B62002Y-01 数据结构 2017 年 3 月 22 日

作业二

主讲教师：金蓓弘 张远航 2015K8009929045

第 3 章 栈和队列

3.3 运行结果为`stack`.

3.7 具体操作过程如下表所示.

步骤	OPTR 栈	OPND 栈	输入字符	主要操作
1	#		A-B×C/D+E↑F#	Push(OPND, 'A')
2	#	A	-B×C/D+E↑F#	Push(OPTR, '-')
3	#-	A B	B×C/D+E↑F#	Push(OPND, 'B')
4	#-×	A B	×C/D+E↑F#	Push(OPTR, '×')
5	#=×	A B C	C/D+E↑F#	Push(OPND, 'C')
6	#-×	A B C	/D+E↑F#	Operate('B', '×', 'C')
7	#-/	A B×C	/D+E↑F#	Push(OPTR, '/')
8	#-/	A B×C D	D+E↑F#	Push(OPND, 'D')
9	#-/	A B×C D	+E↑F#	Operate('B×C', '/', 'D')
10	#-	A B×C/D	+E↑F#	Operate('A', '-', 'B×C/D')
11	#	A-B×C/D	+E↑F#	Push(OPTR, '+')
12	#+	A-B×C/D	E↑F#	Push(OPND, 'E')
13	#+	A-B×C/D E	↑F#	Push(OPTR, '↑')
14	#+↑	A-B×C/D E	F#	Push(OPND, 'F')
15	#+↑	A-B×C/D E F	#	Operate('E', '↑', 'F')
16	#+	A-B×C/D E^F	#	Operate('A-B×C/D', '+', 'E^F')
17	#	A-B×C/D+E^F	#	Return(GetTop(OPND))

3.10 利用栈改写局部代码（C 实现）如下：

```
1  void test(int *sum) {
2      SqStack s;
3      int x;
4
5      InitStack(&s);
6      do {
7          scanf("%d", &x);
```

```
8           Push(&s, &x);
9       } while (x != 0);
10      while (!StackEmpty(&s)) {
11          Pop(&s, &x);
12          *sum += x;
13          printf("%d\n", *sum);
14      }
15      DestoryStack(&s);
16  }
```

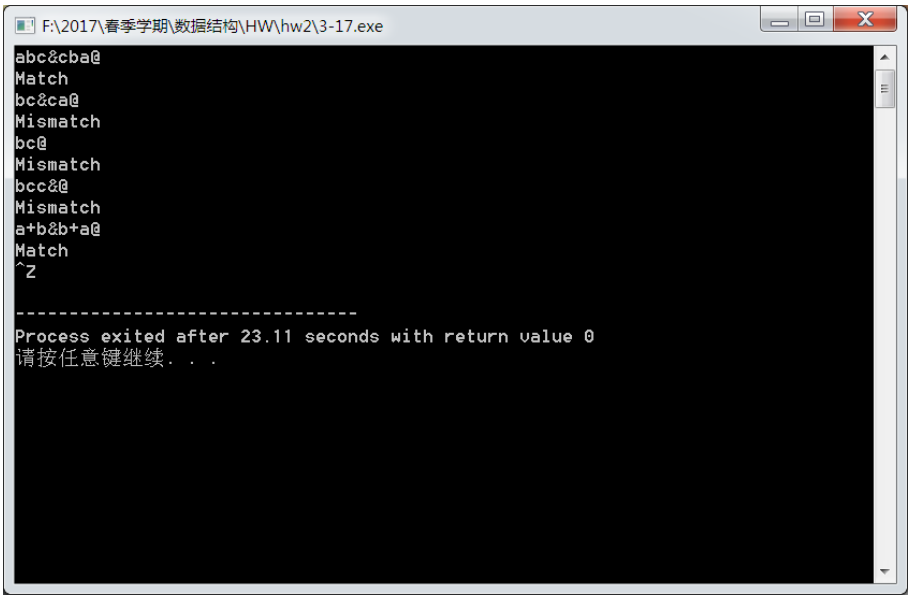
3.17 代码如下.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define OK 1
5   #define ERROR 0
6   #define OVERFLOW -2
7
8   #define TRUE 1
9   #define FALSE 0
10
11  #define STACK_INIT_SIZE 100
12  #define STACKINCREMENT 10
13
14  typedef char SElemType;
15  typedef int bool;
16  typedef int Status;
17
18  typedef struct {
19      SElemType *base;
20      SElemType *top;
21      int stacksize;
22  } SqStack;
23
24  Status InitStack(SqStack *S) {
25      S -> base = (SElemType *)malloc(STACK_INIT_SIZE * sizeof(SElemType));
26      if (!S -> base) exit(OVERFLOW);
27      S -> top = S -> base;
28      S -> stacksize = STACK_INIT_SIZE;
29
30      return OK;
31  }
32
33  Status Push(SqStack *S, SElemType e) {
34      if (S -> top-S -> base == S->stacksize) {
35          S -> base = (SElemType *)realloc(S -> base, (S -> stacksize +
```

```
            STACKINCREMENT) * sizeof(SElemType));
36          if (!S -> base) exit (OVERFLOW);
37          S -> top = S -> base + S -> stacksize;
38          S -> stacksize += STACKINCREMENT;
39      }
40      *(S -> top++) = e;
41
42      return OK;
43  }
44
45  Status Pop(SqStack *S, SElemType *e) {
46      if (StackEmpty(S)) {
47          printf("Empty stack\n");
48          return ERROR;
49      }
50      *e = *(--S -> top);
51
52      return OK;
53  }
54
55  bool StackEmpty(SqStack *S) {
56      return (S -> top == S -> base)? TRUE: FALSE;
57  }
58
59  bool matchPattern(char s[]) {
60      int i = 0;
61      SqStack S;
62      SElemType x;
63
64      InitStack(&S);
65      while (s[i] != '&' && s[i] != '@') {
66          Push(&S, s[i]);
67          i++;
68      }
69      if (s[i] == '@') return FALSE;
70      ++i;
71      while (!StackEmpty(&S)) {
72          Pop(&S, &x);
73          if (x != s[i]) return FALSE;
74          i++;
75      }
76      if (s[i] == '@') return TRUE;
77
78      return FALSE;
79  }
80
81  int main() {
```

```
82      char s[100];
83
84      while (scanf("%s", &s) != -1)
85          printf("%s\n", matchPattern(s)? "Match": "Mismatch");
86
87      return 0;
88  }
```

依次测试下列字符串：`abc&cba@`，`bc&ca@`，`bc@`，`bcc&@`，`a+b&b+a@`：



3.20 (Flood-Fill 算法，栈实现)

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define OK 1
5   #define ERROR 0
6   #define OVERFLOW -2
7
8   #define TRUE 1
9   #define FALSE 0
10
11  #define STACK_INIT_SIZE 100
12  #define STACKINCREMENT 10
13
14  #define SIZE 10
15
16  typedef int bool;
17  typedef int Status;
18
19  typedef struct{
```

```
20      int x, y;
21  } PosType;
22
23  typedef struct{
24      int color;
25      bool visited;
26      PosType seat;
27  } SElemType;
28
29  typedef struct {
30      SElemType *base;
31      SElemType *top;
32      int stacksize;
33  } SqStack;
34
35  int m, n;
36
37  Status InitStack(SqStack *S) {
38      S -> base = (SElemType *)malloc(STACK_INIT_SIZE * sizeof(SElemType));
39      if (!S -> base) exit(OVERFLOW);
40      S -> top = S -> base;
41      S -> stacksize = STACK_INIT_SIZE;
42
43      return OK;
44  }
45
46  Status Push(SqStack *S, SElemType e) {
47      if (S -> top-S -> base == S->stacksize) {
48          S -> base = (SElemType *)realloc(S -> base, (S -> stacksize +
                STACKINCREMENT) * sizeof(SElemType));
49          if (!S -> base) exit (OVERFLOW);
50          S -> top = S -> base + S -> stacksize;
51          S -> stacksize += STACKINCREMENT;
52      }
53      *(S -> top++) = e;
54
55      return OK;
56  }
57
58  Status Pop(SqStack *S, SElemType *e) {
59      if (StackEmpty(S)) {
60          printf("Empty stack\n");
61          return ERROR;
62      }
63      *e = *(--S -> top);
64
65      return OK;
```

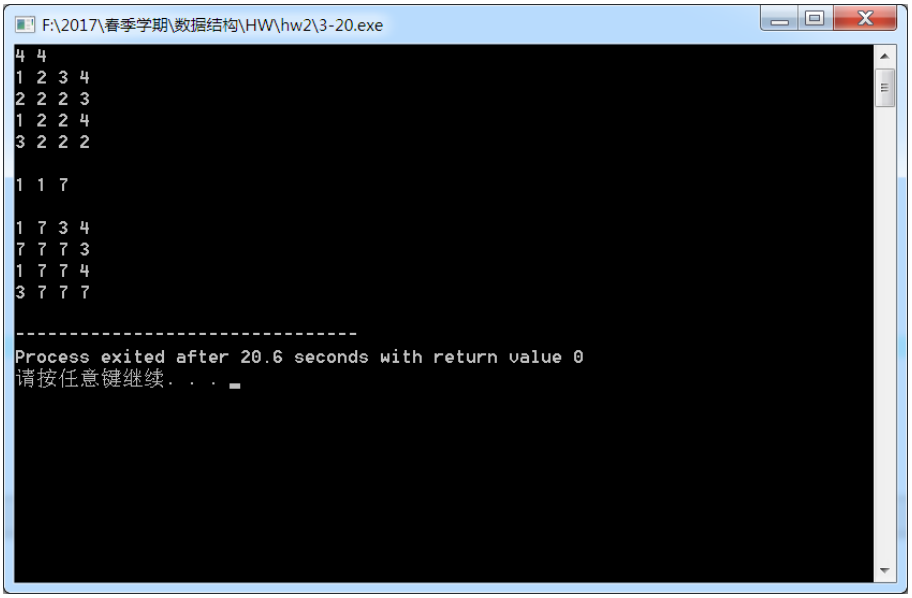
```
66  }
67
68  bool StackEmpty(SqStack *S) {
69      return (S -> top == S -> base)? TRUE: FALSE;
70  }
71
72  void FloodFill(SElemType image[SIZE][SIZE], PosType source, int fillColor) {
73      SqStack s;
74      InitStack(&s);
75      SElemType e;
76      int oldColor = image[source.x][source.y].color;
77
78      Push(&s, image[source.x][source.y]);
79      while (!StackEmpty(&s)) {
80          Pop(&s, &e);
81          source = e.seat;
82          image[source.x][source.y].color = fillColor;
83          image[source.x][source.y].visited = TRUE;
84
85          if (source.x < m && !image[source.x+1][source.y].visited && image[
                source.x+1][source.y].color == oldColor)
86              Push(&s, image[source.x+1][source.y]);
87          if (source.x > 0 && !image[source.x-1][source.y].visited && image[
                source.x-1][source.y].color == oldColor)
88              Push(&s,image[source.x-1][source.y]);
89          if (source.y < n && !image[source.x][source.y+1].visited && image[
                source.x][source.y+1].color == oldColor)
90              Push(&s, image[source.x][source.y+1]);
91          if (source.y > 0 && !image[source.x][source.y-1].visited && image[
                source.x][source.y-1].color == oldColor)
92              Push(&s, image[source.x][source.y-1]);
93      }
94  }
95
96  void PrintImage(SElemType image[SIZE][SIZE]) {
97      int i, j;
98
99      printf("\n");
100     for(i = 0; i < m; i++){
101         for(j = 0; j < n; j++)
102             printf("%d ", image[i][j].color);
103         printf("\n");
104     }
105 }
106
107 int main() {
108     int i, j;
```

```
109     SElemType I[SIZE][SIZE];
110     int fillColor;
111     PosType source;
112
113     for(i = 0; i < SIZE; i++)
114         for (j = 0; j < SIZE; j++) {
115             I[i][j].seat.x = i;
116             I[i][j].seat.y = j;
117             I[i][j].visited = 0;
118             I[i][j].color = 0;
119         }
120
121     scanf("%d%d", &m, &n);
122     for (i = 0; i < m; i++)
123         for (j = 0; j < n; j++)
124             scanf("%d", &I[i][j].color);
125
126     printf("\n");
127     scanf("%d%d%d", &source.x, &source.y, &fillColor);
128
129     FloodFill(I, source, fillColor);
130     PrintImage(I);
131
132     return 0;
133 }
```

读入一个 4×4 图像

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 2 & 2 & 3 \\ 1 & 2 & 2 & 4 \\ 3 & 2 & 2 & 2 \end{matrix}'$$

以 $(1,1)$ 为原点，将同一区域中点的颜色由 2 置换为 7：



3.21 代码如下.

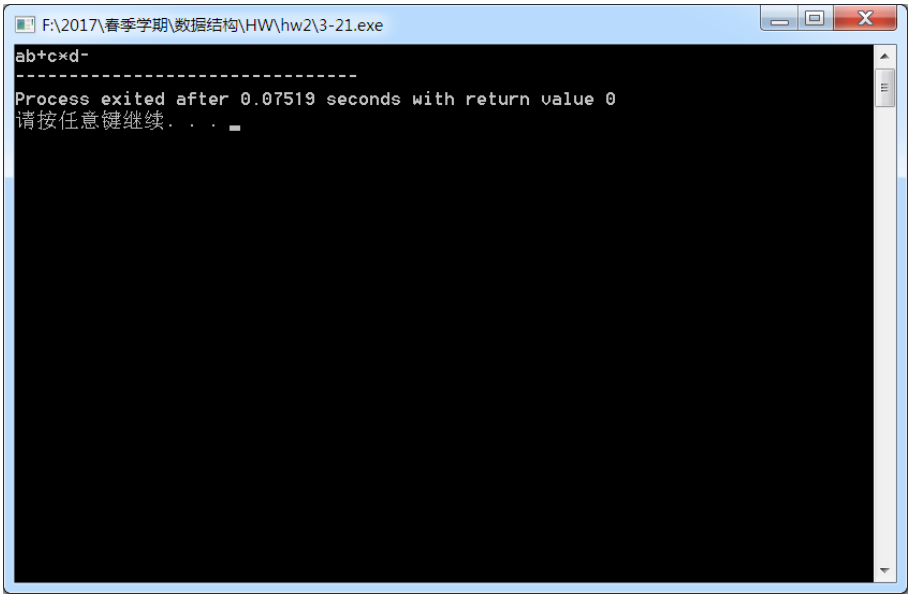
```
#include <stdio.h>
#include <stdlib.h>

#define OPSETSIZE 7

#define OK 1
#define ERROR 0

#define STACK_INIT_SIZE 100
#define STACKINCREMENT 10


typedef struct Node{
    char data;
    struct Node* next;
} Node, *cur;

typedef struct {
    cur front;
    cur rear;
} LinkQueue;


typedef struct{
    char *base;
    char *top;
    int stacksize;
} SqStack;

```

```
30 typedef int Status;
31
32 Status InitQueue(LinkQueue *Q){
33     Q -> front = Q -> rear = (cur)malloc(sizeof(Node));
34     Q -> front -> next = NULL;
35     return OK;
36 }
37
38
39 Status InitStack(SqStack *S){
40     S -> base = (char*)malloc(STACK_INIT_SIZE*sizeof(char));
41     S -> top = S -> base;
42     S -> stacksize = STACK_INIT_SIZE;
43     return OK;
44 }
45
46 Status Pop(SqStack *S, char *e){
47     if (S -> top == S -> base)
48         return ERROR;
49     *e = *--S -> top;
50     return OK;
51 }
52
53 char GetTop(SqStack *S){
54     if (S -> top == S -> base) {
55         return '\0';
56     }
57
58     return *(S -> top-1);
59 }
60
61 Status Push(SqStack *S, char e){
62     if (S -> top-S -> base >= S -> stacksize){
63         S -> base = (char*)realloc(S -> base, (S -> stacksize+STACKINCREMENT)
              * sizeof(char));
64         S -> top = S -> base+S -> stacksize;
65         S -> stacksize += STACKINCREMENT;
66     }
67
68     *S -> top++ = e;
69     return OK;
70 }
71
72 Status EnQueue(LinkQueue *Q, char e){
73     cur p;
74
75     p = (cur)malloc(sizeof(Node));
```

```
76      p -> data = e;
77      p -> next = NULL;
78      Q -> rear -> next = p;
79      Q -> rear = p;
80
81      return OK;
82  }
83
84  Status DeQueue(LinkQueue *Q, char *e) {
85      if (Q -> front == Q -> rear)
86          return ERROR;
87      cur p = Q -> front -> next;
88      *e = p -> data;
89      Q -> front -> next = p -> next;
90      if (Q -> rear == p)
91          Q -> rear = Q -> front;
92
93      free(p);
94      return OK;
95  }
96
97  char OPSET[OPSETSIZE] = {'+' , '-' , '*' , '/' , '(' , ')' , '#'};
98
99  char Prior[OPSETSIZE][OPSETSIZE] = {
100     '>', '>', '<', '<', '<', '>', '>',
101     '>', '>', '<', '<', '<', '>', '>',
102     '>', '>', '>', '>', '<', '>', '>',
103     '>', '>', '>', '>', '<', '>', '>',
104     '<', '<', '<', '<', '<', '=', ' ',
105     '>', '>', '>', '>', ' ', '>', '>',
106     '<', '<', '<', '<', '<', ' ', '='
107 };
108
109 int ReturnOpOrd(char op, char* TestOp) {
110     int i;
111
112     for(i = 0; i< OPSETSIZE; i++) {
113         if (op == TestOp[i])
114             return i; }
115     return -1;
116 }
117
118 char precede(char Aop, char Bop) {
119     return Prior[ReturnOpOrd(Aop, OPSET)][ReturnOpOrd(Bop, OPSET)];
120 }
121
122 LinkQueue ReversePolish(char a[]) {
```

```
123     int i = 0;
124     char c = a[i], e;
125 
126     LinkQueue result;
127     SqStack Op;
128     InitQueue(&result);
129     InitStack(&Op);
130     Push(&Op, '#');
131     while (c != '#' || GetTop(&Op) != '#') {
132         if (ReturnOpOrd(c,OPSET) == -1) {
133             EnQueue(&result, c);
134             c = a[++i];
135         }
136         else
137             switch (precede(GetTop(&Op), c)) {
138                 case '<':
139                     Push(&Op, c);
140                     c = a[++i];
141                     break;
142                 case '=':
143                     Pop(&Op,&e);
144                     c = a[++i];
145                     break;
146                 case '>':
147                     Pop(&Op,&e);
148                     EnQueue(&result, e);
149                     break;
150             }
151     }
152     return result;
153 }
154 
155 int main() {
156     char a[100] = "(a+b)*c-d#", ch;
157     LinkQueue result = ReversePolish(a);
158 
159     while (result.front != result.rear) {
160         DeQueue(&result, &ch);
161         printf("%c", ch);
162     }
163 
164     return 0;
165 }
```

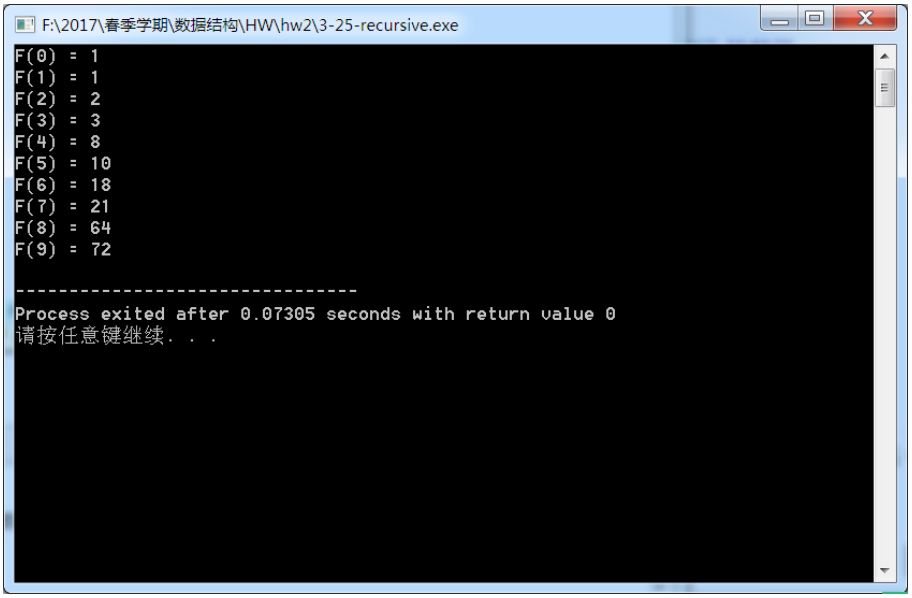
将表达式 (a+b)*c-d 转为逆波兰表达式（以井号结尾）：



3.25 递归代码如下：

```
1  #include <stdio.h>
2  #define ERROR -1
3
4  int F(int n) {
5      if (n < 0)
6          return ERROR;
7      if (n == 0)
8          return n+1;
9
10     return n * F(n/2);
11 }
12
13 int main() {
14     int n;
15
16     for (n = 0; n < 10; n++)
17         printf("F(%d) = %d\n", n, F(n));
18
19     return 0;
20 }
```

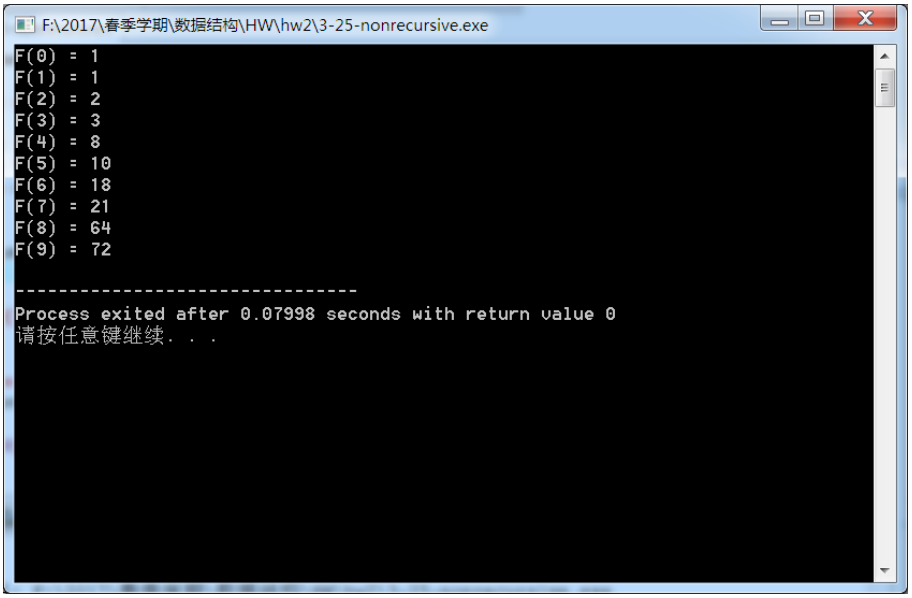
试求 $F(0)$ 至 $F(9)$ 如下：



改为非递归如下：

```
#include <stdio.h>
#define ERROR -1

int F(int n) {
    if (n < 0)
        return ERROR;

    int ans = 1;

    while (n > 0) {
        ans *= n;
        n = n/2;
    }

    return ans;
}

int main() {
    int n;

    for (n = 0; n < 10; n++)
        printf("F(%d) = %d\n", n, F(n));

    return 0;
}
```

试求 $F(0)$ 至 $F(9)$ 如下，结果同上：



3.31 代码如下.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define OK 1
5  #define ERROR 0
6
7  #define INCREMENT 10
8  #define STACK_INIT_SIZE 100
9
10 typedef int Status;
11
12 typedef struct Node {
13     char data;
14     struct Node *next;
15 } Node, *cur;
16
17 // this two-way list structure allows exiting on both sides
18 typedef struct{
19     Node *front;
20     Node *rear;
21 } TwoWayList;
22
23 Status Push(TwoWayList *S, char e) {
24     cur p;
25
26     p = (cur)malloc(sizeof(Node));
27     p -> data = e;
28     p -> next = NULL;
29     S -> rear -> next = p;
```

```
30      S -> rear = p;
31      return OK;
32  }
33
34  Status DeFront(TwoWayList *S, char *e) {
35      cur p;
36
37      if (S -> rear == S -> front) return ERROR;
38      p = S -> front -> next;
39      *e = p -> data;
40      S -> front -> next = p -> next;
41      if (S -> rear == p)
42          S -> rear = S -> front;
43      free(p);
44
45      return OK;
46  }
47
48  Status InitStack(TwoWayList *S) {
49      S -> front = S -> rear =  (cur)malloc(sizeof(Node));
50      S -> front -> next = NULL;
51
52      return OK;
53  }
54
55  Status DeRear(TwoWayList *S, char *e) {
56      cur p, prior = S -> front;
57
58      if (S -> rear == S -> front) return ERROR;
59      p = S -> rear;
60      *e = p -> data;
61      while (prior -> next != S -> rear) prior = prior -> next;
62      S -> rear = prior;
63      free(p);
64
65      return OK;
66  }
67
68  // checks if a given string is a palindrome
69  Status isPalindrome(char a[]) {
70      TwoWayList S;
71      int i = 0;
72      char f, r;
73
74      InitStack(&S);
75      for(; a[i] != '@'; i++)
76          Push(&S, a[i]);
```

```
77
78      while (S.rear != S.front) {
79          DeFront(&S, &f);
80          // length is odd
81          if (S.rear == S.front) return OK;
82          DeRear(&S, &r);
83          if (f != r) return ERROR;
84      }
85
86      return OK;
87  }
88
89  int main() {
90      char s[100];
91
92      while (scanf("%s", &s) != -1)
93          printf(isPalindrome(s)? "Yes\n": "No\n");
94
95      return 0;
96  }
```

依次测试下列字符串：`abba@`，`abcba@`，`abcde@`，`ababab@`：