> 一研为定，万山无阻

@[toc]

第四次 直播 单链表的头插与尾插

- 使用 `C++` 的引用进行读写数据

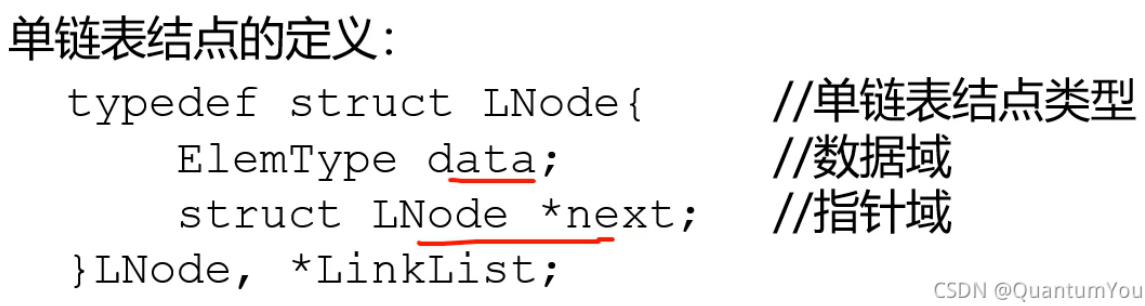
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
void modify_pointer(char*& p)
{
    p = (char*)malloc(100);//申请空间
    fgets(p, 100, stdin);//如果使用fgets传入的是一个指针变量，中间参数是指针指向的空间大小
}

int main()
{
    char* p;
    modify_pointer(p);
    puts(p);
    return 0;
}
```


顺序表的定义

顺序表

> 1、插入和删除操作移动大量元素。
> 2、数组的大小不好确定
> 3、占用一大段连续的存储空间，造成很多碎片。

单链表：逻辑上相邻的元素在物理上不相邻



`LinkList` 等价于 `struct LNode *`

- **头指针**：链表中第一个结点的存储位置，用来标识单链表。
- **头结点**：在单链表第一个结点之前附加的一个结点，为了操作上的方便

顺序表的增删改查

```
#include <stdio.h>
#include <stdlib.h>

#define MaxSize 50
typedef int ElemType;
```

```c
//静态分配
typedef struct{
    ElemType data[MaxSize];
    int length;//当前顺序表中有多少个元素
}SqList;
//动态分配
#define InitSize 100
typedef struct{
    ElemType *data;
    int capacity;//动态数组的最大容量
    int length;
}SeqList;
//i代表插入的位置，从1开始，e要插入的元素
bool ListInsert(SqList &L,int i,ElemType e)
{
    if(i<1||i>L.length+1)//判断要插入的位置是否合法
        return false;
    if(L.length>=MaxSize)//超出空间了
        return false;
    for(int j=L.length;j>=i;j--)//移动顺序表中的元素
        L.data[j]=L.data[j-1];
    L.data[i-1]=e;//数组下标从零开始，插入第一个位置，访问的下标为0
    L.length++;
    return true;
}
//删除使用元素e的引用的目的是拿出对应的值
bool ListDelete(SqList &L,int i,ElemType &e)
{
    if(i<1||i>L.length)//如果删除的位置是不合法
        return false;
    e=L.data[i-1];//获取顺序表中对应的元素，赋值给e
    for(int j=i;j<L.length;j++)
        L.data[j-1]=L.data[j];
    L.length--;//删除一个元素，顺序表长度减1
    return true;
}
//查找成功，返回位置，位置从1开始，查找失败，返回0
int LocateElem(SqList L,ElemType e)
{
    int i;
    for(i=0;i<L.length;i++)
        if(L.data[i]==e)
            return i+1;//加1就是元素在顺序表中的位置
    return 0;
}
//打印顺序表元素
void PrintList(SqList &L)
{
    for(int i=0;i<L.length;i++)
    {
        printf("%4d",L.data[i]);
    }
    printf("\n");
}
int main()
{
    SqList L;//顺序表的名称
    bool ret;//查看返回值，布尔型是True,或者False
```

```
    ElemType del;//要删除的元素
    //首先手动在顺序表中赋值
    L.data[0]=1;
    L.data[1]=2;
    L.data[2]=3;
    L.length=3;//总计三个元素
    ret=ListInsert(L,2,60);
    if(ret)
    {
        printf("插入成功\n");
        PrintList(L);
    }else{
        printf("插入失败\n");
    }
    ret=ListDelete(L,1,del);
    if(ret)
    {
        printf("删除成功\n");
        printf("删除元素值为 %d\n",del);
        PrintList(L);
    }else{
        printf("删除失败\n");
    }
    ret=LocateElem(L,60);
    if(ret)
    {
        printf("查找成功\n");
        printf("元素位置为 %d\n",ret);
    }else{
        printf("查找失败\n");
    }
    system("pause");//停在控制台窗口
}

```

头插法

- 使用头插法新建链表

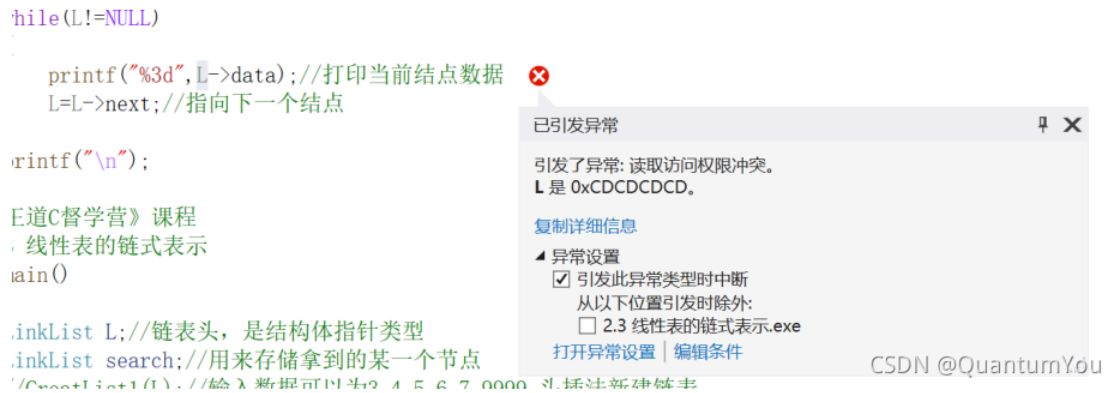
```

LinkList CreatList1(LinkList &L)//list_head_insert
{
    LNode *s;int x;
    L=(LinkList)malloc(sizeof(LNode));//带头结点的链表
    L->next=NULL;//L->data里边没放东西
    scanf("%d",&x);//从标准输入读取数据
    //3 4 5 6 7 9999
    while(x!=9999){
        s=(LNode*)malloc(sizeof(LNode));//申请一个新空间给s，强制类型转换
        s->data=x;//把读取到的值，给新空间中的data成员
        s->next=L->next;//让新结点的next指针指向链表的第一个元素（第一个放我们数据的元素）
        L->next=s;//让s作为第一个元素
        scanf("%d",&x);//读取标准输入
    }
    return L;
}
```

尾插法

- 使用尾插法尾插法新建链表

```
LinkList CreatList2(LinkList &L)//list_tail_insert
{
    int x;
    L=(LinkList)malloc(sizeof(LNode));//带头节点的链表
    LNode* s, * r = L;//LinkList s,r=L;也可以，r代表链表表尾结点，指向链表尾部
    //3 4 5 6 7 9999
    scanf("%d",&x);
    while(x!=9999){
        s=(LNode*)malloc(sizeof(LNode));
        s->data=x;
        r->next=s;//让尾部结点指向新结点
        r=s;//r指向新的表尾结点
        scanf("%d",&x);
    }
    r->next=NULL;//尾结点的next指针赋值为NULL
    return L;
}
```

Tips: `next`指针，没有赋值为NULL造成的



单链表的查找

- 关于 `q->next = p->next ;` 的理解， `->` 是指针访问成员变量（地址）`p->next`整体访问结构体空间里的一个成员

按序号查找

关键代码如下（伪代码）注意特殊情况（边界值的考虑）

```
LNode  *p = L->next ;
int j = 1 ;
while(P&&j<i){
   p=p->next ;
   j++ ;
}
return p ;
```

按值查找

关键代码如下（伪代码）注意特殊情况（边界值的考虑）

```
LNode  *p = L->next ;
while( P!= NULL && p->data != e){
    p=p->next ;
}
return p ;
```

单链表的操作

```
#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;
typedef struct LNode{
    ElemType data;
    struct LNode *next;//指向下一个结点
}LNode,*LinkList;
//头插法新建链表
LinkList CreatList1(LinkList &L)//list_head_insert
{
    LNode *s;int x;
    L=(LinkList)malloc(sizeof(LNode));//带头结点的链表
    L->next=NULL;//L->data里边没放东西
    scanf("%d",&x);//从标准输入读取数据
    //3 4 5 6 7 9999
    while(x!=9999){
        s=(LNode*)malloc(sizeof(LNode));//申请一个新空间给s，强制类型转换
        s->data=x;//把读取到的值，给新空间中的data成员
        s->next=L->next;//让新结点的next指针指向链表的第一个元素（第一个放我们数据的元素）
        L->next=s;//让s作为第一个元素
        scanf("%d",&x);//读取标准输入
    }
    return L;
}
//尾插法新建链表
LinkList CreatList2(LinkList &L)//list_tail_insert
{
    int x;
    L=(LinkList)malloc(sizeof(LNode));//带头节点的链表
    LNode* s, * r = L;//LinkList s,r=L;也可以，r代表链表表尾结点，指向链表尾部
    //3 4 5 6 7 9999
    scanf("%d",&x);
    while(x!=9999){
        s=(LNode*)malloc(sizeof(LNode));
        s->data=x;
        r->next=s;//让尾部结点指向新结点
        r=s;//r指向新的表尾结点
        scanf("%d",&x);
    }
    r->next=NULL;//尾结点的next指针赋值为NULL
    return L;
}
```

```
//按序号查找结点值
LNode *GetElem(LinkList L,int i)
{
    int j=1;
    LNode *p=L->next;
    if(i==0)
        return L;
    if(i<1)
        return NULL;
    while(p&&j<i)
    {
        p=p->next;
        j++;
    }
    return p;
}
//按值查找
LNode *LocateElem(LinkList L,ElemType e)
{
    LNode *p=L->next;
    while(p!=NULL&&p->data!=e)
        p=p->next;
    return p;
}
//新结点插入第i个位置
bool ListFrontInsert(LinkList L,int i,ElemType e)
{
    LinkList p=GetElem(L,i-1);
    if(NULL==p)
    {
        return false;
    }
    LinkList s=(LNode*)malloc(sizeof(LNode));//为新插入的结点申请空间
    s->data=e;
    s->next=p->next;
    p->next=s;
    return true;
}
//删除第i个结点
bool ListDelete(LinkList L,int i)
{
    LinkList p=GetElem(L,i-1);
    if(NULL==p)
    {
        return false;
    }
    LinkList q;
    q=p->next;
    p->next=q->next;//断链
    free(q);//释放对应结点的空间
    return true;
}
//打印链表中每个结点的值
void PrintList(LinkList L)
{
    L=L->next;
    while(L!=NULL)//NULL是为了代表一张空的藏宝图
    {
```

```
        printf("%3d",L->data);//打印当前结点数据
        L=L->next;//指向下一个结点
    }
    printf("\n");
}

//2.3 线性表的链式表示
int main()
{
    LinkList L;//链表头，是结构体指针类型
    LinkList search;//用来存储拿到的某一个节点
    //CreatList1(L);//输入数据可以为3 4 5 6 7 9999,头插法新建链表
    CreatList2(L);//输入数据可以为3 4 5 6 7 9999
    PrintList(L);//链表打印
    //search=GetElem(L,2);
    //if(search!=NULL)
    //{
    //  printf("按序号查找成功\n");
    //  printf("%d\n",search->data);
    //}
    //search=LocateElem(L,6);//按值查询
    //if(search!=NULL)
    //{
    //  printf("按值查找成功\n");
    //  printf("%d\n",search->data);
    //}
    //ListFrontInsert(L,2,99);//新结点插入第i个位置
    //PrintList(L);
    //ListDelete(L,4);//删除第4个结点
    //PrintList(L);
}
```