3.3

程序输出结果：stack

3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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\*C/D+E^F# | PUSH(OPND,B) |
| 4 | #- | AB | \*C/D+E^F# | PUSH(OPTR,\*) |
| 5 | #-\* | AB | C/D+E^F# | PUSH(OPND,C) |
| 6 | #-\* | ABC | /D+E^F# | operate(B,\*,C) |
| 7 | #- | AG | /D+E^F# | PUSH(OPTR,/) |
| 8 | #-/ | AG | D+E^F# | PUSH(OPND,D) |
| 9 | #-/ | AGD | +E^F# | operate(G,/,D) |
| 10 | #- | AH | +E^F# | operate(A,-,H) |
| 11 | # | I | +E^F# | PUSH(OPTR,+) |
| 12 | #+ | I | E^F# | PUSH(OPND,E) |
| 13 | #+ | IE | ^F# | PUSH(OPTR,^) |
| 14 | #+^ | IE | F# | PUSH(OPTR,F) |
| 15 | #+^ | IEF | # | operate(E,^,F) |
| 16 | #+ | IJ | # | operate(I,+,J) |
| 17 | # | K | # | RETURN |

3.10

#include<stdio.h>

#include<stdlib.h>

#define MAXSTACK 100

typedef int DD;

typedef struct Stack{

DD data[MAXSTACK];

int top;

}Stack;

void Init(Stack \*s);

int StackEmpty(Stack \*s);

int push(Stack \*a, DD x);

int pop(Stack \*a, DD \*x);

void printStack(Stack \*s);

void test(int sum);

int main()

{

int sum = 0;

Stack s;

Init(&s);

int x;

scanf("%d", &x);

while(x>0)

{

if(x > 0)

push(&s, x);

scanf("%d", &x);

}

while(!StackEmpty(&s))

{

pop(&s, &x);

sum += x;

}

printf("%d\n", sum);

return 0;

}

void Init(Stack \*s){

s->top = -1;

}

int StackEmpty(Stack \*s){

return (s->top == -1 ? 1 : 0);

}

int push(Stack \*a, DD x)

{

if(a->top == MAXSTACK - 1)

{

printf("error: stack full, cannot push \n");

return -1;

}

a->data[++(a->top)] = x;

return 1;

}

int pop(Stack \*a, DD \*x)

{

if(a->top == -1)

{

printf("error: Stack empty\n");

return 0;

}

\*x = a->data[a->top--];

return 1;

}

void printStack(Stack \*s)

{

while(s->top != -1)

printf("%d ", s->data[s->top--]);

printf("\n");

}

3.17

#include<stdio.h>

#include<stdlib.h>

#define MAXSTACK 100

typedef char DD;

typedef struct Stack{

DD data[MAXSTACK];

int top;

}Stack;

int StackEmpty(Stack \*s);

void push(Stack \*a, DD x);

void pop(Stack \*a, DD \*x);

int IsReverse();

int main(){

IsReverse() == 1 ? printf("Yes\n") : printf("No\n");

return 0;

}

int IsReverse(){

char e, x;

Stack s;

while((e = getchar()) != '&'){

if(e == '@')

return 0;

push(&s, e);

}

while((e = getchar()) != '@'){

if(StackEmpty(&s))

return 0;

pop(&s, &x);

if(x != e)

return 0;

}

pop(&s, &x);

if(!StackEmpty(&s))

return 0;

return 1;

}

int StackEmpty(Stack \*s){

return (s->top == -1 ? 1 : 0);

}

void push(Stack \*a, DD x)

{

if(a->top == MAXSTACK - 1)

{

printf("error: stack full, cannot push \n");

exit -1;

}

a->data[++(a->top)] = x;

}

void pop(Stack \*a, DD \*x)

{

if(a->top == -1)

{

printf("error: Stack empty\n");

exit -1;

}

\*x = a->data[a->top--];

}

3.18

#include<stdio.h>

#include<stdlib.h>

#define MAXSTACK 100

typedef char DD;

typedef struct Stack{

DD data[MAXSTACK];

int top;

}Stack;

void Init(Stack \*s);

int StackEmpty(Stack \*s);

void push(Stack \*a, DD x);

void pop(Stack \*a, DD \*x);

void printStack(Stack \*s);

int ValidBracket(char \*s);

int main(){

char s[] = "[[]2(/)]";

ValidBracket(s) == 1 ? printf("Valid brackets\n") : printf("Invalid brackets\n");

return 0;

}

int ValidBracket(char \*s){

int i = 0;

char x,\*q;

Stack p;

Init(&p);

for(q = s; \*q; q++){

switch(\*q){

case '(': push(&p, \*q);break;

case '[': push(&p, \*q);break;

case '{': push(&p, \*q);break;

case ')': pop(&p, &x);break;

case ']': pop(&p, &x);break;

case '}': pop(&p, &x);break;

}

}

if(!StackEmpty(&p))

return 0;

return 1;

}

void Init(Stack \*s){

s->top = -1;

}

int StackEmpty(Stack \*s){

return (s->top == -1 ? 1 : 0);

}

void push(Stack \*a, DD x)

{

if(a->top == MAXSTACK - 1)

{

printf("error: stack full, cannot push \n");

exit -1;

}

a->data[++(a->top)] = x;

}

void pop(Stack \*a, DD \*x)

{

if(a->top == -1)

{

printf("error: Stack empty\n");

exit -1;

}

\*x = a->data[a->top--];

}

void printStack(Stack \*s)

{

while(s->top != -1)

printf("%c ", s->data[s->top--]);

printf("\n");

}

3.20

#define M 10

#define N 10

void Repaint(int g[M][N], int i, int j, int color){

int old;

old = g[i][j];

if(i > 0 && g[i-1][j] == old)

g[i-1][j] = color;

if(j > 0 && g[i][j-1] == old)

g[i][j-1] = color;

if(i < M-1 && g[i+1][j] == old)

g[i+1][j] = color;

if(j < N-1 && g[i][j+1]

== old)

g[i][j+1] = color;

}

3.21

3.24

int g(int m, int n, int s)

{

if(m < 0 || n < 0)

{

printf("Invalid m\n");

exit -1;

}

else if(m == 0)

return 0;

else

return n + g(m-1, 2\*n);

}

3.25

递归算法：

int rec\_F(int n)

{

if(n == 0)

return n + 1;

else if(n > 0)

return n\*rec\_F(n/2);

else

{

printf("Invalid n\n");

exit -1;

}

}

非递归算法：

int nonrec\_F(int n)

{

Stack s;

Init(&s);

int a, result;

if(n < 0)

{

printf("Invalid n\n");

exit -1;

}

else if(n == 0)

return n + 1;

else

{

for(a = n; n != 0; a /= 2)

{

push(&s, a);

}

while(!StackEmpty(&s))

{

pop(&s, &a);

result \*= a;

}

}

return result;

}

3.28

#include<stdio.h>

#include<stdlib.h>

typedef struct node \*Queue;

typedef int DD;

struct node{

DD data;

Queue next;

};

void InitQueue(Queue s)

{

s = (Queue)malloc(sizeof(struct node));

s->next = s;

}

void InsertQueue(Queue s, DD x)

{

Queue p;

p = (Queue)malloc(sizeof(struct node));

p->data = x;

p->next = s->next;

s->next = p;

s = p;

}

Queue DeleteQueue(Queue s, DD x)

{

if(s->next == s)

return NULL;

Queue p;

p = s->next->next;

x = p->data;

p = p->next;

s->next->next = p->next;

free(p);

return s;

}

3.31

#include<stdio.h>

int reflex(char s[]);

int main()

{

char s[] = "acbca@";

char t[] = "abc@";

reflex(s) ? printf("yes\n") : printf("no\n");

reflex(t) ? printf("yes\n") : printf("no\n");

return 0;

}

int reflex(char s[])

{

char \*i, \*j;

for(i = s; \*i != '@'; i++);

i--;

for(j = s; \*i == \*j && i >= j; i--, j++);

return (i > j) ? 0 : 1;

}