3-10

typedef struct linklist

{

Elemtype data;

struct linklist \* next;

}linklist;

typedef struct{

linklist \* front, \*rear;

}linkqueue;

//置空

linkqueue \* SetNullQ(linkqueue \* p)

{

p -> front = p -> rear = NULL;

return p;

}

//判空

int EmptyQ(linkqueue \* p)

{

if(p -> front == q -> front)

return 1;

else

return 0;

}

//取队头元素

datatype \* FrontQ(linkqueue \* p)

{

datatype \* ret;

if(EmptyQ(p))

return NULL;

else

{

ret = (datatype\*)malloc(sizeof(datatype));

\*ret = p -> front -> data;

return ret;

}

}

//入队

void EnQueueQ(linkqueue \* p, datatype x)

{

p -> rear -> next = (linklist\*)malloc(sizeof(linklist));

p -> rear = p -> rear -> next;

p -> rear -> data = x;

p -> rear -> next = NULL;

}

//出队

datatype \* DeQueueQ(linklist \* q)

{

datatype \* ret;

linklist \* s;

if(EmptyQ(q))

return NULL;

else

{

ret = (datatype \*)malloc(sizeof(datatype));

s = q -> front -> next;

if(s == NULL)

{

\*ret = q -> front -> data;

q -> front = NULL;

q -> rear = q -> front;

}

else

{

\*ret = q -> front -> data;

q -> front = s;

}

return ret;

}

}

3-11

int dc(linklist \* L, int n)

{

int i;

char s[n/2];

p = L -> next;

for(i = 0; i < n/2; ++i)

{

s[i] = p -> data;

p = p -> next;

}

i--;

if(n % 2 == 1)

p = p -> next;

while(p != NULL && s[i] == p -> data)

{

i--;

p = p -> next;

}

if(i == -1)

return 1;

else

return 0;

}

3-12

bool BracketsCheck(char \* str)

{

char s[100];

int e = 0;

InitStack(s);

int i = 0;

while(str[i] != '\0')

{

if(str[i] == '(')

Push(s, '(');

if(str[i] == ')')

{

if(!Empty(s))

Pop(s, e);

else

return false;

}

++i;

}

if(Empty(s))

return true;

else

return false;

}

3-13

datatype v[max];

int ta, tb;

void initstack(void)

{

ta = -1;

tb = max;

}

void Push(int i, datatype x)

{

if(ta < tb)

{

if(i == 0)

{

v[++ta] = x;

}

if(i == 1)

{

v[--tb] = x

}

}

else

printf("Stack overflow\n");

}

datatype pop(int i)

{

datatype x;

if(i == 0)

{

if(ta > -1)

{

x = v[ta--];

return x;

}

else

return NULL;

}

if(i == 1)

{

if(tb < max)

{

x = v[tb++];

return x;

}

else

return NULL;

}

return NULL;

}

datatype top(int i)

{

datatype x;

if(i == 0)

{

if(ta > -1)

{

x = v[ta];

return x;

}

else

return NULL;

}

if(i == 1)

{

if(tb < max)

{

x = v[tb];

return x;

}

else

return NULL;

}

return NULL;

}

3-14

typedef struct QNode

{

Elemtype data;

struct QNode \* next;

}QNode;

typedef struct Queue

{

QNode \* rear;

}Queue;

Queue \* T;

//置空

void InitStack(Queue \* T)

{

T -> rear -> next -> next = T -> rear -> next;

T -> rear = T -> rear -> next;

}

//入队

Queue \* EnQueue(Queue \* T, Elemtype x)

{

QNode \* s = (QNode\*)malloc(sizeof(QNode));

s -> data = x;

s -> next = T -> rear -> next;

T -> rear -> next = s;

T -> rear = s;

return T;

}

//出队

Queue \* DeQueue(Queue \* T, Elemtype x)

{

if(T -> rear -> next == T -> rear)

return NULL;

else if(T -> rear -> next -> next == T -> rear)

{

x = T -> rear -> data;

T -> rear = T -> rear -> next;

T -> rear -> next = T -> rear;

}

else

{

x = T -> rear -> next -> next -> data;

T -> rear -> next -> next = T -> rear -> next -> next -> next;

}

}

3-16

typedef struct

{

int data[MaxSize];

int quelen;

int rear;

}Sequeue;

//判满

int QueueFull(SeQueue \* Q)

{

if(Q -> quelen == MaxSize)

return 1;

else

return 0;

}

//入队

int EnQueue(SeQueue \* Q, int x)

{

if(QueueFull(Q))

return 0;

rear = (rear + 1) % MaxSize;

Q -> data[rear] = x;

++quelen;

return 1;

}

//出队

int DeQueue(SeQueue \* Q)

{

if(quelen == 0)

return NULL;

int front = (rear - quelen + 1 + MaxSize) % MaxSize;

quelen--;

return Q -> data[front];

}