2.19 已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试写一高效的算法，删除表中所有值大于 mink 且小于 maxk 的元素(若表中存在这样的元素)，同时释放被删结点空间，并分析你的算法的时间复杂度(注意：mink 和 maxk 是给定的两个参变量，它们的值可以和表中的元素相同，也可以不同)。

```
Status ListDelete_L(LinkList &L,ElemType mink,ElemType maxk)
{
    LinkList p,q,prev=NULL;
    if(mink>maxk)return ERROR;
    p=L;
    prev=p;
    p=p->next;
    while(p&&p->data<maxk){
        if(p->data<=mink){
            prev=p;
            p=p->next;
        }
        else{
            prev->next=p->next;
            q=p;
            p=p->next;
            free(q);
        }
    }
    return OK;
}
```

3.22 如题 3.21 的假设条件，试写一个算法，对逆波兰式表示的表达式求值。

```
char CalVal_InverPoland(char Buffer[])
{
    Stack Opnd;
    InitStack(Opnd);
    int i=0;
    char c;
    ElemType e1, e2;
    while(Buffer[i]!='#')
    {
        if(!Is0perator(Buffer[i]))
        {
            Push(Opnd, Buffer[i]);
        )
        else
        {
            Pop(Opnd, e2);
            Pop(Opnd, e1);
            c=Cal(e1,Buffer[i], e2);
            Push(Opnd, c);
        }
        i++;
```

```
    }
    return c;
}

char Cal(char c1, char op, char c2)
{
    int x, x1, x2;
    char ch[10];
    ch[0]=c1;
    ch[1]='\0';
    xl=atoi(ch);
    ch[0]=c2;
    ch[1]='\0';
    x2=atoi(ch);
    switch(op)
    {
        case '+': x=x1+x2; break;
        case '-': x=x1-x2; break;
        case '*': x=x1*x2; break;
        case '/': x=x1/x2; break ;
        default: break;
    }
    itoa(x,ch,10);
    return ch[0];
}
```

4.23 假设以块链结构作串的存储结构。试编写判别给定串是否具有对称性的算法，并要求算法的时间复杂度为 O(*StrLength(S)*)。

```
Status LS_Symmetry(LString S){
    //借助栈完成以块链为存储结构的串的对称性判别,对称返回TRUE, 反之FALSE。
    IntStack(T);
    p = S.head;
    i = 0;
     for(j = 1;j <= S.curlen;j++){
        if(j <= S.curlen / 2) //前半部分入栈
            Push(T, p -> ch[i]);
        else if(j > (S.curlen + 1) / 2) {//与后一半出栈比较
            Pop(T, e);
            if(p -> ch[i] != e)
                return FALSE;
        }
        if(++i == CHUNKSIZE){//下一个块
            p = p->next;
            i = 0;
        }
     }
    return TRUE;
}//LS_Symmetry
```

5.36 试按教科书 5.5 节图 5.8 所示存储结构，编写按上题描述的格式输出广义表的递归算法。

```
void GList_PrintList(GList A)
{
    if(!A) printf("(O)"); //空表
    else if(!A->tag) printf("%d", A->atom); //原子
    else
    {
        printf("(");
        GList_PrintList(A->ptr.hp);
        if(A->ptr.tp)
        {
            printf(",");
            GList_PrintList(A->ptr.tp);
        } //只有当表尾非空时才需要打印逗号
    printf(")");
    }//else
} //GList_PrintList
```

6.53 试编写算法，求给定二叉树上从根结点到叶子结点的一条其路径长度等于树的深度减一的路径(即列出从根结点到该叶子结点的结点序列)。若这样的路径存在多条，则输出路径终点(叶子结点)在“最左”的一条。

```
int maxh;
Status Printpath_MaxdepthS1(Bitree T) //求深度等于树高度减一的最靠左的结点
{
    Bitree pathd;
    maxh=Get_Depth(T); //Get_Depth 函数见 6.44
    if(maxh<2) return ERROR; //无符合条件结点
    Find_h(T,1);
    return OK;
}//Printpath_MaxdepthS1

void Find_h(Bitree T,int h)//寻找深度为 maxh-1 的结点
{
    path[h]=T;
    if(h==maxh-1)
    {
        for(i=1;path;i++)
        printf("%c",path->data);
        exit; //打印输出路径
    }
    else
    {
        if(T->lchild) Find_h(T->lchild,h+1);
```

```
        if(T->rchild)Find_h(T->rchild,h+1);
    }
    path[h]=NULL; //回溯
}//Find_h
```

6.74 试写一递归算法，以 6.73 题给定的树的广义表表示法的字符序列形式输出以孩子-兄弟链表表示的树。

```
void PrintGlist_CSTree(CSTree T)
{
    printf("%c",T->data);
    if(T->firstchild) //非叶结点
    {
        printf("(");
        for(p=T->firstchild;p;p=p->nextsib)
        {
            PrintGlist_CSTree(p);
            if(p->nextsib) printf(","); //最后一个孩子后面不需要加逗号
        }
        printf(")");
    } //if
}//PrintGlist_CSTree
```

7.24 试利用栈的基本操作编写，按深度优先搜索策略遍历一个强连通图的非递归形式的算法。算法中不规定具体的存储结构，而将图 Graph 看成是一种抽象的数据类型。

```
void STraverse_Nonrecursive(Graph G)//非递归遍历强连通图 G
{
    int visited[MAXSIZE];
    InitStack(S);
    Push(S,GetVex(S,1)); //将第一个顶点入栈
    visit(1);
    visited=1;
    while(!StackEmpty(S))
    {
        while(Gettop(S,i)&&i)
        {
            j=FirstAdjVex(G,i);
            if( j&&lvisited[i])
            {
                visit(j);
                visited[i]=1;
                Push(S,j); //向左走到尽头
            }
        }//while
        if(!StackEmpty(S))
```

```
        {
            Pop(S,j);
            Gettop(S,i);
            k=NextAdjVex(G,i,j); //向右走一步
            if(k&&!visited[k])
            visit(k);
            visited[k]=1;
            Push(S,k);
        }//if
    }//while
}//Straverse_Nonrecursive
```

7.34 试编写一个算法，给有向无环图 G 中每个顶点赋以一个整数序号，并满足以下条件：若从顶点主至顶点 j 有一条弧，则应使 i<j。

```cpp
#include<iostream>
#include<cstdio>
#include<queue>
using namespace std;
struct edgenode {
    int adjvertex;
    edgenode* next;
};
struct vertnode {
    char vertex;
    edgenode* firstedge;
    int indegree;
    int number;
};
struct AMgraph {
    vertnode vertlist[100];
    int n, e;
};
```

```cpp
AMgraph* Creategraph() {
    AMgraph* g = new AMgraph;
    scanf("%d,%d", &g->n, &g->e);
    for (int i = 0; i < g->n; i++) {
        cin >> g->vertlist[i].vertex;
        g->vertlist[i].firstedge = new edgenode;
        g->vertlist[i].firstedge->next = NULL;
        g->vertlist[i].indegree = 0;
    }
    int adj1, adj2;
    edgenode* p, *temp;
    for (int i = 0; i < g->e; i++) {   //根据样例可知要采用尾插法
        scanf("%d,%d", &adj1, &adj2);
        p = new edgenode;
        p->adjvertex = adj2;
        p->next = NULL;
        temp = g->vertlist[adj1].firstedge;
        while (temp->next != NULL)  temp = temp->next;
        temp->next = p;
        g->vertlist[adj2].indegree++;
    }
    return g;
}
void Topologicalsort(AMgraph* g) {   //拓扑排序
    queue<int> q;
    int cnt = 0;
    for (int i = 0; i < g->n; i++)
        if (g->vertlist[i].indegree == 0)  q.push(i);
    int f, inq;
    edgenode* p;
    while (!q.empty()) {
        f = q.front();
        q.pop();
        g->vertlist[f].number = ++cnt;
        for (p = g->vertlist[f].firstedge->next; p; p = p->next) {
            inq = p->adjvertex;
            if ((--g->vertlist[inq].indegree) == 0)
                q.push(inq);
        }
    }
}
void Print(AMgraph* g) {
    for (int i = 0; i < g->n; i++)
        cout << g->vertlist[i].vertex << "," << g->vertlist[i].number <<
}
int main() {
    AMgraph* g = Creategraph();
    Topologicalsort(g);
    Print(g);
    return 0;
}
```