第二章：

1. 设单链表结点指针域为next，试写出删除链表中指针p所指结点的直接后继的C语言语句。

delete p->next;

1. 设单链表中某指针p所指结点（即p结点）的数据域为data，链指针域为next，请写出在p结点之前插入s结点的操作。

s->next = p->next;

p->next = s;

s->data = s->data + t->data;

t->data = s->data – t->data;

s->data = s->data – t->data;

s = t;

t = t->next;

1. 设计一高效算法，将顺序表的所有元素逆置，要求算法的空间复杂度为O（1）.

for (int i = 0; i < (Seq.len / 2); i++) {

swap(SeqList.data[i], SeqList.data[Seq.len – i – 1];

}

1. 从有序顺序表中删除所有其值重复的元素，使表中所有元素的值均不同。

int i = 0, j = 1;

while (i < len && j < len) {

if (Seq.data[i] == Seq.data[j]) {

j++;

continue;

}

else if (Seq.data[i] < Seq.data[j]) {

Seq.data[++i] = Seq.data[j];

continue;

}

}

Seq.len = i + 1;

1. 从顺序表中删除具有最小值的元素（假设唯一）并由函数返回被删元素的值，空出的位置由最后一个元素填补，若顺序表为空则显示出错并退出。

if (len == 0) {

cout << “ERROR!” << endl;

return;

}

else {

int temp = Seq.data[0];

int pos = 0;

for (int i = 1; i < Seq.len; i++) {

if (Seq.data[i] < temp) {

temp = Seq.data[i];

pos = i;

}

Seq.data[pos] = Seq.data[Seq.len – 1];

return temp;

}

1. 给定两个单链表，编写算法找出两个链表的公共结点

int len1, len2;

len1 = length(List1);

len2 = length(List2);

Node\* head1 = List1.head;

Node\* head2 = List2.head;

if (len1 < len2) {

for (int i = 0; i < (len2 - len1); i++) {

head2 = head2->next;

}

}

else {

for (int i = 0; i < (len1 - len2); i++) {

head1 = head1->next;

}

}

while (head1 != head2 && head1 != NULL) {

head1 = head1->next;

head2 = head2->next;

}

if (head1 != NULL)

return head1;

else

return NULL;

1. 写出下图双链表中对换值为23和15的两个结点相互位置时修改指针的有关语句。（要写出结构体的定义，结点结构可参考：(llink,data,rlink)）

 struct Node {

Node\* llink;

Node\* rlink;

int data;

};

void Switch(Node\* node23, Node\* node15) {

node15->llink = node23->llink;

node15->llink->rlink = node15;

node23->rlink = node15->rlink;

node23->rlink->llink = node23;

node15->rlink = node23;

node23->llink = node15;

}

1. 编写一个算法，将两个递增有序的顺序表L1，L2合并成一个新的递增有序顺序表，返回结果顺序表L

int len1 = length(L1);

int len2 = length(L2);

SeqList.L(len1 + len2);

int pos = 0;

int pos1 = 0;

int pos2 = 0;

while (pos1 < len1 || pos2 < len2) {

//一个已经到头

if (pos1 >= len1) {

L[pos++] = L2[pos2++];

}

else if (pos2 >= len2) {

L[pos++] = L2[pos2++];

}

if (L1[pos1] < L2[pos2]) {

L[pos++] = L1[pos1++];

}

else if (L2[pos2] < L1[pos1]) {

L[pos++] = L2[pos2++];

}

else {

L[pos++] = L1[pos1++];

L[pos++] = L2[pos2++];

}

}

L.len = pos;

return L;

1. 设有一不带头结点的单链表，编程将链表按从小到大的顺序排序。(要求不用另外的数组或结点完成)

int data\_switch = 0;

if (head->data > head->next->data) {

data\_switch = head->data;

head->data = head->next->data;

head->next->data = data\_switch;

}

for (int i = 0; i < len; i++) {

Node\* temp = head;

for (int j = 0; j < len - i; j++) {

if (temp->data > temp->next->data) {

data\_switch = temp->data;

temp->data = temp->next->data;

temp->next->data = data\_switch;

}

temp = temp->next;

}

}

1. 设有两个从小到大排序的带头结点的有序链表。试编写求这两个链表交运算的算法（即L1∩L2）。要求结果链表仍是从小到大排序，但无重复元素。

Node\* temp1 = L1.head;

Node\* temp2 = L2.head;

while (temp1 != NULL && temp2 != NULL) {

if (temp1->data < temp2->data)

temp1 = temp1->next;

else if (temp2->data < temp1->data)

temp2 = temp2->next;

else {

L.insert(L, temp1->data);

temp1 = temp1->next;

temp2 = temp2->next;

}

}

return L;

1. 设计算法判断带头结点的循环双链表是否对称

Node\* front = head;

Node\* back = head->pre;

while (front != back && front->pre != back) {

if (front->data != back->data)

return false;

front = front->next;

back = back->pre;

}

return true;

1. 两个整数序列A = a1,a2,a3,...,am和B = b1,b2,b3,...,bn分别存储再两个单链表中，设计一个算法，判断序列B是否是序列A的连续子序列。

int lenA = A.len;

int lenB = B.len;

int posA = 0;

int posB = 0;

Node\* testA = A.head;

Node\* tempA = A.head;

Node\* tempB = B.head;

while (posA <= (lenA = lenB)) {

if (testA->data != B.head->data) {

testA = testA->next;

posA++;

}

else {

tempA = testA;

tempB = B.head;

posB = 0;

while (posB < lenB) {

if (tempA->data == tempB->data) {

tempA = tempA->next;

tempB = tempB->next;

posB++;

if (posB == (lenB - 1))

return true;

}

else {

break;

}

}//while

}//else

}//while

return false;

13. 编写程序用顺序表求解约瑟夫问题。

int to\_solve\_joseph()

{

int n; //人数

int q; //报数到q死

int result = 0;

SeqList joseph(1000);

joseph\_Seq(&joseph, (n - 1), q);

result = joseph.getData(n - 1);

return result;

}

void joseph\_Seq(SeqList& J, int n, int q)

{

if (n == 1){

J.insert(1, 0); //第[0]位为1人时结果

}

else {

int temp;

joseph\_Seq(J, (n - 1), q);

temp = J.getData(n - 2);

temp = (temp + q) % n;

J.insert(temp, (n - 1)); //第[n - 1]位为n人时结果

}

}

14. 假设有两个按元素值递增次序排列的线性表，均以单链表形式存储。请编写程序将这两个单链表归并为一个按元素值递减次序排列的单链表，并要求利用原来两个单链表的结点存放归并后的单链表。

Node\* pa, \* pb;

pa = La->next;

pb = Lb->next;

La->next = NULL;

Lb->next = NULL;

Node\* ra, \* rb;

while (pa && pb)

{

if (pa->data < pb->data)

{

ra = pa->next;

pa->next = La->next;

La->next = pa;

pa = ra;

}

else

{

rb = pb->next;

pb->next = La->next;

La->next = pb;

pb = rb;

}

}

while (pa)

{

ra = pa->next;

pa->next = La->next;

La->next = pa;

pa = ra;

}

while (pb)

{

rb = pb->next;

pb->next = La->next;

La->next = pb;

pb = rb;

}

第三章：

Queue（设定每个队列元素的数据类型为Type）：

int isQueueEmpty(Queue &Q); //判断队列是否空，1为空，0为不空。

int getFront(Queue &Q, Type &x); //通过x返回队头元素的值。

void enQueue(Queue &Q, Type x); //将新元素x插入到队列的队尾。

void deQueue(Queue &Q); //从队列中退出队头元素。

Stack（设定每个栈元素的数据类型与队列相同，为Type）:

void initStack(Stack &S); //对新创建的栈初始化，置成空栈。

int isStackEmpty(Stack &S); //判断栈空否，1为栈空，0为不空。

void push(Stack &S, Type x); //将新元素x进栈。

void pop(Stack &S); //栈顶元素出栈。

int getTop(Stack &S, Type &x); //通过x返回栈顶元素的值。

可以利用以上栈和队列的操作完成算法设计题

1. 有五个元素，其入栈次序是ABCDE,在各种可能的出栈次序中，第一个出栈的元素为C且第二个出栈元素为D的出栈次序有哪几个？请分别写出。

3 : CDEBA CDBEA CDBAE

2. 如果元素进栈序列为ABCDE，运用栈操作，能否得到出栈序列BCAED和DBACE?并解释原因。

BCAED可以；DBACE不可，栈后进先出，D后必为C或E，无法为B

3. 利用两个栈s1,s2模拟一个队列时，如何用栈的运算来实现该队列的运算：

queue\_empty: 判定队列是否为空；

bool queue\_empty(Stack s1, Stack s2)

{

if (stack\_len(s1) == 0 && stack\_len(s2) == 0)

return true;

else

return false;

}

enqueue: 插入一个元素；

void enqueue(Stack& s1,Stack &s2, int x)

{

push(s1, x);

}

dequeue: 删除一个元素；

void dequeue(Stack& s1, Stack& s2, int x)

{

pop(s1, x);

}

4. 设有两个栈S1，S2都采用顺序栈的方式，并且共享一个存储区[0, ..., maxsize-1]，为了尽量利用空间，减少溢出的可能，可采用栈顶相向、迎面增长的存储方式。试设计S1、S2有关的入栈和出栈的操作算法。

bool Stack::push(int pos, int x) {

//pos = 1, s1; pos = -1, s2

if (top1 == (top2 - 1))

return false;

else {

if (pos == 1) {

data[++top1] = x;

}

else {

data[--top2] = x;

}

return true;

}

}

bool Stack::pop(int i) {

if ((top1 == 0 && i == 1) || (top2 == MAX\_SIZE && i == -1) {

return false;

}

if (i == 1) {

top1--;

}

else if (i == -1) {

top2++;

}

return true;

}

}

5. 设计算法，通过输入一个n，输出斐波那契数列中的第n个元素的数值 。（注意递归方法的运用）

int getn()const;

{

int a[n];

a[0] = 1;

a[1] = 1;

for (int i = 2; i < n; i++)

{

a[i] = a[i - 1] + a[i - 2];

}

return a[n];

}

6. 假设一个算术表达式中包含圆括弧、方括弧和花括弧三种类型的括弧，编写一个算法判断表达式中的括弧是否正确配对。

bool brackets\_match(string target) {

Stack brackets;

int len = target.size;

char temp;

char topOf;

bool flag = 1;

for (int i = 0; i < len; i++) {

char brk = target[i];

char temp;

if (brk == '('

|| brk == '['

|| brk == '{') {

push(&brackets, brk);

}//if

else if (brk == ')'

|| brk == ']'

|| brk == '}') {

temp = get\_top(brackets);

//匹配成功

if ((temp == '(' && brk == ')'

|| temp == '[' && brk == ']')

|| temp == '{' && brk == '}') {

pop(&brackets);

continue;

}

//未匹配

else {

flag = 0;

break;

}//else

}//else if

// 非括号

else

continue;

}//for

topOf = get\_top(brackets); //若栈空，返回0

if (flag == 0 || get\_top == 0)

return false;

else

return true;

}//brackets\_match

7. Q是一个队列，S是一个空栈，实现将队列中的元素逆置的算法。

void converse(Queue& Q, Stack& s1, int i)

{

for (int j = 0; j < Q.length; j++)

{

deQueue(Q,Q->j);

pop(s1, Q->j);

}

}

8. 如果允许在循环队列的两端都可以进行插入和删除操作。试写出该循环队列的类型定义，并给出从队头插入和从队尾删除的算法

struct cQueue

{

double\* data; //数组，顺序表队列

int front; //头

int rear; //尾

};

Status EnQueue\_F(cQueue& Q, double e)

{//头插

if ((Q.front + MAXSIZE - 1) % MAXSIZE == Q.rear)return ERROR;

Q.front = (Q.front + MAXSIZE - 1) % MAXSIZE;

Q.data[Q.front] = e;

return OK;

}

Status DeQueue\_R(TuQueue& Q, double& e)

{//尾删

if (Q.front == Q.rear)return ERROR;

e = Q.data[(Q.rear + MAXSIZE - 1) % MAXSIZE];

Q.rear = (Q.rear + MAXSIZE - 1) % MAXSIZE;

return OK;

}

9.如果希望循环队列中的元素都能得到利用，则需要设置一个标志域tag，并以tag的值为0或1来区分对头指针front和队尾指针rear相同时的队列状态是“空”还是“满”，试编写与此结构相对应的入队的出队算法。

void in\_queue(int tag, int x, Queue& Q)

{

if (isQueueEmpty(Q) == 1)

{

tag = 0; //队空

}

else if ((front == rear) && isQueueEmpty(Q) == 0)

{

tag = 1; //队满

}

if (tag == 1)

return;

else

enQueue(Q, x);

rear++;

}

}

void out\_queue(Queue& Q, int x)

{

if (isQueueEmpty(Q) == 1)

{

tag = 0; //队空

}

else if ((front == rear) && isQueueEmpty(Q) == 0)

{

tag = 1; //队满

}

if (isQueueEmpty(Q) == 1)

return；

else

{

deQueue(Q);

}//else

}//out\_queue10. McCathy函数定义如下：



1. 编写一个递归函数计算给定x的M(x)值。

int calculate \_recursive (int x) {

if (x <= 100) {

return calculate(x + 12);

}

else {

return (x - 10);

}

}

（2）编写一个非递归函数计算给定x的M(x)值。

int calculate\_not\_recursive(int x) {

while (x <= 100)

x += 12;

return (x - 10);

}

第四章：

1. 设s=’I AM A STUDENT’, t=’GOOD’, q=’WORKER’, 求Replace(s,’STUDENT’,q) 和

Concat(SubString(s,6,2), Concat(t,SubString(s,7,8)))。

Replace(s, ’STUDENT’, q) : ‘I AM A WORKER’

Concat(SubString(s, 6, 2) : ‘A ’

Concat(t, SubString(s, 7, 8))) : ‘GOOD STUDENT’

2、设主串 S=‘xxyxxxyxxxxyxyx’ ，模式串 T=‘xxyxy’。 请问：如何用最少的比较次数找到 T 在 S 中出现的位 置？相应的比较次数是多少？

void Next(char\* L, int\* next) {

int j;

int k;

next[0] = -1;

j = 0;

k = -1;

while (j < strlen(L) - 1)

{

if (k == -1 || L[k] == L[j])

{

j++;

k++;

next[j] = k;

}

else

k = next[k];

}

}

while (S[i] != '\0' && L[j] != '\0')

{

if (S[i] == L[j])

{

i++;

j++;

}

else

{

if (next[j] == -1)

{

i++;

j = 0;

}

else

j = next[j];

}

}

3、两个字符串S1和S2的长度分别为m和n。求这两个字符串最大共同子串算法的时间复杂度为T(m,n)。估算最优的T(m,n),并简要说明理由。

最优的T(m,n)是O(n)。串S2是串S1的⼦串,且在S1中的位置是1，且开始求 出最⼤公共⼦串的⻓度恰是串S2的⻓度。⾮特殊情况下,T(m,n) =O(m\*n)。

4、设字符串S=”aabaabaabaac”,P=”aabaac”。

（1） 给出S和P的next值和nextval值；

void Next(char\* L, int\* next) {

int j;

int k;

next[0] = -1;

j = 0;

k = -1;

while (j < strlen(L) - 1)

{

if (k == -1 || L[k] == L[j])

{

j++;

k++;

next[j] = k;

}

else

k = next[k];

}

}

（2） 若S做主串，P做模式串，试给出利用KMP算法的匹配过程。

while (S[i] != '\0' && L[j] != '\0')

{

if (S[i] == L[j])

{

i++;

j++;

}

else

{

if (next[j] == -1)

{

i++;

j = 0;

}

else

j = next[j];

}

}

5、设串采用静态数组存储结构，编写函数实现两个串的比较Compare(S, T)。要求比较结果有等于和不等于两种情况。

bool Compare(string S, string T)

{

if (S.length != T.length)

return false;

else {

for (int i = 0; i < S.length; i++)

{

if (S[i] != T[i])

return false;

}

}

return true;

}

6、设串采用静态数组存储结构，编写函数实现串的替换Replace(S, start, T, V)，即要求在主串S中，从位置start开始查找是否存在子串T，若主串S中存在子串T，则用子串V替换子串T，且函数返回1；若主串S中不存在子串T，则函数返回0。

bool Replace(HString& S, int start, HString T, HString V)

{

int pos = find(S, T);

if (pos == -1)

return 0;

else {

StrDelete(S, pos, T.length);

StrInsert(S, V);

return 1;

}

}