实验二 线性链表的实现和应用

1、采用书上第 28 页定义的线性表链式存储结构，编程实现书中算法 2.8、算法 2.9、算法 2.10、算法 2.11，以及输出线性链表的算法。另外，编写主函数对所实现的算法进行测试。

```
#include <stdio.h>
_____
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef int ElemType;
typedef struct LNode
{     ElemType data;
      struct LNode *next;
}LNode,*LinkList;

//从表尾到表头逆向创建 n 个元素的单链表
void CreateList_L(LinkList &L, int n) {   // 算法 2.11
  // 逆位序输入 n 个元素的值，建立带表头结点的单链线性表 L
  _____
  _____
  L = (LinkList)malloc(sizeof(LNode));
  L->next = NULL;                        // 先建立一个带头结点的单链表
  for (i=n; i>0; --i) {
    p = (LinkList)malloc(sizeof(LNode));   // 生成新结点
    _____                 //输入元素值
    p->next = L->next;       L->next = p;       // 插入到表头
  }
} // CreateList_L

//输出单链表
Status OutputList_L(LinkList L)
{
      LinkList p=L->next;
      if (!p) return ERROR;
      while(_____)
      {     printf("%d ",p->data);
            _____
      }
      printf("\n");
      return OK;
}
```

```
Status GetElem_L(LinkList &L,int i, ElemType &e) {   // 算法 2.8   //取单链表位置 i 的元素值
  // L 为带头结点的单链表的头指针。
  // 当第 i 个元素存在时，其值赋给 e 并返回 OK，否则返回 ERROR
  ——————————
  ——————————
  p = L->next;   j = 1;                  // 初始化，p 指向第一个结点，j 为计数器
  while (p && j<i) {    // 顺指针向后查找，直到 p 指向第 i 个元素或 p 为空
    p = p->next;   ++j;
  }
  if ( !p || j>i ) return ERROR;   // 第 i 个元素不存在
  e = p->data;    // 取第 i 个元素
  return OK;
} // GetElem_L

//向单链表中插入一个元素
Status ListInsert_L(LinkList &L, int i, ElemType e) {   // 算法 2.9
  // 在带头结点的单链线性表 L 的第 i 个元素之前插入元素 e
  ——————————
  ——————————
  p = L;   j = 0;
  while (p && j < i-1) {   // 寻找第 i-1 个结点
    p = p->next;
    ++j;
  }
  if (!p || j > i-1) return ERROR;          // i 小于 1 或者大于表长
  s = (LinkList)malloc(sizeof(LNode));   // 生成新结点
  s->data = e;   s->next = p->next;          // 插入 L 中
  p->next = s;
  return OK;
} // ListInsert_L

//从单链表中删除一个元素
Status ListDelete_L(LinkList &L, int i, ElemType &e) {   // 算法 2.10
  // 在带头结点的单链线性表 L 中，删除第 i 个元素，并由 e 返回其值
  ——————————
  ——————————
  p = L;   j = 0;
  while (p->next && j < i-1) {   // 寻找第 i 个结点，并令 p 指向其前趋
    p = p->next;
    ++j;
  }
  if (!(p->next) || j > i-1) return ERROR;   // 删除位置不合理
  q = p->next;
  p->next = q->next;                  // 删除并释放结点
  e = q->data;
  free(q);
  return OK;
} // ListDelete_L
```

```
void main()
{
    ElemType b,d,dd;
    LinkList L;
    printf("创建单链表，输入 5 个元素：\n");
    CreateList_L(L,5);
    printf("输出单链表所有元素！ \n");
    OutputList_L(L);
    printf("输出单链表第 2 个位置元素到 dd！ \n");
    GetElem_L(L,2, dd);
    printf("dd=%d\n",dd);
    printf("插入元素 b: ");
    scanf("%d",&b);
    printf("在单链表第 4 个位置插入%d！ \n",b);
    ListInsert_L(L,4,b);
    printf("输出插入操作后单链表所有元素！ \n");
    OutputList_L(L);
    printf("删除单链表第 3 个位置的元素！ \n");
    ListDelete_L(L,3,d);
    printf("输出删除操作后单链表所有元素！ \n");
    OutputList_L(L);
}
```