|  |
| --- |
| 数据结构实验指导  **实验一：线性表的操作**  实验一A：线性表的顺序存储结构操作  【题目】  设顺序表Va中的数据元素递增有序，试写一算法，将x插入到顺序表的适当位置上，以保证该表的有序性。  【实验目的】  掌握用动态数组实现线性表的顺序存储结构及顺序表的初始化、查找、插入操作。  【分析】  本程序设计方法：先根据插入结点x的值查找插入位置，然后从表尾到插入位置的元素依次往后移一步空出x元素的位置，送入x，顺序表长度增1。  【程序】  #include<stdio.h>  #include<stdlib.h>  #define list\_init\_size 10  #define increment 3  typedef struct {  int \*elem;  int length,listsize;  }sqlist; //类型定义  int \*p;  void initlist\_sq(sqlist &L) //初始化顺序表  {  L.elem=(int \*)malloc(list\_init\_size\*sizeof(int));  if(!L.elem) printf("overflow");  else {L.length=0;  L.listsize=list\_init\_size;  }    }    void output(sqlist L) //输出顺序表  { for(p=L.elem;p<=L.elem+L.length-1;p++)  printf(" %d",\*p);  }    void ex2\_11(sqlist &L,int x) //顺序表中插入x  { int \*q;  if (L.length>=L.listsize) //判表是否满  {L.elem=(int \*)realloc(L.elem,  (L.listsize+increment)\*sizeof(int));  L.listsize+=increment;  }  p=L.elem;  while (\*p<x && p<=L.elem+L.length-1) //查找插入位置  p++;  for(q=&(L.elem[L.length-1]);q>=p;q--) //元素后移空出插入位置  \*(q+1)=\*q;  \*p=x;  ++L.length;  }    void main()  {sqlist L;int i=0,x;  initlist\_sq(L);  printf("\n输入顺序表的长度：");  scanf("%d",&L.length);  p=L.elem;  printf("\n输入顺序表各元素：");  for(;i<L.length;i++)  scanf("%d",p++);  output(L);  printf("\n输入x=");  scanf("%d",&x);  ex2\_11(L,x);  printf("\n结果是：");  output(L);  printf("\n");  }  【运行结果】  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222124_460513461333.jpg  图1顺序表中插入x运行界面  实验一B：线性表的链式存储结构操作  【实验目的】  掌握用单链表的创建、打印、查找、插入、删除等基本操作和它们的综合操作。  【实验一B题目1】  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）同时释放被删除结点空间。  【分析】  算法先从首结点开始查找，找到大于mink的结点停下，再判断是否小于maxk，若是则删除，重复这样的判断，直至不小于maxk为止；若不是则结束。此算法由下面程序中的EX2\_19函数实现。  【实验一B题目2】  试写一算法，对单链表实现就地逆置。  【分析】  逆置的方法是改变每个结点中的指针，使其指向前驱，算法中用p当前结点，q指向p的前驱，r指向p的后继，p->next=q则当前结点的指针指向前驱了，然后三根指针同步往后移直至所有结点处理完。此算法由下面程序中的ex2\_22函数实现。  【实验一B题目3】  假设有两个按元素值递增有序排列的线性表A和B，均以单链表作存储结构，请编写算法将A和B表归并成一个按元素值递减有序（即非递增有序，允许表中含有值相同的元素）排列的线性表C，并要求利用表（即A表和B表）的结点空间构造C表。  【分析】  反复进行两表的比较，将元素值小的结点插入到 C表的首端，直至两表的元素都插入完。此算法由下面程序中的ex2\_24函数实现。  【程序】  #include<stdio.h>  #include<stdlib.h>  typedef struct node{  int data;  struct node \*next;  }lnode,\*linklist;  linklist creat() //创建带头单链表  {linklist p,head;int x;  head=(linklist)malloc(sizeof(lnode));  p=head;  scanf("%d",&x);  while (x>0)  {p->next=(linklist)malloc(sizeof(lnode));  p=p->next,p->data=x;scanf("%d",&x);  }  p->next=NULL;  return head;  }  void out(linklist head) //输出单链表  {linklist p;  for(p=head->next;p;p=p->next)  printf("%3d",p->data);  }  void ex2\_19(linklist l,int mink,int maxk) //删除mink~maxk函数   {linklist p,q;  if (mink<maxk)  {p=l;  while (p->next && p->next->data<=mink) //查找开始位置  p=p->next;  while (p->next && p->next->data<maxk) //判断、删除  {q=p->next;p->next=q->next;free(q);}  }  }  void ex2\_22(linklist L) //逆置单链表  {linklist p,q,r;  q=NULL;p=L->next;  while (p)  {r=p->next;  p->next=q;  q=p;p=r;  }  L->next=q;  }  linklist ex2\_24(linklist A,linklist B)              //两个递增有序链表合并成一个递减有序的单链表  {linklist C,p,q,r;  C=A;p=A->next;q=B->next;C->next=NULL;  while (p&&q)  {if (p->data==q->data)  {r=q;q=q->next;free(r);}  else {if(p->data<q->data)  { r=p;p=p->next;}  else {r=q;q=q->next;}  r->next=C->next;C->next=r;  }  }  while (p)  { r=p;p=p->next;  r->next=C->next;C->next=r;  }  while (q)  { r=q;q=q->next;  r->next=C->next;C->next=r;  }  return C;  }  void main()  {linklist HL,HL2;int n,m;  printf("\n创建2-19的单链表:\n输入链表的有序数据（用0表示输入结束）：");  HL=creat();  printf("2\_19的单链表为： ");  out(HL);  printf("\n输入 mink,maxk");  scanf("%d,%d",&n,&m);  ex2\_19(HL,n,m);  printf("2\_19的结果是：");  out(HL);  printf("\n运行2\_22:");  ex2\_22(HL);  printf("\n2\_22的结果是：");  out(HL);  printf("\n创建2\_24的单链表A：\n输入链表的有序数据（用0表示输入结束）：");  HL=creat();  printf("\n创建2\_24的单链表B：\n输入链表的有序数据（用0表示输入结束）：");  HL2=creat();  HL=ex2\_24(HL,HL2);  printf("2\_24的结果是：");  out(HL);  printf("\n");  }  【运行结果】  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222128_684202636761.jpg  图2实验一B单链表操作运行界面  **实验二 栈和队列的操作**  实验二A 链栈的操作  【题目】  将单链表作为栈，写出初始化栈、判空栈、进栈和出栈操作。  【实验目的】  掌握初始化栈、判空栈、进栈和出栈的基本操作的实现  【分析】  程序中栈顶指针采用两级指针实现操作的，这样不用返回栈顶指针了，且一个函数完成一个基本操作，所以用一个菜单来实现反复的操作。  【程序】  #include<stdio.h>  #include<malloc.h>  typedef struct node{  int value;  struct node \*next;  }stackNode,\*LinkStack;    void initStack(LinkStack &topPt) //初始化栈  {topPt=NULL;}    int isEmpty(LinkStack top) //判空栈  {return top==NULL;}    void push (LinkStack &topPt,int e) //进栈  {stackNode \*p;  p=(stackNode\*)malloc(sizeof(stackNode));  p->value=e;p->next=topPt;  topPt=p;  }    int pop(LinkStack &topPt,int &vPt) //出栈  { stackNode \*p;  if (isEmpty(topPt)) return 0;  p=topPt;vPt=p->value;  topPt=p->next;  free(p);return 1;  }    main()  {int choice=1,e,i;  LinkStack top,p;  initStack(top);  while (choice!=4)  {printf("\t1:push\n");  printf("\t2:pop\n");  printf("\t3:show the stack\n");  printf("\t4:exit\n");  scanf("%d",&choice);  if(choice==1) {printf("input an element(int):");  scanf("%d",&e);  push(top,e);  }  else if(choice==2){i=pop(top,e);  if(i==0) printf("Stack is empty\n" );  else printf("the poped value is:%d\n",e);  }  else if(choice==3){p=top;  printf("all element:");  while(p)  {printf("%3d",p->value);  p=p->next;  }  }  else if(choice!=4)printf("error\n");  }  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222135_770845795302.jpg  图3 实验二A栈操作运行界面  实验二B 循环队列的操作  【题目】  将顺序表作为队列，写出初始化、判空、入队和出队操作。  【实验目的】  掌握顺序表上实现循环队列方法，掌握初始化队列、入队列和出队列的操作实现，掌握队列满和空的条件。  【分析】  程序中用动态数组来实现循环队列，并通过结构体指针来操作循环队列。  【程序】  #include<stdio.h>  #include<malloc.h>  typedef struct{  int queueSize;  int \*q;  int head;  int tail;  }sQueue;    void initQueue(sQueue \*qPt,int size) //初始化循环队列  {qPt->queueSize=size;  qPt->q=(int\*)malloc(size\*sizeof(int));  qPt->head=qPt->tail=0;  }    int isEmpty(sQueue \*qPt) //判空  {return qPt->head==qPt->tail;}  int isFull(sQueue \*qPt) //判满  {return qPt->head==(qPt->tail+1)%qPt->queueSize;}    int enQ(sQueue \*qPt,int x) //进队  {if (isFull(qPt)) return 0;  qPt->q[qPt->tail]=x;  qPt->tail=(qPt->tail+1)%qPt->queueSize;  return 1;  }    int deQ(sQueue \*qPt,int \*vPt) //出队  {if(isEmpty(qPt)) return 0;  \*vPt=qPt->q[qPt->head];qPt->head=(qPt->head+1)%qPt->queueSize;  return 1;  }    main()  {sQueue sQ;int choice=1,x,i;  initQueue(&sQ,10);  while(choice!=4)  {printf("\t1:enter-sQueue\n");  printf("\t2:delete-sQueue\n");  printf("\t3:show the sQueue\n");  printf("\t4:end\n");  scanf("%d",&choice);  if(choice==1){printf("Enter an element(int):");  scanf("%d",&x);  i=enQ(&sQ,x);  if(i==0)printf("Queue is full\n");  }  else if (choice==2){i=deQ(&sQ,&x);  if (i==0)printf("Queue is empty\n");  else printf("The element of deQ is %d\n",x);  }  else if(choice==3){if(sQ.head==sQ.tail) printf("Queue is empty");  else  {printf("All element of Queue is");  i=sQ.head;  while (i!=sQ.tail)  {printf("%3d",sQ.q[i]);i=(i+1)%10;}  }  }  else if(choice!=4) printf("error\n");  }  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222144_786187901435.jpg  图4 实验二B循环队列操作的运行界面  **实验三 二叉树的操作**  【实验目的】  掌握二叉树上递归和非递归的前序、中序、后序遍历算法。  【实验三题目1】  编写递归算法，将二叉树中所有结点的左右子树相互交换。  【分析】  本题按前序遍历算法实现二叉树中所有结点的左右子树相互交换，即按前序的顺序判断结点的子树是否为空，若非空则左右子树相互交换。  【程序】  #include<stdio.h>  #include<malloc.h>  typedef struct binode{  int data;  struct binode \*lchild,\*rchild;  }binode,\*bitree;  typedef struct{  bitree elem[100];  int top;  }stack;    bitree creat\_bt(){ //按扩展前序建二叉树  bitree t;int x;  scanf("%d",&x);  if (x==0) t=NULL;  else { t=(bitree)malloc(sizeof(binode));  t->data=x;  t->lchild=creat\_bt();  t->rchild=creat\_bt();  }  return t;  }    void exchange(bitree t) //左、右子树交换  {bitree p;  if(t!=NULL)  { p=t->lchild;t->lchild=t->rchild;  t->rchild=p;  exchange(t->lchild);  exchange(t->rchild);  }  }    void inorder(bitree bt) //递归的中序遍历  { if (bt){  inorder(bt->lchild);  printf("% d",bt->data);  inorder(bt->rchild);  }  }    main()  {bitree root;  printf("\n");  printf("建二叉树，输入元素：");  root=creat\_bt(); /\*create tree of useing preorder\*/  printf("交换前的中序序列是：");  inorder(root);  exchange(root);  printf("\n交换后的中序序列是：");  inorder(root);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222150_979255397928.jpg  图5 二叉树左右子树交换运行界面  【实验三题目2】  编写非递归算法，求二叉树中的叶结点数。  【分析】  用栈来实现二叉树的非递归算法，本题用非递归的中序遍历算法求叶子。  【程序】  #include<stdio.h>  #include<malloc.h>  #define M 20  typedef struct binode{  int data;  struct binode \*lchild,\*rchild;  }binode,\*bitree;    void preorder(bitree t) //非递归的前序遍历  { int top=0;  bitree p,s[M];  p=t;  while (top>0||p)  {while(p!=NULL)  { printf("% d",p->data);  if(p->rchild!=NULL);  s[top++]=p->rchild;  p=p->lchild;  }  if(top>0) p=s[--top];  }  }    int inorderleaf(bitree t) // 用非递归的中序遍历求叶子数  { int top=0,n=0;  bitree p,s[M];  p=t;  while (p||top>0){  while(p!=NULL)  { s[top++]=p;  p=p->lchild;  }  if(top>0) {p=s[--top];  if( p->lchild==NULL && p->rchild==NULL ) n++;  p=p->rchild;}    }  return n;  }    bitree creat\_bt(){ //按扩展前序建二叉树  bitree t;int x;  scanf("%d",&x);  if (x==0) t=NULL;  else { t=(bitree)malloc(sizeof(binode));  t->data=x;  t->lchild=creat\_bt();  t->rchild=creat\_bt();  }  return t;  }    main()  {bitree root;  printf("\n");  printf("建树，输入元素： ");  root=creat\_bt();  printf("前序序列为：");  preorder(root);  printf("\n叶结点数=%d",inorderleaf(root));  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222157_958172242438.jpg  图6 求二叉树叶结点数运行界面  【实验三题目3】  编写按层次顺序（同一层自左至右）遍历二叉树的算法。  【分析】  用队列实现二叉树的层次遍历。  【程序】  #include <stdio.h>  #include<malloc.h>  #define maxn 20  typedef struct node {  char data;  struct node \*lchild;  struct node \*rchild;  }bnode,\*bitree;  void treelevel(bnode \*t) //层次遍历  {  bnode \*q[maxn],\*p;  int front,rear;  front=rear=0;  if (!t) return;  q[rear]=t; //树根进队  rear =(rear+1)%maxn;  while (front!=rear) //判队列空否  {  p=q[front]; //出队  front=(front+1)%maxn;  printf ("%c",p->data);// 输出  if (p->lchild) {//左子树不空  q[rear]=p->lchild;  rear=(rear+1)%maxn;  }  if (p->rchild) {//右子树不空  q[rear]=p->rchild;  rear=(rear+1)%maxn;  }  }  }  bnode \*creat\_bt(){  bnode \*t;char x;  scanf("%c",&x);  if (x==' ') t=NULL;  else { t=(bnode \*)malloc(sizeof(bnode));  t->data=x;  t->lchild=creat\_bt();  t->rchild=creat\_bt();  }  return t;  }  void preorder(bitree t)  {  if(t!=NULL)  { printf("%c",t->data);  preorder(t->lchild);  preorder(t->rchild);  }  }  main()  {bnode \*t;  printf("\t建树，输入元素(char)：");  t=creat\_bt();  printf("\t二叉树的前序为：");  preorder(t);  printf("\n\t二叉树的层次序为：");  treelevel(t);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222200_254902580485.jpg  图7 二叉树层次序遍历运行界面  **实验四 图的遍历操作**  【实验四题目1】  图在邻接矩阵存储结构上的深度优先遍历算法实现。  【实验目的】  掌握图的邻接矩阵存储算法、掌握深度优先遍历算法及其实现。  【分析】  首先键盘输入图的顶点和边信息存放邻接矩阵中，然后用递归的深度优先遍历算法进行遍历，在遍历过程中用全局变量visited数组标记访问过的顶点。  【程序】  #include<stdio.h>  #define Max 10  typedef enum{FALSE,TRUE} Boolean;//定义布尔类型  typedef struct{  char vexs[Max];  int edges[Max][Max];  int n,e;  }MGraph; //定义图的邻接矩阵存储类型  Boolean visited[Max];//定义全局变量  void CreateMGraph(MGraph \*G)//图用邻接矩阵存储  {int i,j,k;  printf("输入图的顶点数n，边数e：");  scanf("%d,%d",&G->n,&G->e);  for(i=0;i<G->n;i++) //邻接矩阵置0  for(j=0;j<G->n;j++)  G->edges[i][j]=0;  printf("输入各条边：\n");  for(k=0;k<G->e;k++)  { scanf(" %d,%d",&i,&j);//输入一条边  G->edges[i][j]=1;  G->edges[j][i]=1;  }  }  void outm(MGraph \*G)//输出邻接矩阵  {int i,j;  for(i=0;i<G->n;i++)  {for(j=0;j<G->n;j++)  printf("%3d",G->edges[i][j]);  printf("\n");}  }  void DFSM(MGraph \*G,int i) //深度优先遍历  {int j;  printf("%3d",i);  visited[i]=TRUE;  for(j=0;j<G->n;j++)  if (G->edges[i][j]==1 && !visited[j])  DFSM(G,j);  }  void DFSTraverse(MGraph \*G)  {int i;  for(i=0;i<G->n;i++)//初始化visited标记数组  visited[i]=FALSE;  for(i=0;i<G->n;i++)  if(!visited[i]) DFSM(G,i);//调用深度优先遍历函数  }  main()  {MGraph G;  printf("创建邻接矩阵\n");  CreateMGraph(&G);  printf("\n邻接矩阵为：\n");  outm(&G);  printf("\n深度优先遍历序列为：");  DFSTraverse(&G);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222209_481597309201.jpg  图8 邻接矩阵上的深度优先遍历运行界面  【实验四题目2】  图在邻接表存储结构上的广度优先遍历算法的实现。  【实验目的】  掌握创建邻接表，掌握广度优先遍历算法及实现。  【分析】  首先创建图的邻接表，然后用队列来实现图的广度优先遍历算法。  【程序】  #include<stdio.h>  #include<malloc.h>  #define Max 10  typedef enum{FALSE,TRUE} Boolean;  typedef struct node{  int adjvex;  struct node \*next;  }EdgeNode; //定义边类型  typedef struct vnode{  char vextex;  EdgeNode \*firstedge;  }VertexNode; //定义顶点类型  typedef struct{  VertexNode adjlist[Max];  int n,e;  }ALGraph;//定义图的邻接表类型  Boolean visited[Max];  void CreateALGraph(ALGraph \*G)//建邻接表  { EdgeNode \*s;  int i,j,k;  printf("输入图的顶点数n，边数e：");  scanf("%d,%d",&G->n,&G->e);  printf("输入各顶点值(char):");  for(i=0;i<G->n;i++) //建邻接表中的顶点表  {scanf(" %c",&G->adjlist[i].vextex);  G->adjlist[i].firstedge=NULL;  }  printf("输入各条边（序号）：\n");  for(k=0;k<G->e;k++) //建邻接表中的边表  {scanf("%d,%d",&i,&j);  s=(EdgeNode \*)malloc(sizeof(EdgeNode));  s->adjvex=j;  s->next=G->adjlist[i].firstedge;  G->adjlist[i].firstedge=s;  s=(EdgeNode \*)malloc(sizeof(EdgeNode));  s->adjvex=i;  s->next=G->adjlist[j].firstedge;  G->adjlist[j].firstedge=s;  }  }  void BFS(ALGraph \*G,int k) //广度优先遍历  { EdgeNode \*p;  int i,Q[Max],front,rear;  front=rear=0;  printf("%c",G->adjlist[k].vextex);  visited[k]=TRUE;  rear=(rear+1)%Max;  Q[rear]=k;  while (front!=rear)  {front=(front+1)%Max;  i=Q[front];  p=G->adjlist[i].firstedge;  while (p)  {if(!visited[p->adjvex])  {printf(" %c",G->adjlist[p->adjvex].vextex);  visited[p->adjvex]=TRUE;  rear=(rear+1)%Max;  Q[rear]=p->adjvex;  }  p=p->next;  }  }  }  void print(ALGraph G)//输出邻接表  {int i;EdgeNode \*p;  for(i=0;i<G.n;i++)  { printf("\n%c-->",G.adjlist[i].vextex);  p=G.adjlist[i].firstedge;  while (p!=NULL)  {printf("%3d",p->adjvex);  p=p->next;  }  // printf("\n");  }  }  main()  {ALGraph G;  int v;  printf("创建邻接表\n");  CreateALGraph(&G);  printf("邻接表为：");  print(G);  for(v=0;v<G.n;v++)  visited[v]=FALSE;  printf("\n广度优先遍历序列为：");  for(v=0;v<G.n;v++)  if(!visited[v]) BFS(&G,v);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222212_816749704832.jpg  图9邻接表上广度优先遍历运行界面  **实验五 树的动态查找**  【题目】  二叉排序树上查找结点x，若不存在，则插入；若存在，则删除。  【实验目的】  掌握二叉排序树的查找、插入和删除算法及其算法的实现。  【分析】  本题用非递归方法实现二叉排序树的动态查找的，首先从树根开始查找，找到后调用删除函数删除该结点；若找到不，则调用插入函数插入该结点。  【程序】  #include<stdio.h>  #include<malloc.h>  #define M 100  typedef struct binode{  int data;  struct binode \*lchild,\*rchild;  }binode,\*bitree;//定义二叉链表类型  void preorderf(bitree t)//前序遍历  { int top=0;  bitree p,s[M];  p=t;  do  {while(p!=NULL)  {printf("%d\t",p->data);  if(p->rchild!=NULL);  s[top++]=p->rchild;  p=p->lchild;  }  if(top>=0) p=s[--top];  }while (top>=0);  }  bitree ins(bitree t ,bitree f,int x)//二叉排序树中插入x  { bitree s;  s=(bitree)malloc(sizeof(binode));  s->data=x;s->lchild=s->rchild=NULL;  if (t==NULL) t=s;  else if (s->data<f->data) f->lchild=s;  else f->rchild=s;  return t;  }  bitree del(bitree t ,bitree p,bitree f) //删除二叉排序树中p指的结点  {bitree q,child; //t为根指针、p指被删结点、f是p的前驱指针  if (p->lchild&&p->rchild){//被删结点有左、右子树  q=f=p;p=p->rchild;  while (p->lchild!=NULL){//找p的中序后继  f=p;p=p->lchild;  }  q->data=p->data;//  }//p转化为只有一个子树了  if (p->lchild) child=p->lchild; //被删结点有左子树  else child=p->rchild; 被删结点有右子树  if (f==NULL)t=child;  else if (p==f->lchild)f->lchild=child;  else f->rchild=child;  free (p);  return t;  }  void main()  {bitree root=NULL,p,f;  int x;  printf("\n输入x值：");  scanf("%d",&x);  while (x!=0)  { p=root;f=NULL;  while(p &&p->data!=x)  { f=p;  if (x<p->data) p=p->lchild;  else p=p->rchild;  }  if (p==NULL) {root=ins(root,f,x);printf("插入完成");}  else {root=del(root,p,f);printf("删除完成");}  printf("\n继续输入x,直到输入0为止");  scanf("%d",&x);  }  printf("\n二叉排序树的前序为;");  preorderf(root);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222214_155867770692.jpg  图10 二叉排序树的动态查找运行界面  **实验六 排序算法的实现**  【实验六题目1】  链表直接插入排序的算法的实现。  【实验目的】  掌握插入排序的算法及在链式存储结构上的实现方法。  【分析】  本程序先建无序的单链表，然后从第二个结点（由r指针指向）开始的每个结点依次插入到前面有序链表中，有序链表的尾指针是q。  【程序】  #include<stdio.h>  #include<malloc.h>  typedef int elemtype;  typedef struct node{  int data;  struct node \*next;  }lnode,\*linklist;  linklist creat() //创建带头单链表  {linklist p,head;int x;  head=(linklist)malloc(sizeof(lnode));  p=head;  scanf("%d",&x);  while (x)  {p->next=(linklist)malloc(sizeof(lnode));  p=p->next,p->data=x;scanf("%d",&x);  }  p->next=NULL;  return head;  }  void insertsort(linklist head)//插入排序  {linklist p,q,r;  q=head->next; r=q->next; //q指向有序表的尾  while (r ) // r指向无序链表的首端  {p=head;  while ((p->next->data<=r->data) && p!=q)//查找插入位置  p=p->next;  if(p->next->data>r->data)  {q->next=r->next; r->next=p->next; p->next=r;}  else q=q->next;  r=q->next ;  }  }  output(linklist L)  {L=L->next;  while (L)  {printf("%3d",L->data);  L=L->next;}  }  main()  {linklist L;  printf("\n建链表，输入序列(0表示结束):");  L=creat();  insertsort(L);  printf("\n链式直接插入排序后的序列:");  output(L);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222215_482637286196.jpg  图11 链表插入排序的运行界面  【实验六题目2】  试以单链表为存储结构实现简单选择排序的算法。  【实验目的】  掌握选择排序的算法及在链式存储结构上的实现方法。  【分析】  在带头的无序链表中选出最小结点放到有序链表的尾端，尾端由指针p指向，最小结点由r指向。  【程序】  #include<stdio.h>  #include<malloc.h>  typedef int elemtype;  typedef struct node{  int data;  struct node \*next;  }lnode,\*linklist;  linklist creat()  {linklist p,head;int x;  head=(linklist)malloc(sizeof(lnode));  p=head;  scanf("%d",&x);  while (x)  {p->next=(linklist)malloc(sizeof(lnode));  p=p->next,p->data=x;scanf("%d",&x);  }  p->next=NULL;  return head;  }  void selectsort(linklist head) {//选择排序  linklist p,q,r,s;  p=head;  while (p->next){ //p链表中找出最小  q=p->next; r=p;  while (q->next!=NULL) // 找出最小值由r指向  {if (q->next->data<r->next->data) r=q;  q=q->next;  }  if (r!=p) {s=r->next; r->next=s->next;  s->next=p->next;p->next=s;}//最小结点插在p后  p=p->next;  }  }  output(linklist L)  {L=L->next;  while (L)  {printf("%3d",L->data);  L=L->next;}  }  main()  {linklist L;  printf("\n建链表，输入序列(直至输入0 结束):");  L=creat();  selectsort(L);  printf("\n链式选择排序后的序列:");  output(L);  printf("\n");  }  mhtml:file://C:\Users\Administrator\Documents\常用文档（1）\数据结构课件\数据结构.mht!http://202.120.223.158/download/20090109222216_830817582098.jpg  图12 链表选择排序的运行界面 |

