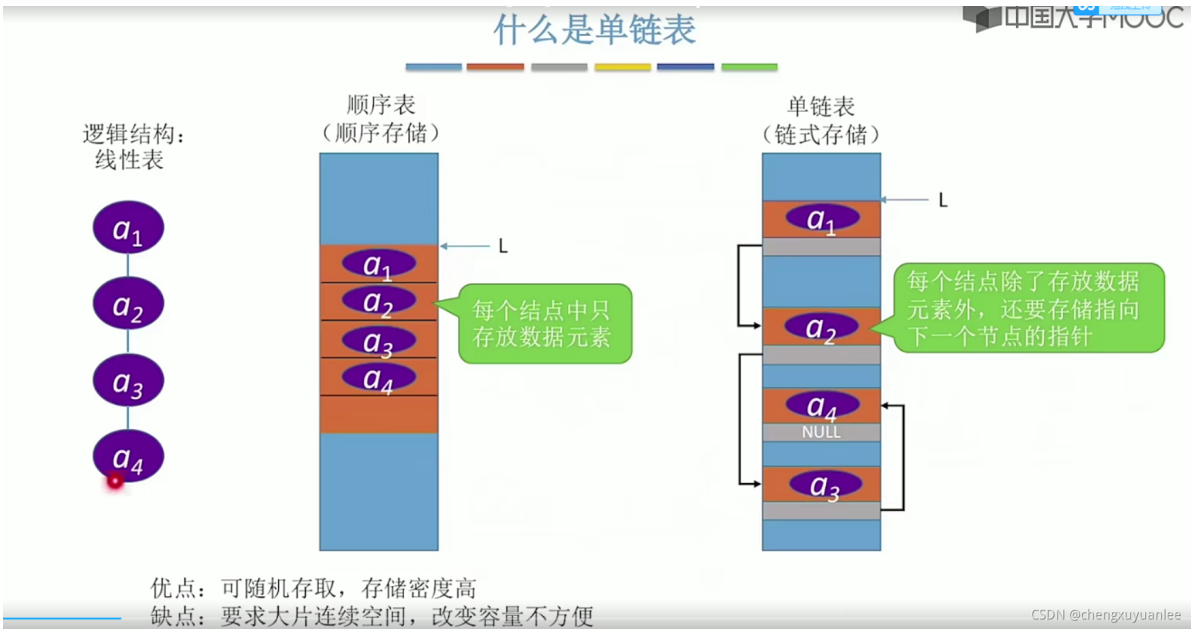
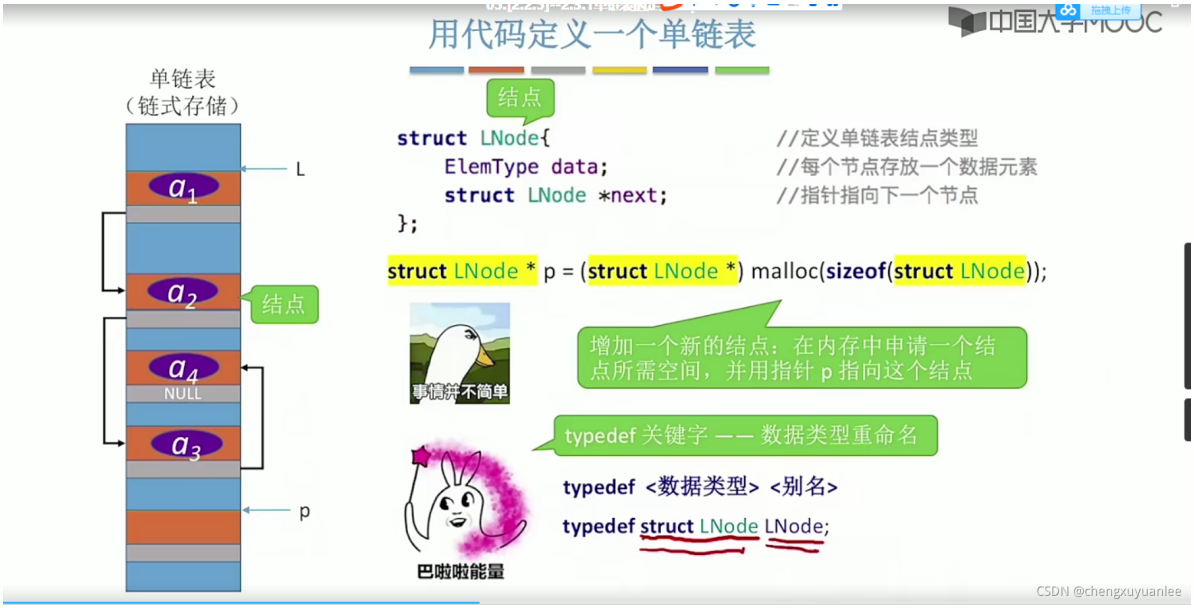
目录

作者为转行小白，刚接触数据结构算法，如有错误虚心接受批评指正。希望能够在计算机的学习过程中得到进步。

链表为线性表的一种，是一种链式存取的数据结构，用一组地址任意的存储单元存放线性表中的数据元素。地址空间是可以不连续的。将整体分为数据域和指针域两部分。



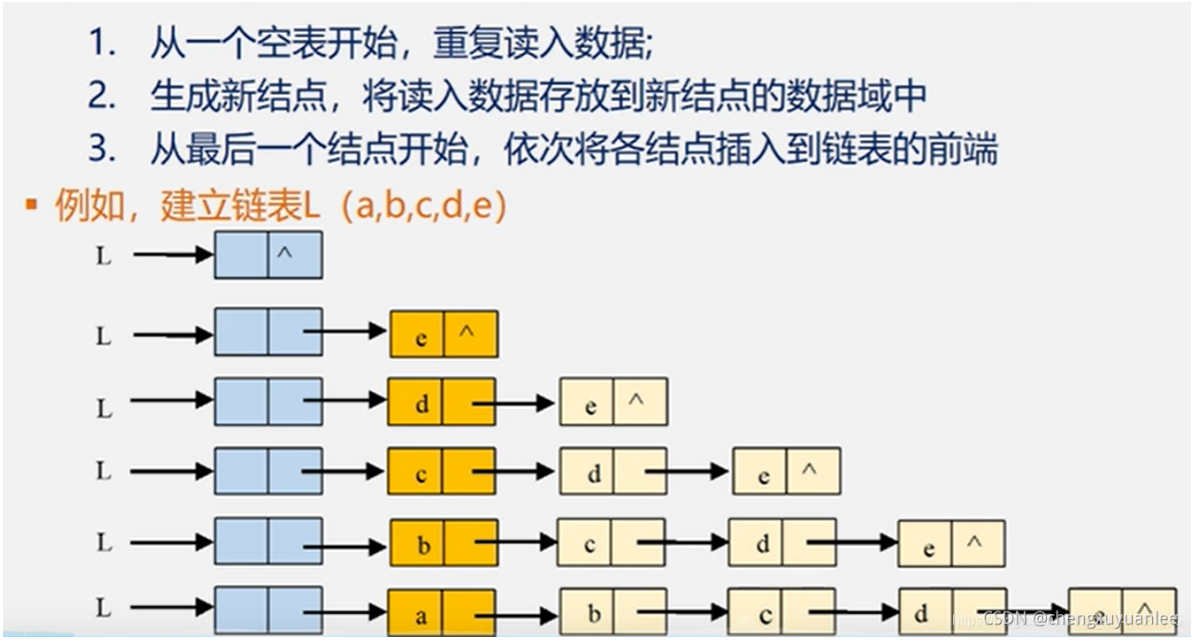


图片来自于 王道考研

头插法建立单链表

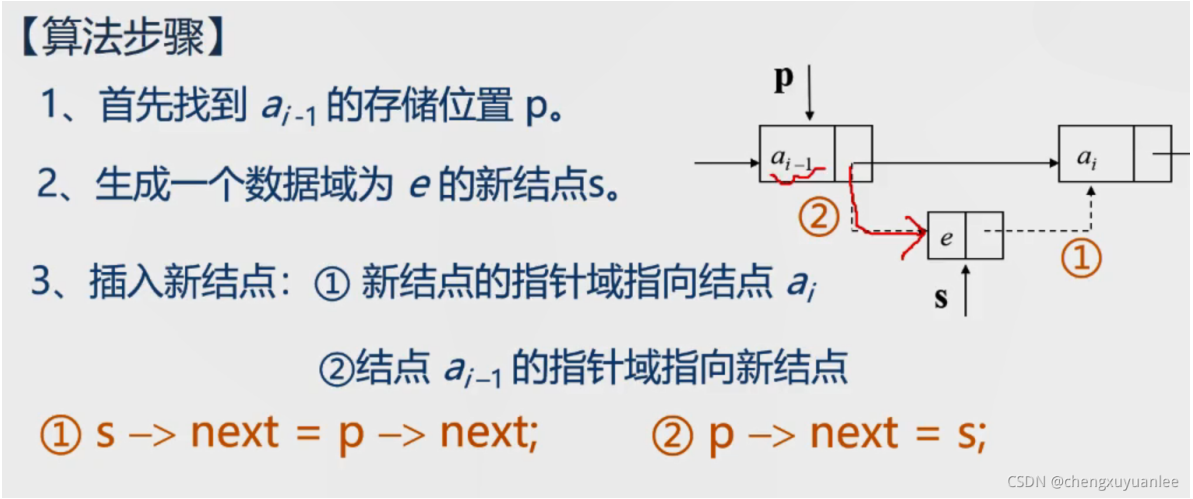
链表建立与插入类似，头插法即在头结点的后继插入，输出结果与输入结果相反，如下图想得到

a,b,c,d,,e应输入e，d，c，b，a



头插法核心代码

*



*

图片来自于 bilibili 青岛大学 王卓老师

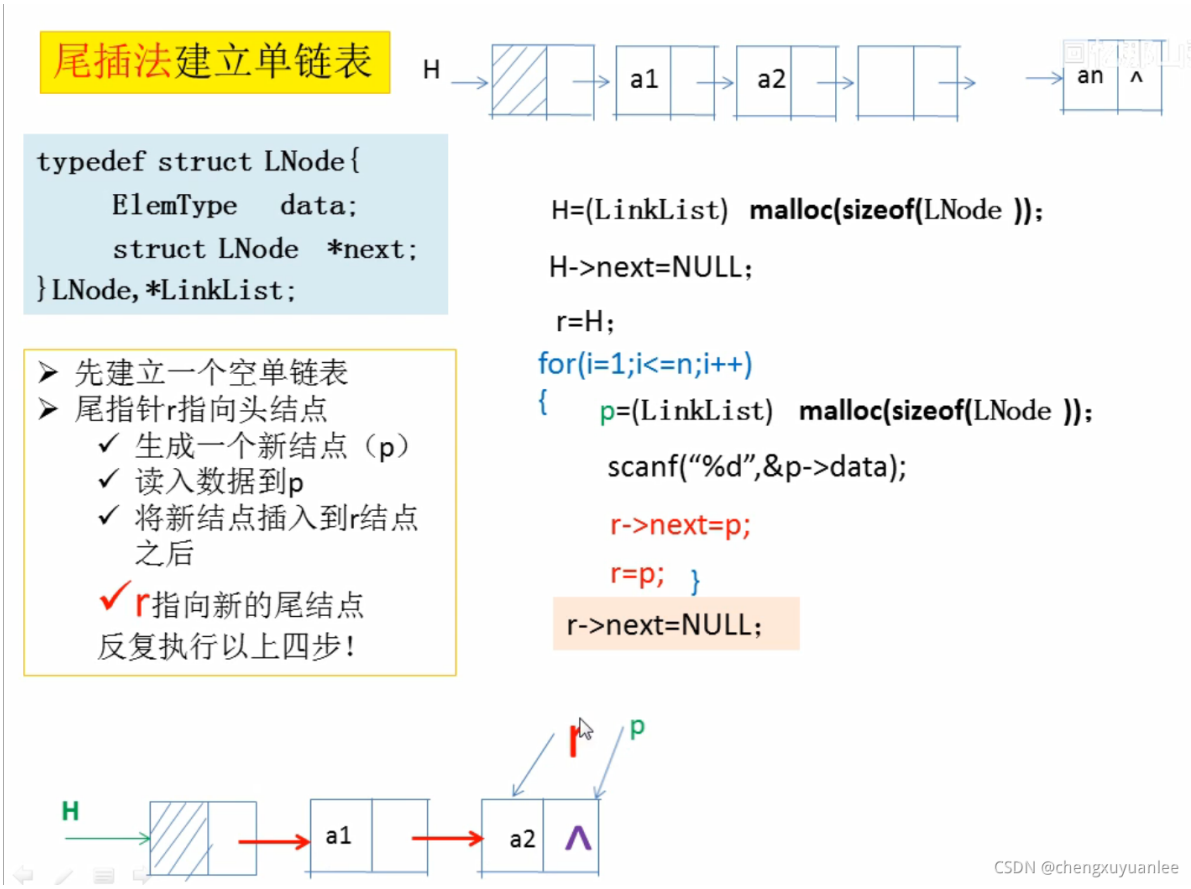
头插法代码(如有错误，请指正，作者转行小白)

```
#include <stdio.h>
#include <stdlib.h>
//头插法建立链表
typedef struct node
{
    int data;
    struct node* next;
}Node, * LinkList;//数据建立

LinkList create(LinkList L, int arr[], int  n)
{
    LinkList p;
    L = (LinkList)malloc(sizeof(Node));//为头节点开辟空间，并将头指针L指向头结点
    if (L == NULL) {                    //判断空间分配是否成功（不判断会报错）
        printf("内存分配不成功! \n");
    }
    else {
        L->next = NULL;                    //头结点指针域为空，空表判断
        for (int i = 0;i < 5;i++) {
            p = (LinkList)malloc(sizeof(Node));  //为普通结点开辟空间
            if (p == NULL) {
                printf("内存分配不成功! \n");//判断空间分配是否成功（不判断会报错）
            }
            else {
                p->data = arr[i];                  //为结点设置数据域
                p->next = L->next;                //第一次为尾结点  后面为当前节点的next域
指向后
                                                  //   一个节点
                L->next = p;                     //将当前节点与前一个节点相连
            }
        }
    }   //判断空间分配是否成功（不判断会报错）对应上面L开辟的if
    return L;  //返回头指针 L
}
```

```
void print(LinkList q){   //打印链表
    LinkList p;
    p = q->next;    //设置p为头结点
    while(p){                    // if（p）等价if（p! =NULL）
        printf("%d  ", p->data);
        p = p->next;           //跳向下一个结点
    }

}

int main() {
    LinkList L = 0;
    LinkList q = 0;
    int arr[5] = { 1,2,3,4,5 }; //链表成员
    int n = 0;   //链表总共n个
    q=create(L, arr, n);        //单链表建立
    print(q); //单链表打印/遍历
    return 0;
}
```

尾插法建立单链表

尾插法顾名思义为在尾部插入结点，与头插法不同在当同样输出a，b，c，d，e只需正序输入即可。在用尾插法时应注意创建尾指针与头指针共同指向头结点，再进行普通节点创建。



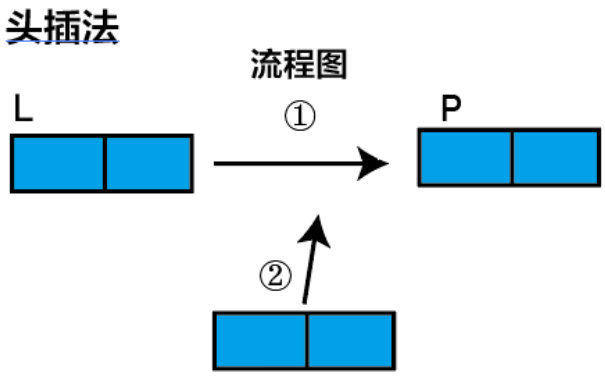
图片来自于 bilibili 回忆那山那水

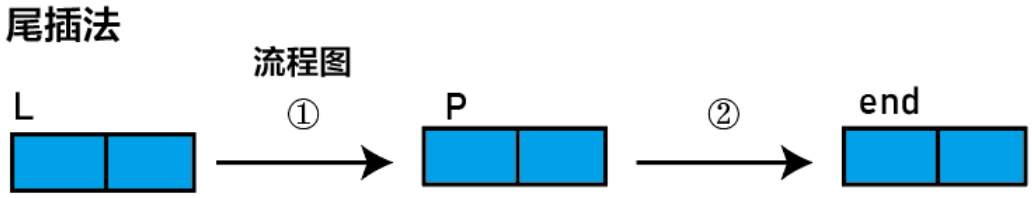
尾插法创建单链表代码如下：

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

```c
#include <stdlib.h>
#include <string.h>
#include <errno.h>
//头插法建立链表
typedef struct node {
    int data;
    struct node* next;
}node,*LinkList;

LinkList create(LinkList L,int arr[],int n) {
   L = (LinkList)malloc(sizeof(node)); //开辟空间并将头指针指向头结点
   if (L==NULL) {

       printf("内存分配不成功！\n");
   }
   else
   {
       L->next = NULL;
       LinkList end = L;//创建 尾结点并指向头结点
       LinkList p;
       for (int i = 0;i < 5;i++) {
       p= (LinkList)malloc(sizeof(node));
       if (p==NULL)
       {
           printf("内存分配不成功！\n");
       }
       else {

           p->data = arr[i];
           end->next = p;    //将头指针的next域指向p（连接前后结点）
           end = p;         //将p的数据域指针域赋给尾结点，之后p继续创造下一个结点
         }
       }
       p->next = NULL; //最后的结点的指向为NULL
                       //该处p是否可以用end
   }
   return L;
}
void print(LinkList q) {   //n为总个数
    LinkList p;
    p = q->next;    //设置p为头结点
    while (p) {                      // if（p）等价if（p！=NULL）
        printf("%d  ", p->data);
        p = p->next;           //跳向下一个结点
    }

}
int main() {
    LinkList L = 0;
    LinkList q = 0;
    int arr[5] = { 1,2,3,4,5 }; //链表成员
    int n = 0;   //链表总共n个
    q=create(L, arr, n);        //单链表建立
    print(q); //单链表打印/遍历
    return 0;
}
```

头插法 尾插法流程图对比





头插法 尾插法代码对比

```
//头插法核心代码段：
p->next = L->next;    //先将插入第n个节点与n+1连接
L->next = p;          //再将n-1与第n节点相连
```

顺序不可以颠倒，n-1节点先连接n则导致后面找不到n+1

```
//尾插法核心代码段：
 end->next = p;    //将头指针的next域指向p（连接前后结点）
 end = p;         //将p的数据域指针域赋给尾结点，之后p继续创造下一个结点
```

与头插法不同在于。尾插法是在上一结点尾部插入，
因此不需要进行第一步操作，即L—next=p—next;
直接进行头结点与当前相连即可，但end需指向p，p即可进行下一个节点的创建
最后要将尾结点next域置为空 即 p->next=NULL;

单链表的查询

创建函数在另一个文章，本段只截取查询函数部分

```
void FindList(LinkList L, int j,int e) {//查找数值为e在j位置
    LinkList p = L->next;j = 1;              //j从0或1开始取决于p从头结点还是头指针开始
    if (p == NULL) { printf("链表无元素"); } //判断链表是否包含元素
    else {
        while (p && p->data != e) {    //p不为空且数据不等于e
            p = p->next;                //指向下一个结点
```

```
            j++;                            //指向下一个结点
        }
        if (p == NULL && p->data != e) {         //经上次循环退出后有两个结果
            printf("该链表中不包含元素%d", e);     //走到尾结点未发现e
        }
        else { printf("找到了元素在第%d位置", j);    //找到了元素e
        }
    }
    free(p);

}
```

查找元素e并返回当前位置的 主函数

```
int main() {
    LinkList L = 0;
    LinkList q = 0;
    int arr[5] = { 1,2,3,4,5 }; //链表成员
    int  n= 0;   //链表总共n个
    int j = 0;
    q=create(L, arr, n); //单链表建立
    //查找第i项
    FindList(q,j,5);   //查找链表里是否有3，返回位置j
    return 0;
}
```

单链表的删除（按值查找）

此代码未考虑到具有相同数值的情况，后续会增补上，删除之前可进行判断列表是否为空，为空则不需要进行，直接返回。单链表的删除核心在于将前驱结点直接与当前结点的后继节点相连。同时最后要注意到free掉 要删除的空间。

1、遍历列表，判断条件：遇到相等则停止进行下一步判断

2、判断是否有和输入数值相等的情况，有则删除并释放空间，没有则打印“未发现”

```
LinkList DeleteList(LinkList q,int j) {       //删除等于j的值
    LinkList p=q;                           //设置p为头结点
    LinkList w= p->next;                    //设置w为p的前一个节点
    while(w&&w->data!=j){                 //w不为空且w的数据域等于j
        p = w;
        w = w->next;                        //此两步为将p和w跳向下两个结点
    }
    if (w->data==j) {
        p->next = w->next;  //前驱p指向w的后继结点
        free(w);            //释放空间
    }
    else {
        printf("列表找过未发现值%d", j);
    }
    return q;
}
```

单链表的删除（按序查找）

1、遍历列表以count计数

2、判断输入数值是否在链表范围内

3、如果在范围内，删除项释放空间。如果不在打印 “输入数值不对”，

```
LinkList DeleteList(LinkList q,int j) {
    LinkList t = q;                          //t为前驱，w为当前结点
    LinkList w = t->next; int count = 1;  //以count来计数
    if (w==NULL) {                          //判断头结点的next域是否为空
        printf("链表已经空了");
    }
    else {
        while (w && count<j) {         //w非空且当前结点小于输入数值
            t = w;
            w = w->next;              //后移
            count++;
        }

        if (w && count == j) {       //当前位置与输入数值相等则删除
            t->next = w->next;
            free(w);
        }
        else if(w==NULL) {            //当w为空时代表前驱的next域指向尾结点
            printf("输入值不正确");    //表示走遍列表并为找到数值
        }

    }
    return q;
}
```

单链表的插入（第j位插入某数值）

链表的插入结合了链表**头插法创建**与**删除**的操作进行的步骤如下：

1、创建头结点w，计数器count；

2、遍历找到第j位（判断），

3、创建空间->输入数据域->连接节点与后继节点->连接前驱结点与后继节点

```
LinkList AddList(LinkList q, int j,int e){ //在第j位插入e
    LinkList w = q->next;int count = 1;
    while (w&&count<j) {
        w = w->next;
        count++;
    }
    if (w&&count==j) {
        if (create==NULL) {
            return NULL;
        }
        LinkList create = (LinkList)malloc(sizeof(node));     //插入需要创建新的空间
        create->data = e;
        create->next = w->next;
        w->next = create;
```

```
    }
    else if (w == NULL) {
        printf("输入数据错误");
    }
    return q;
}
```

修改单链表值（第j位的值更改）

```
LinkList ChangeList(LinkList q, int j,int e){ //在第j位插入e
    LinkList Change = q->next;int count = 1;
    while (Change&&count<j) {
        Change = Change->next;
        count++;
    }
    if (Change && count == j) {
        Change->data = e;               //将data域数值改为输入数值
    }
    else if (Change == NULL){
        printf("位置不合理");
    }
    return q;
}
```