**实验1：线性表的存储**

**实验目的：**

1. **了解顺序表的存储结构，熟悉并掌握线性表的创建、插入，删除算法。**
2. **了解链表的存储结构，熟悉并掌握链表的创建和插入算法。**

**实验内容：**

**一、顺序表**

1. **首先创建一个顺序表：从键盘读入一组整数（长度小于等于20），按输入顺序放入顺序表，输入以－1结束（注意－1不放到顺序表内）；将创建好的顺序表元素依次输出到屏幕上。**
2. **编写插入函数，实现在已创建好的顺序表中插入一个元素，插入的元素值和插入位置从键盘读入，插入后将顺序表元素依次输出到屏幕上。**
3. **编写删除函数，实现在已创建好的顺序表中删除一个元素，欲删除的元素位置（序号）从键盘读入，删除后将顺序表元素依次输出到屏幕上。**
4. **编写主程序完成顺序表的创建、插入和删除操作。**

**创建顺序表的例程已经给出，供大家参考。插入、删除函数及主程序由大家自己完成**

#define MAXNUM 20

#define true 1

#define false 0

typedef struct

{

int data[MAXNUM];

int length;

}list**\_**type;

**提示：创建顺序表函数**

/\*create a list:input data from keyboard,end by -1\*/

void createlist(list\_type \*lp)

{

int i, elem;

lp->length=0;

printf("\nplease input datas of the list\n");

for(i=0; i< MAXNUM; i++)

{

scanf(" %d", &elem);

if(elem== -1) break;

lp->data[i]=elem;

lp->length++;

}

}

**提示：遍历元素并依次输出函数**

void showlist(list\_type \*lp)

{

int i;

printf("\nThese %d records are:\n", lp->length);

if(lp->length<=0)

{

printf("No data!\n");

return;

}

for(i=0; i<lp->length; i++)

printf(" %d ", lp->data[i]);

printf("\nlength of the list is:%d", lp->length);

}

#include<stdio.h>

#define MAXNUM 20

#define true 1

#define false 0

typedef struct

{

int data[MAXNUM];

int length;

}list\_type;

/\* 提示：创建顺序表函数 \*/

/\*create a list:input data from keyboard,end by -1\*/

void createlist(list\_type \*lp)

{

int i, elem;

lp->length=0;

printf("\nplease input datas of the list\n");

for(i=0; i< MAXNUM; i++)

{

scanf(" %d", &elem);

if(elem== -1) break;

lp->data[i]=elem;

lp->length++;

}

}

/\* 提示：遍历元素并依次输出函数 \*/

void showlist(list\_type \*lp)

{

int i;

printf("\nThese %d records are:\n", lp->length);

if(lp->length<=0)

{

printf("No data!\n");

return;

}

for(i=0; i<lp->length; i++)

printf(" %d ", lp->data[i]);

printf("\nlength of the list is:%d\n", lp->length);

}

void Insert(list\_type \*L, int x, int i)

{ int j;

if(L->length>=MAXNUM-1)

printf("overflow"); //如果表满了，则输出上溢异常

else if ((i<1)||(i>L->length+1))

printf("position is not correct"); //元素的插入位置不合理

else { for(j=L->length;j>=i;j--)

L->data[j+1]=L->data[j]; //元素后移

L->data[i]=x; //插入元素

L->length++; //表长度增加1

}

}

void Delete(list\_type \*L,int i)

{ int j; if((i<1)||(i>L->length))

printf(" position is not correct");

else { for(j=i+1;j<=L->length;j++)

L->data[j-1]=L->data[j]; //元素前移

L->length--; //表长度减1

}

}

int main(void)

{

list\_type l;

int i,x;

createlist(&l);

showlist(&l);

printf("输入要插入的元素下标\n");

scanf("%d",&i);

printf("输入要插入的元素\n");

scanf("%d",&x);

Insert(&l,x,i);

showlist(&l);

printf("输入要删除的元素下标\n");

scanf("%d",&i);

Delete(&l,i);

showlist(&l);

return 0;

}

**二、链表**

**1. 编写插入函数，实现在已创建好的链表中插入一个元素，插入元素值和插入位置从键盘读入，插入后将链表元素依次输出到屏幕上。**

**2. 编写主程序，调用插入函数实现链表的创建（5个结点）。并将创建好的链表显示在屏幕上。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<stdbool.h>

#define true 1

#define false 0

// 定义链表中的节点

typedef struct node

{

int member; // 节点中的成员

struct node \*pNext; // 指向下一个节点的指针

}Node,\*pNode;

// 函数声明

pNode CreateList(); // 创建链表函数

void TraverseList(pNode ); // 遍历链表函数

bool Insert\_Node(pNode ,int ,int); // 链表节点插入函数,第一个参数是头节点，第二个参数是要在第几个节点前插入，第三个参数是要插入的数据

int main()

{

pNode pHead = NULL; // 定义初始化头节点，等价于 struct Node \*pHead == NULL

int data; // 作为Insert\_Node函数的第三个参数

int num; // 作为Inset\_Node函数第二个参数

int choose;

int return\_val;

pHead = CreateList(); // 创建一个非循环单链表，并将该链表的头结点的地址付给pHead

printf("你输入的数据是：");

TraverseList(pHead); // 调用遍历链表函数

printf("是否还要进行如下操作：\n");

printf("1.插入数据 2.退出\n");

printf("请输入：");

scanf("%d",&choose);

while(choose==1)

{

switch (choose)

{

case 1:

{

printf("请输入要在第几个节点前插入数据：");

scanf("%d",&num);

printf("请输入要插入的数据：");

scanf("%d",&data);

if(Insert\_Node(pHead,num,data) == true)

{

printf("插入成功\n插入后的数据是：\n");

TraverseList(pHead);

}

else

{

printf("插入失败\n");

}

printf("操作完成后的数据是：");

TraverseList(pHead);

break;

}

case 2:

{

break;

}

}

printf("是否还要进行如下操作：\n");

printf("1.插入数据 2.退出\n");

printf("请输入：");

scanf("%d",&choose);

}

return 0;

}

// 创建链表函数

pNode CreateList()

{

int i; // 用于下面循环

int len; // 用来存放有效节点的字数

int val; // 用于临时存放用户输入的数据

pNode pHead = (pNode)malloc(sizeof(Node)); // 分配一个不存放有效数据的头结点

pNode pTail = pHead; // 链表的最后一个节点

pTail->pNext = NULL; // 最后一个节点的指针置为空

printf("请输入节点个数：");

scanf("%d",&len);

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

{

printf("第 %d 个节点的数值：",i+1);

scanf("%d",&val);

pNode pNew = (pNode)malloc(sizeof(Node)); // 为节点分配空间

pNew->member = val; //将用户输入的数据赋给节点的成员

pTail->pNext = pNew; //将最后一个节点的指针指向下一个新的节点

pNew->pNext = NULL; //将新节点中的指针置为空

pTail = pNew; //将新节点赋给最后的一个节点

}

return pHead; //返回头节点

}

// 遍历链表函数

void TraverseList(pNode pHead)

{

pNode p = pHead->pNext; //将头节点的指针给予临时节点p

while(NULL != p) //节点p不为空，循环

{

printf("%d ",p->member);

p = p->pNext;

}

printf("\n");

return ;

}

// 链表节点插入函数

// 第一个参数是头节点，第二个参数是要在第几个节点前插入，第三个参数是要插入的数据

bool Insert\_Node(pNode pHead, int front,int data)

{

int i = 0;

pNode \_node = pHead;

pNode pSwap; // 用于交换

if ((front < 1) && (NULL != \_node)) //判断用户输入的数据是否大于等于1,及\_node是否为空

{

return false;

}

while (i < front - 1) //通过循环使指针指向要插入哪个节点前的节点。说的自己都不懂了，还是看下面的图吧。

{

\_node = \_node->pNext;

++i;

}

pNode pNew = (pNode)malloc(sizeof(Node));

pNew->member = data; // 把输入的数据赋给要插入的节点

pSwap = \_node->pNext; // 把下一个节点的地址，给用于交换的pSwap

\_node->pNext = pNew; // 把要插入的节点的地址，给上个节点的指针域

pNew->pNext = pSwap; // 把插入节点的下一个节点的地址，给插入节点的指针域

return true;

}