
121                 exit (0);
122             }break;
123         case '4':
124             {
125                 DestroyBiTree(T);
126                 T = NULL;
127             }break;
128         default : printf("CHOOSE ERROR!\n");
129     }
130
131     putchar(10);
132 }
133
134 system("pause");
135 return 0;
136 }
137
138
139 ////////////////////////////////////////////////
140 //函数定义(树)
141 Status CreateBiTree(BiTree& T)
142 {
143     char ch;
```

```
144     ch = getchar();
145     while(ch == 10)
146     {
147         ch = getchar();
148     }
149 
150     if( ch == '.')
151     {
152         T = NULL;
153     }
154     else
155     {
156         if (!(T = (BiTree) malloc(sizeof(BiTNode))))
157         {
158             return ERROR;
159         }
160 
161         T->data = ch;
162         T->tag  = 0;
163         CreateBiTree(T->lchild);
164         CreateBiTree(T->rchild);
165     }
166     return OK;
167 }
168 
169 Status DestroyBiTree(BiTree& T)
170 {
171     if (T == NULL)
172     {
173         return ERROR;
174     }
175 
176     DestroyBiTree(T->lchild);
177     DestroyBiTree(T->rchild);
178 
179     DestroyNode(T);
180     return OK;
181 }
182 
183 void PreOrderTraverse(BiTree root, void (*visit) (BiTree))
184 {
185     if (root != NULL)
186     {
187         visit(root);
188 
189         PreOrderTraverse(root->lchild, visit);
190         PreOrderTraverse(root->rchild, visit);
191     }
192 }
```

```
193
194 void PostOrderTraverse(BiTree root, void (*visit) (BiTree))
195 {
196       SqStack S;    //工作栈
197       InitStack(S);
198       BiTree p=root;
199       while(p || !StackEmpty(S))
200       {
201         if (p)
202         {
203             Push(S, p);
204             p=p->lchild;
205         }
206         else
207         {
208             p = GetTop(S);
209             if (p->tag == 0)
210             {
211                 p->tag = 1;
212                 p = p->rchild;
213             }
214             else
215             {
216                 p = Pop(S);
217                 visit(p);   //访问根结点
218                 p = NULL;
219             }
220         }
221       }
222
223       putchar(10);
224       PreOrderTraverse(root, InitNodeTag);
225  }
226
227 void PrintNodeData(BiTree root)
228 {
229     printf("%c ", root->data);
230 }
231
232 void InitNodeTag(BiTree root)
233 {
234     root->tag = 0;
235 }
236
237 void DestroyNode(BiTree root)
238 {
239     free(root);
240 }
241
```

```
242
243
244 ////////////////////////////////////////////////
245 //函数定义(栈)
246 //初始化栈
247 Status InitStack(SqStack& S)
248 {
249     S.base = (SElemType *) malloc(STACK_INIT_SIZE * sizeof(SElemType));
250     if (!S.base)
251     {
252         exit(OVERFLOW);
253     }
254
255     S.top = S.base;
256     S.stacksize = STACK_INIT_SIZE;
257
258     return OK;
259 }
260
261 //销毁栈
262 Status DestroyStack(SqStack& S)
263 {
264     if (!S.base)
265     {
266         free(S.base);
267         S.stacksize = 0;
268     }
269
270     return OK;
271 }
272
273 //将栈置为空栈
274 Status ClearStack(SqStack& S)
275 {
276     if (!S.base)
277     {
278         return ERROR;
279     }
280
281     S.top = S.base;
282
283     return OK;
284 }
285
286 //栈判空
287 bool StackEmpty(SqStack& S)
288 {
289     if (!S.base)
290     {
```

```
291         return false;
292     }
293
294     if (S.base == S.top)
295     {
296         return true;
297     }
298     else
299     {
300         return false;
301     }
302 }
303
304 //插入元素e为新的栈顶元素
305 Status Push(SqStack& S, SElemType e)
306 {
307     if (S.top - S.base >= S.stacksize)
308     {
309         S.base = (SElemType *) realloc (S.base, (S.stacksize +
              STACK_INCREMENT_SIZE) * sizeof(SElemType));
310         if (!S.base)
311         {
312             exit(OVERFLOW);
313         }
314
315         S.top = S.base + S.stacksize;
316         S.stacksize += STACK_INCREMENT_SIZE;
317     }
318
319     *(S.top++) = e;
320
321     return OK;
322 }
323
324 //删除栈顶元素并用e返回其值
325 SElemType Pop(SqStack& S)
326 {
327     if (S.base == S.top)
328     {
329         return NULL;
330     }
331
332     return *--S.top;
333 }
334
335 SElemType GetTop(SqStack& S)
336 {
337     if (S.base == S.top)
338     {
```

```
339         return NULL;
340     }
341
342     return *(S.top - 1);
343 }
344
345
346 void PrintStack(SqStack& S)
347 {
348     SElemType *p = S.base;
349
350     PrintStar();
351     while (p < S.top)
352     {
353         printf("%c ", (*p)->data);
354         ++p;
355     }
356     putchar(10);
357     PrintStar();
358 }
359
360 void PrintStar(void)
361 {
362     printf("**************************\n");
363 }
```