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$\boxed{\texttt{stack}}$.

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	#		$\underline{\text{A}}$-B×C/D+E↑F#	Push(OPND, ‘A’)
2	#	A	$\underline{\text{-}}$B×C/D+E↑F#	Push(OPTR, ‘-’)
3	#-	A B	$\underline{\text{B}}$×C/D+E↑F#	Push(OPND, ‘B’)
4	#-×	A B	$\underline{\times}$C/D+E↑F#	Push(OPTR, ‘×’)
5	#=×	A B C	$\underline{\text{C}}$/D+E↑F#	Push(OPND, ‘C’)
6	#-×	A B C	$\underline{/}$D+E↑F#	Operate(‘B’, ‘×’, ‘C’)
7	#-/	A B×C	/D+E↑F#	Push(OPTR, ‘/’)
8	#-/	A B×C D	$\underline{\text{D}}$+E↑F#	Push(OPND, ‘D’)
9	#-/	A B×C D	$\underline{+}$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	$\underline{\text{E}}$↑F#	Push(OPND, ‘E’)
13	#+	A-B×C/D E	$\underline{\uparrow}$F#	Push(OPTR, ‘↑’)
14	#+↑	A-B×C/D E	F#	Push(OPND, ‘F’)
15	#+↑	A-B×C/D E F	$\underline{\#}$	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：

```
void test(int *sum) {
    SqStack s;
    int x;

    InitStack(&s);
    do {
        scanf("%d", &x);
```

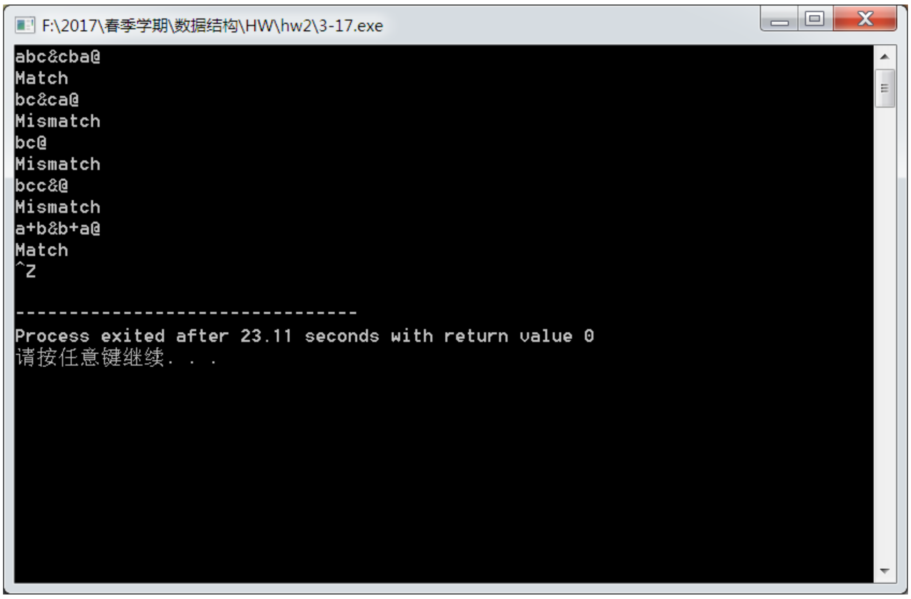
```c
        Push(&s, &x);
    } while (x != 0);
    while (!StackEmpty(&s)) {
        Pop(&s, &x);
        *sum += x;
        printf("%d\n", *sum);
    }
    DestoryStack(&s);
}
```

3.17 代码如下.

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

#define OK 1
#define ERROR 0
#define OVERFLOW -2

#define TRUE 1
#define FALSE 0

#define STACK_INIT_SIZE 100
#define STACKINCREMENT 10

typedef char SElemType;
typedef int bool;
typedef int Status;

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

Status InitStack(SqStack *S) {
    S -> base = (SElemType *)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if (!S -> base) exit(OVERFLOW);
    S -> top = S -> base;
    S -> stacksize = STACK_INIT_SIZE;

    return OK;
}

Status Push(SqStack *S, SElemType e) {
    if (S -> top-S -> base == S->stacksize) {
        S -> base = (SElemType *)realloc(S -> base, (S -> stacksize +
            STACKINCREMENT) * sizeof(SElemType));
        if (!S -> base) exit (OVERFLOW);
        S -> top = S -> base + S -> stacksize;
        S -> stacksize += STACKINCREMENT;
    }
    *(S -> top++) = e;

    return OK;
}

Status Pop(SqStack *S, SElemType *e) {
    if (StackEmpty(S)) {
        printf("Empty stack\n");
        return ERROR;
    }
    *e = *(--S -> top);

    return OK;
}

bool StackEmpty(SqStack *S) {
    return (S -> top == S -> base)? TRUE: FALSE;
}

bool matchPattern(char s[]) {
    int i = 0;
    SqStack S;
    SElemType x;

    InitStack(&S);
    while (s[i] != '&' && s[i] != '@') {
        Push(&S, s[i]);
        i++;
    }
    if (s[i] == '@') return FALSE;
    ++i;
    while (!StackEmpty(&S)) {
        Pop(&S, &x);
        if (x != s[i]) return FALSE;
        i++;
    }
    if (s[i] == '@') return TRUE;

    return FALSE;
}

int main() {
```

```c
    char s[100];

    while (scanf("%s", &s) != -1)
        printf("%s\n", matchPattern(s)? "Match": "Mismatch");

    return 0;
}
```

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



```
abc&cba@
Match
bc&ca@
Mismatch
bc@
Mismatch
bcc&@
Mismatch
a+b&b+a@
Match
^Z

--------------------------------
Process exited after 23.11 seconds with return value 0
请按任意键继续. . .
```

3.20 (Flood-Fill 算法，栈实现)

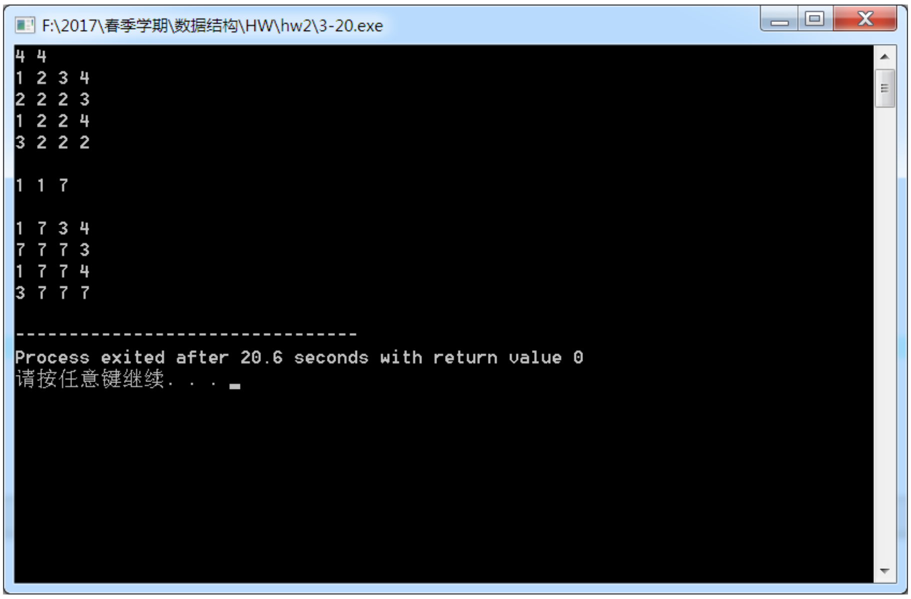
```c
#include <stdio.h>
#include <stdlib.h>

#define OK 1
#define ERROR 0
#define OVERFLOW -2

#define TRUE 1
#define FALSE 0

#define STACK_INIT_SIZE 100
#define STACKINCREMENT 10

#define SIZE 10

typedef int bool;
typedef int Status;

typedef struct{
    int x, y;
} PosType;

typedef struct{
    int color;
    bool visited;
    PosType seat;
} SElemType;

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

int m, n;

Status InitStack(SqStack *S) {
    S -> base = (SElemType *)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if (!S -> base) exit(OVERFLOW);
    S -> top = S -> base;
    S -> stacksize = STACK_INIT_SIZE;

    return OK;
}

Status Push(SqStack *S, SElemType e) {
    if (S -> top-S -> base == S->stacksize) {
        S -> base = (SElemType *)realloc(S -> base, (S -> stacksize +
            STACKINCREMENT) * sizeof(SElemType));
        if (!S -> base) exit (OVERFLOW);
        S -> top = S -> base + S -> stacksize;
        S -> stacksize += STACKINCREMENT;
    }
    *(S -> top++) = e;

    return OK;
}

Status Pop(SqStack *S, SElemType *e) {
    if (StackEmpty(S)) {
        printf("Empty stack\n");
        return ERROR;
    }
    *e = *(--S -> top);

    return OK;
```

```c
}

bool StackEmpty(SqStack *S) {
    return (S -> top == S -> base)? TRUE: FALSE;
}

void FloodFill(SElemType image[SIZE][SIZE], PosType source, int fillColor) {
    SqStack s;
    InitStack(&s);
    SElemType e;
    int oldColor = image[source.x][source.y].color;

    Push(&s, image[source.x][source.y]);
    while (!StackEmpty(&s)) {
        Pop(&s, &e);
        source = e.seat;
        image[source.x][source.y].color = fillColor;
        image[source.x][source.y].visited = TRUE;

        if (source.x < m && !image[source.x+1][source.y].visited && image[source.x+1][source.y].color == oldColor)
            Push(&s, image[source.x+1][source.y]);
        if (source.x > 0 && !image[source.x-1][source.y].visited && image[source.x-1][source.y].color == oldColor)
            Push(&s,image[source.x-1][source.y]);
        if (source.y < n && !image[source.x][source.y+1].visited && image[source.x][source.y+1].color == oldColor)
            Push(&s, image[source.x][source.y+1]);
        if (source.y > 0 && !image[source.x][source.y-1].visited && image[source.x][source.y-1].color == oldColor)
            Push(&s, image[source.x][source.y-1]);
    }
}

void PrintImage(SElemType image[SIZE][SIZE]) {
    int i, j;

    printf("\n");
    for(i = 0; i < m; i++){
        for(j = 0; j < n; j++)
            printf("%d ", image[i][j].color);
        printf("\n");
    }
}

int main() {
    int i, j;
    SElemType I[SIZE][SIZE];
    int fillColor;
    PosType source;

    for(i = 0; i < SIZE; i++)
        for (j = 0; j < SIZE; j++) {
            I[i][j].seat.x = i;
            I[i][j].seat.y = j;
            I[i][j].visited = 0;
            I[i][j].color = 0;
        }

    scanf("%d%d", &m, &n);
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &I[i][j].color);

    printf("\n");
    scanf("%d%d%d", &source.x, &source.y, &fillColor);

    FloodFill(I, source, fillColor);
    PrintImage(I);

    return 0;
}
```

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：



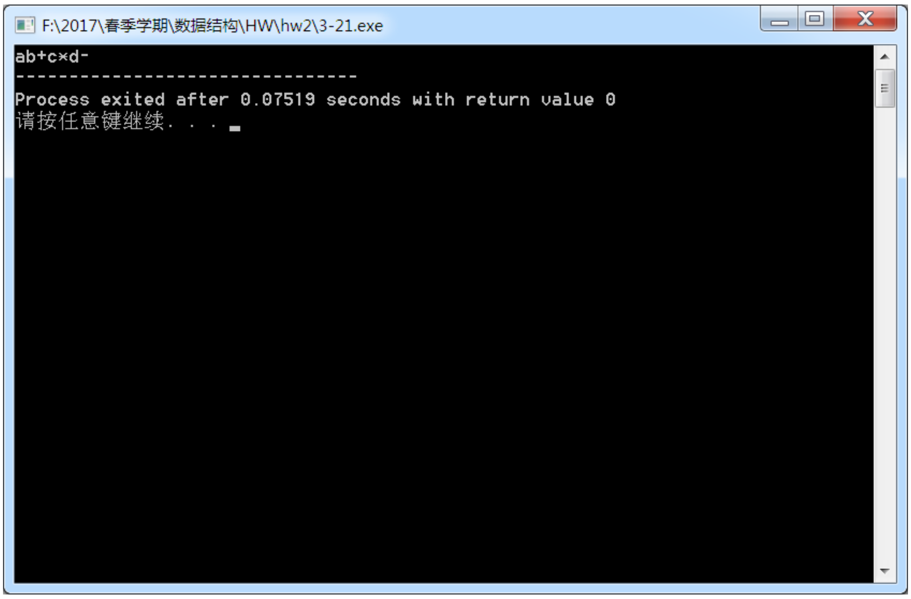
```
4 4
1 2 3 4
2 2 2 3
1 2 2 4
3 2 2 2

1 1 7

1 7 3 4
7 7 7 3
1 7 7 4
3 7 7 7

--------------------------------
Process exited after 20.6 seconds with return value 0
请按任意键继续. . .
```

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;

```

```c
typedef int Status;

Status InitQueue(LinkQueue *Q){
    Q -> front = Q -> rear = (cur)malloc(sizeof(Node));
    Q -> front -> next = NULL;
    return OK;
}


Status InitStack(SqStack *S){
    S -> base = (char*)malloc(STACK_INIT_SIZE*sizeof(char));
    S -> top = S -> base;
    S -> stacksize = STACK_INIT_SIZE;
    return OK;
}

Status Pop(SqStack *S, char *e){
    if (S -> top == S -> base)
        return ERROR;
    *e = *--S -> top;
    return OK;
}

char GetTop(SqStack *S){
    if (S -> top == S -> base) {
        return '\0';
    }

    return *(S -> top-1);
}

Status Push(SqStack *S, char e){
    if (S -> top-S -> base >= S -> stacksize){
        S -> base = (char*)realloc(S -> base, (S -> stacksize+STACKINCREMENT)
            * sizeof(char));
        S -> top = S -> base+S -> stacksize;
        S -> stacksize += STACKINCREMENT;
    }

    *S -> top++ = e;
    return OK;
}

Status EnQueue(LinkQueue *Q, char e){
    cur p;

    p = (cur)malloc(sizeof(Node));
    p -> data = e;
    p -> next = NULL;
    Q -> rear -> next = p;
    Q -> rear = p;

    return OK;
}

Status DeQueue(LinkQueue *Q, char *e) {
    if (Q -> front == Q -> rear)
        return ERROR;
    cur p = Q -> front -> next;
    *e = p -> data;
    Q -> front -> next = p -> next;
    if (Q -> rear == p)
        Q -> rear = Q -> front;

    free(p);
    return OK;
}

char OPSET[OPSETSIZE] = {'+' , '-' , '*' , '/' , '(' , ')' , '#'};

char Prior[OPSETSIZE][OPSETSIZE] = {
    '>', '>', '<', '<', '<', '>', '>',
    '>', '>', '<', '<', '<', '>', '>',
    '>', '>', '>', '>', '<', '>', '>',
    '>', '>', '>', '>', '<', '>', '>',
    '<', '<', '<', '<', '<', '=', ' ',
    '>', '>', '>', '>', ' ', '>', '>',
    '<', '<', '<', '<', '<', ' ', '='
};

int ReturnOpOrd(char op, char* TestOp) {
    int i;

    for(i = 0; i< OPSETSIZE; i++) {
        if (op == TestOp[i])
            return i; }
    return -1;
}

char precede(char Aop, char Bop) {
    return Prior[ReturnOpOrd(Aop, OPSET)][ReturnOpOrd(Bop, OPSET)];
}

LinkQueue ReversePolish(char a[]) {
    int i = 0;
    char c = a[i], e;

    LinkQueue result;
    SqStack Op;
    InitQueue(&result);
    InitStack(&Op);
    Push(&Op, '#');
    while (c != '#' || GetTop(&Op) != '#') {
        if (ReturnOpOrd(c,OPSET) == -1) {
            EnQueue(&result, c);
            c = a[++i];
        }
        else
            switch (precede(GetTop(&Op), c)) {
                case '<':
                    Push(&Op, c);
                    c = a[++i];
                    break;
                case '=':
                    Pop(&Op,&e);
                    c = a[++i];
                    break;
                case '>':
                    Pop(&Op,&e);
                    EnQueue(&result, e);
                    break;
            }
    }
    return result;
}

int main() {
    char a[100] = "(a+b)*c-d#", ch;
    LinkQueue result = ReversePolish(a);

    while (result.front != result.rear) {
        DeQueue(&result, &ch);
        printf("%c", ch);
    }

    return 0;
}
```

将表达式 (a+b)*c-d 转为逆波兰表达式（以井号结尾）：

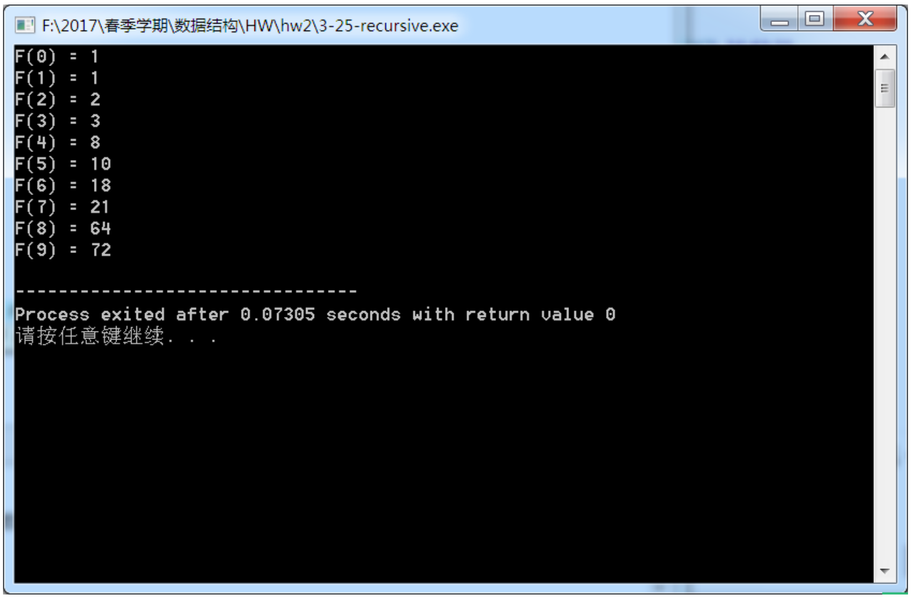


```
ab+c*d-
--------------------------------
Process exited after 0.07519 seconds with return value 0
请按任意键继续. . .
```

3.25 递归代码如下：

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

int F(int n) {
    if (n < 0)
        return ERROR;
    if (n == 0)
        return n+1;

    return n * F(n/2);
}

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：

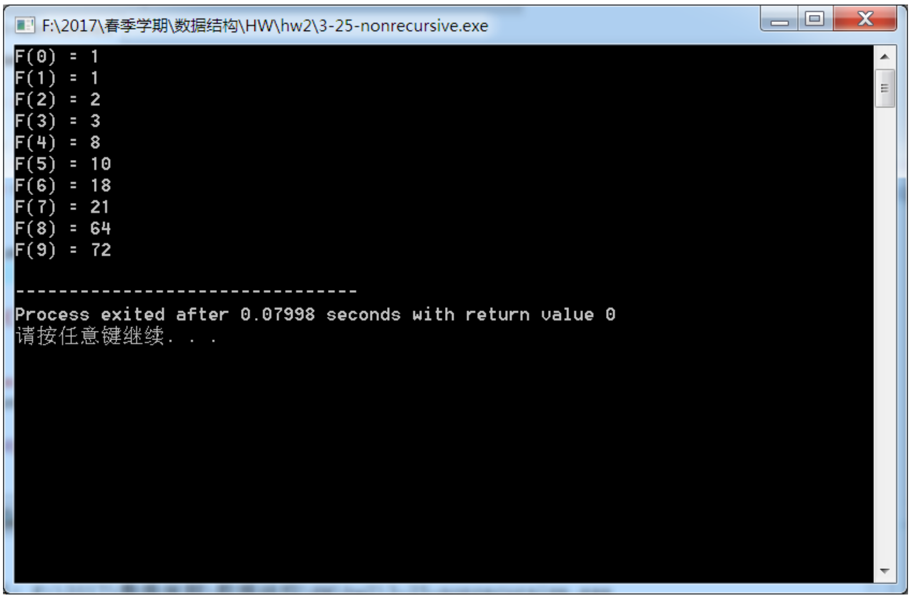


```
F(0) = 1
F(1) = 1
F(2) = 2
F(3) = 3
F(4) = 8
F(5) = 10
F(6) = 18
F(7) = 21
F(8) = 64
F(9) = 72

--------------------------------
Process exited after 0.07305 seconds with return value 0
请按任意键继续. . .
```

改为非递归如下：

```c
#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：



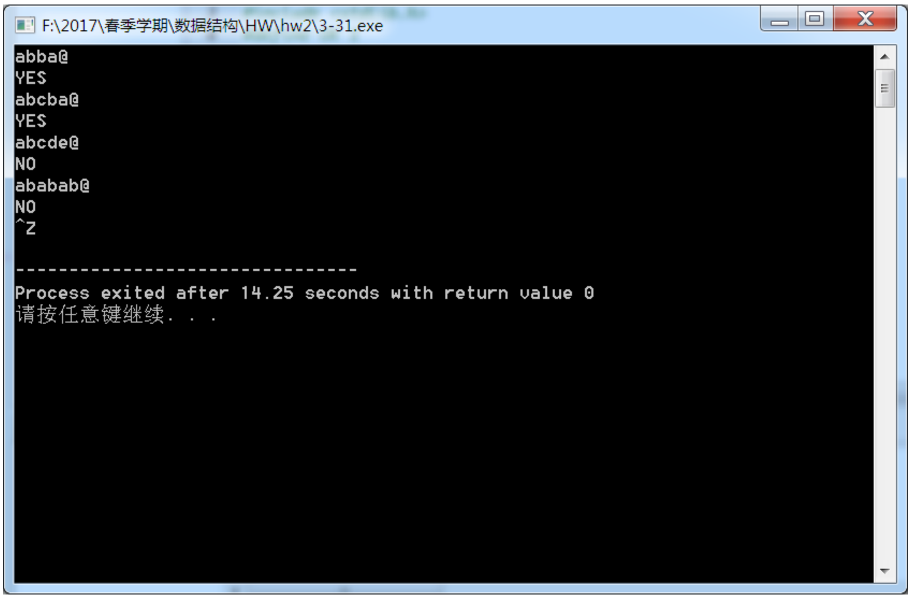
```
F(0) = 1
F(1) = 1
F(2) = 2
F(3) = 3
F(4) = 8
F(5) = 10
F(6) = 18
F(7) = 21
F(8) = 64
F(9) = 72

--------------------------------
Process exited after 0.07998 seconds with return value 0
请按任意键继续. . .
```

3.31 代码如下.

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

#define OK 1
#define ERROR 0

#define INCREMENT 10
#define STACK_INIT_SIZE 100

typedef int Status;

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

// this two-way list structure allows exiting on both sides
typedef struct{
    Node *front;
    Node *rear;
} TwoWayList;

Status Push(TwoWayList *S, char e) {
    cur p;

    p = (cur)malloc(sizeof(Node));
    p -> data = e;
    p -> next = NULL;
    S -> rear -> next = p;
    S -> rear = p;
    return OK;
}

Status DeFront(TwoWayList *S, char *e) {
    cur p;

    if (S -> rear == S -> front) return ERROR;
    p = S -> front -> next;
    *e = p -> data;
    S -> front -> next = p -> next;
    if (S -> rear == p)
        S -> rear = S -> front;
    free(p);

    return OK;
}

Status InitStack(TwoWayList *S) {
    S -> front = S -> rear =  (cur)malloc(sizeof(Node));
    S -> front -> next = NULL;

    return OK;
}

Status DeRear(TwoWayList *S, char *e) {
    cur p, prior = S -> front;

    if (S -> rear == S -> front) return ERROR;
    p = S -> rear;
    *e = p -> data;
    while (prior -> next != S -> rear) prior = prior -> next;
    S -> rear = prior;
    free(p);

    return OK;
}

// checks if a given string is a palindrome
Status isPalindrome(char a[]) {
    TwoWayList S;
    int i = 0;
    char f, r;

    InitStack(&S);
    for(; a[i] != '@'; i++)
        Push(&S, a[i]);
```

```c

    while (S.rear != S.front) {
        DeFront(&S, &f);
        // length is odd
        if (S.rear == S.front) return OK;
        DeRear(&S, &r);
        if (f != r) return ERROR;
    }

    return OK;
}

int main() {
    char s[100];

    while (scanf("%s", &s) != -1)
        printf(isPalindrome(s)? "Yes\n": "No\n");

    return 0;
}
```

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



```
abba@
YES
abcba@
YES
abcde@
NO
ababab@
NO
^Z

--------------------------------
Process exited after 14.25 seconds with return value 0
请按任意键继续. . .
```