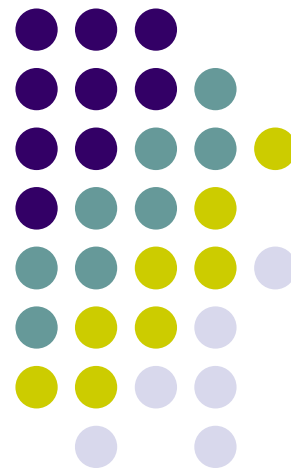
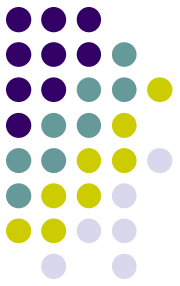


# 程序设计综合实验

武汉理工大学计算机科学与技术学院





# 实验四：计费管理系统的链表基本操作



# 本次实验授课内容

1. 动态内存分配与释放
2. 链表的概念及操作
  - 什么是链表
  - 建立简单的静态链表
  - 建立动态链表
  - 链表的基本操作：查找、插入
3. 实验四的内容与要求



# 什么是内存的动态分配

## 内存动态分配：

- 存放一些临时用的数据
  - 需要时随时开辟
  - 不需要时随时释放

## 为什么需要进行内存的动态分配？

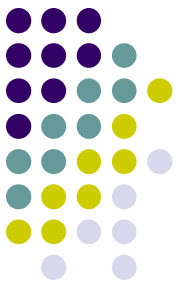
- 结构体数组的长度是固定的（静态存储分配）
- 有些应用中结构体变量的数量无法预知，只能在程序运行中动态地向系统申请内存空间（动态存储分配）



# 怎样建立内存的动态分配

## 对内存的动态分配

- 通过系统提供的库函数来实现（主要有**4个**函数）
  - **malloc, calloc, free, realloc**
  - 以上函数的声明在**stdlib.h**头文件中
    - 在用到这些函数时应当用“**#include <stdlib.h>**”指令把stdlib.h头文件包含到程序文件中。

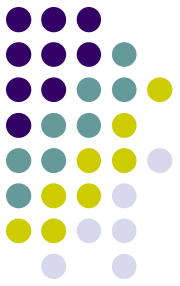


# malloc函数

- 函数原型:

**void \*malloc (unsigned int size);**

- 在内存的动态存储区中分配一个长度为size的连续空间
- 此函数的基类型为指针型函数void，即不指向任何类型的数据，只提供一个地址，返回的指针指向该分配域的开头位置，即函数的值在所分配区域的第一个字节的地址
- 不成功（例如内存空间不足），则返回空指针(**NULL**)



例如：

**malloc(100);**

- 开辟100字节的临时分配域
- 函数值为其第1个字节的地址

**malloc(sizeof(int));**

- 开辟长度为**sizeof(int)**的空间



## calloc函数

- 函数原型:

**void \*calloc**(unsigned int **n**, unsigned int **size**);

- 在内存的动态存储区中分配n个 长度为size的连续空间
  - 用**calloc**函数可以为一维数组开辟动态存储空间（动态数组），**n**为数组元素个数，每个元素长度为**size**
  - 函数返回指向所分配域的起始位置的指针
  - 如果分配不成功，返回**NULL**





例如：

```
p = calloc (50,4);
```

开辟**50 × 4**个字节的临时分配区域  
把起始地址赋给指针变量**p**

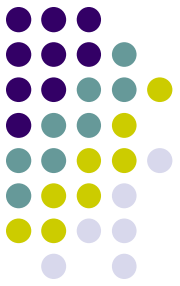


## free函数

- 其函数原型为

**void free(void \*p);**

- 其作用是释放指针变量 p 所指向的动态空间，使这部分空间能重新被其他变量使用
- p是最近一次调用calloc或malloc函数时得到的函数返回值
- free函数无返回值



例如：

**free(p);**

- 释放指针变量 **p** 所指向的已分配的动态空间

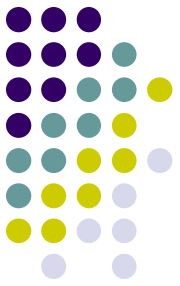


## realloc函数

- 其函数原型为

**void \*realloc** (**void \*p**, **unsigned int size**);

- 将p所指向的动态空间的大小改变为size。  
p的值不变
- 如果重分配不成功，返回**NULL**。
- 如果已经通过**malloc**函数或**calloc**函数获得了动态空间，想改变其大小，可以用**realloc**函数重新分配。



例如：

```
realloc (p,50);
```

将**p**所指向的已分配的动态空间改为**50**字节

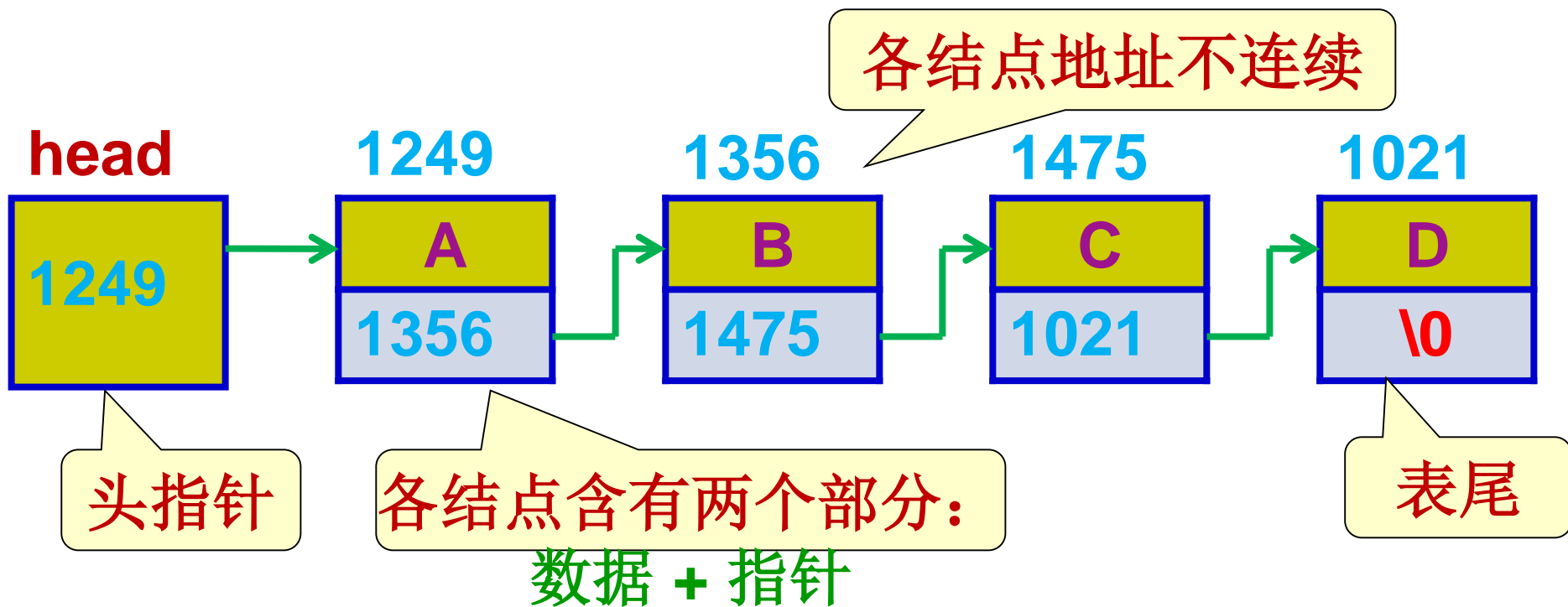
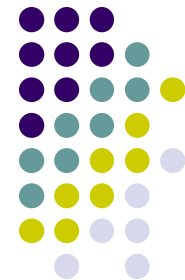


# 链表

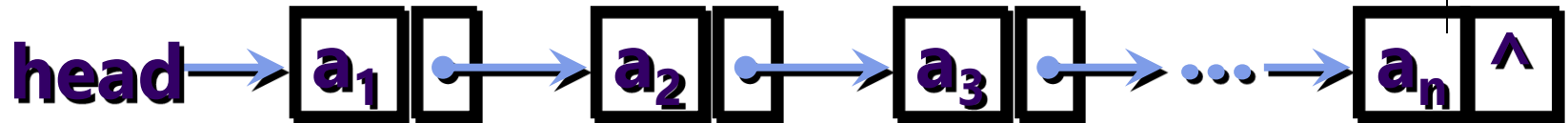
什么是链表：

- 链表是一种常见的重要的数据结构
- 若干个数据项（称为“结点”）按一定的原则连接起来
- 每个结点都是一个结构体变量：
  - 若干数据 + 指针（指向下一个结点）
- 链表必须利用指针变量才能实现
- 依靠这些指针将所有的结点连接成一个链表
- 它是动态地进行存储分配的一种结构

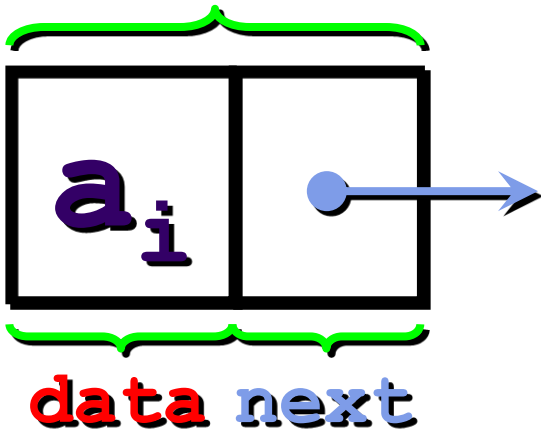
# 链表示例



# 链表示例



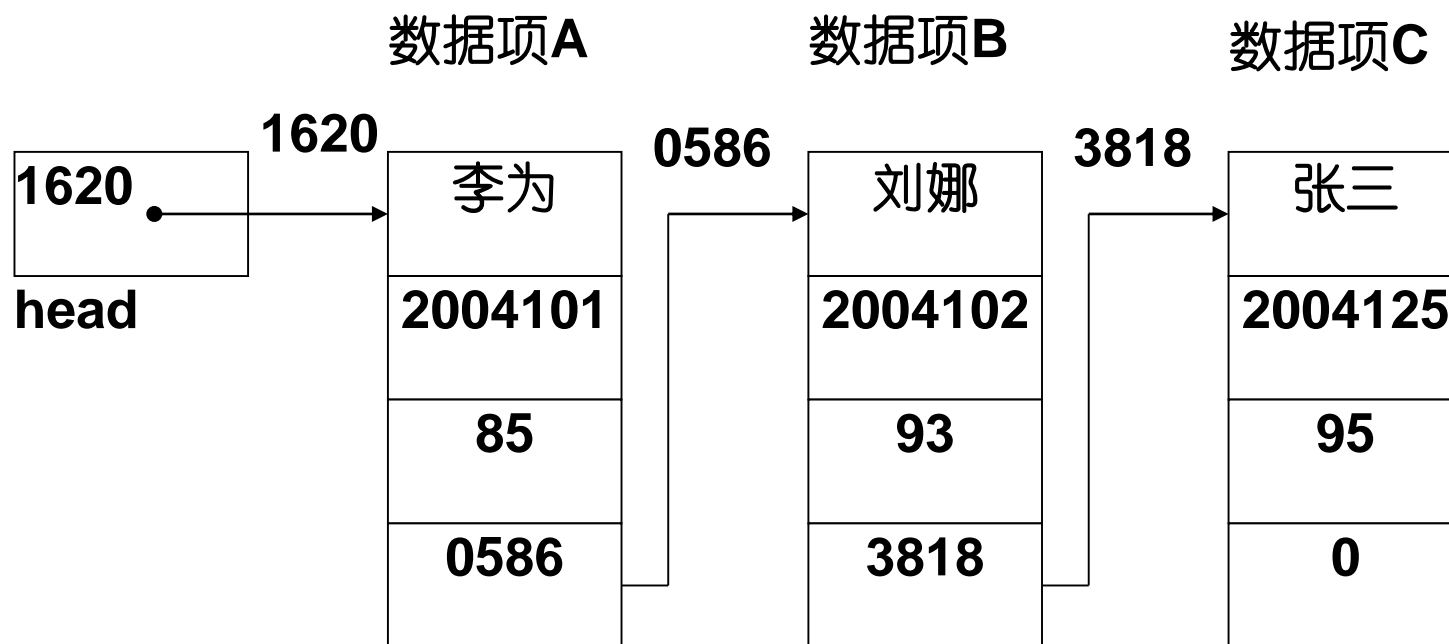
struct student



```
struct student {  
    char name[20];  
    long num;  
    float score;  
    struct student *next;  
};  
struct student *head;
```



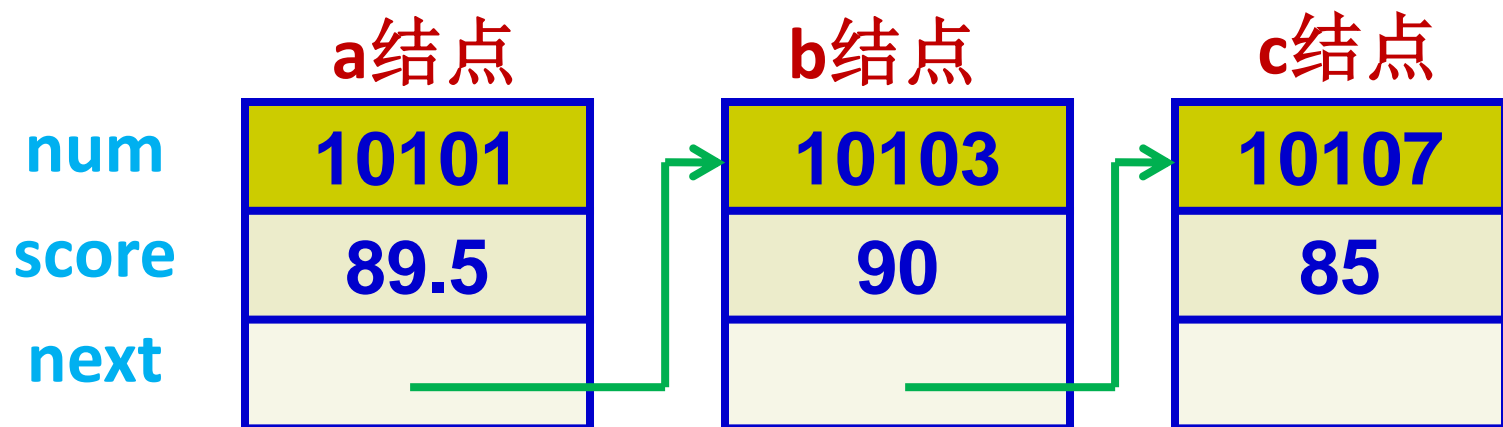
# 链表示例



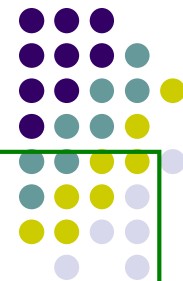


# 建立简单的静态链表

例： 建立一个如图所示的简单链表，它由3个学生数据的结点组成，要求输出各结点中的数据。



# 建立简单的静态链表



```
struct Student
{
    int num;
    float score;
    struct Student *next;
} *head, a, b, c;
```

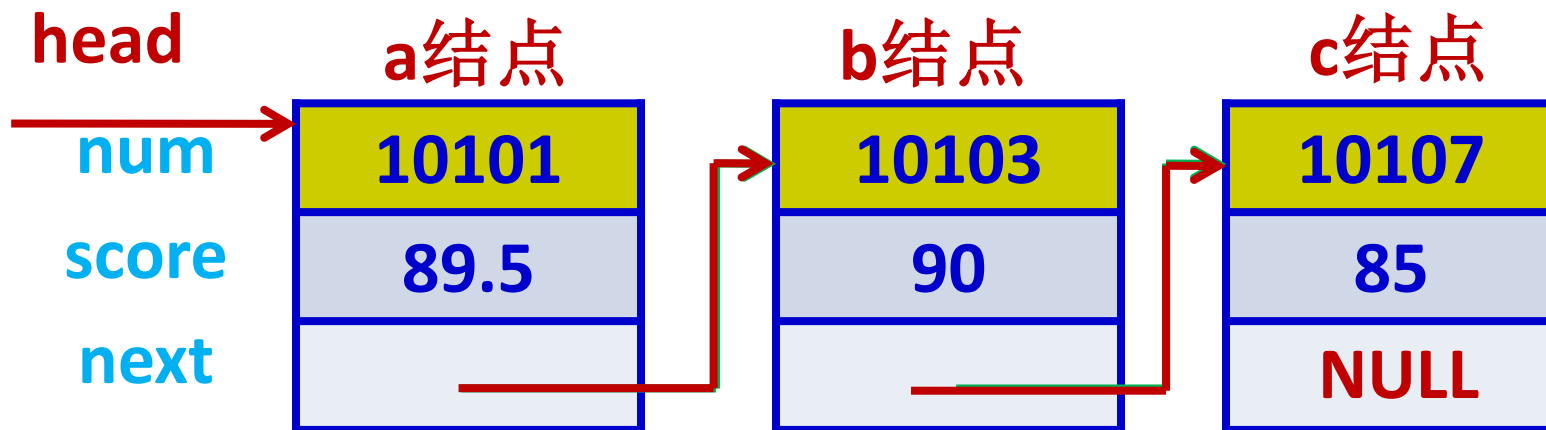
解题思路:

head=&a;

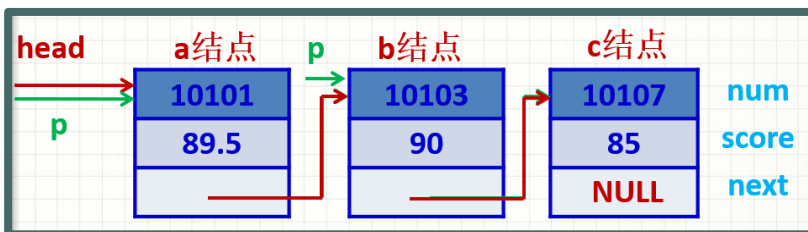
a.next=&b;

b.next=&c;

c.next=NULL;

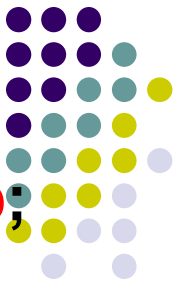


```
#include <stdio.h>
struct Student
{ int num;
  float score;
  struct Student *next;
};
```



```
10101 89.5
```

```
int main()
{ struct Student a,b,c,*head,*p;
  a.num=10101; a.score=89.5;
  b.num=10103; b.score=90;
  c.num=10107; c.score=85;
  head=&a;      a.next=&b;
  b.next=&c;      c.next=NULL;
  p=head;
  do
  {printf("%ld%5.1f\n",p->num, p->score);
   p=p->next;
  }while(p!=NULL);
  return 0;
}
```





# 建立动态链表

所谓建立动态链表是指:

- 在程序执行过程中从无到有地建立起一个链表
- 逐个地开辟结点和输入各结点数据，并建立起前后相链的关系

【示例】： 写一函数建立一个有n名学生数据的单向动态链表

【解题思路】：

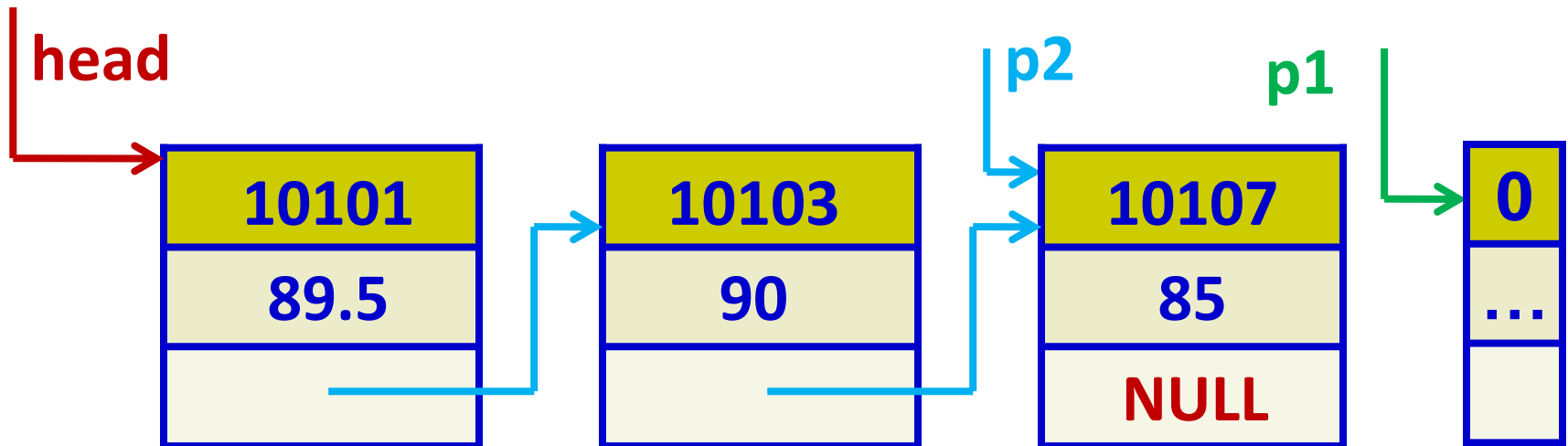
- 定义3个指针变量：**head**, **p1**和**p2**，它们都是用来指向**struct Student**类型数据  
`struct Student *head,*p1,*p2;`
- 用**malloc**函数开辟第一个结点，并使**p1**和**p2**指向它
- 读入一个学生的数据给**p1**所指的第一个结点
- 使**head**也指向新开辟的结点
- 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据
- 使第一个结点的**next**成员指向第二个结点，即连接第一个结点与第二个结点
- 使**p2**指向刚才建立的结点
- 再开辟另一个结点并使**p1**指向它。

。。  
**p1**总是开辟新结点  
**p2**总是指向最后结点  
用**p2**和**p1**连接两个结点



## 【解题思路】：

**p1**总是开辟**新**结点  
**p2**总是指向**最后**结点  
用**p2**和**p1**连接两个结点





**struct Student**类型数据的长度

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

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

```
#define LEN sizeof(struct Student)
```

```
struct Student
```

```
{ long num;
```

```
    float score;
```

```
    struct Student *next;
```

```
};
```

```
int n;
```

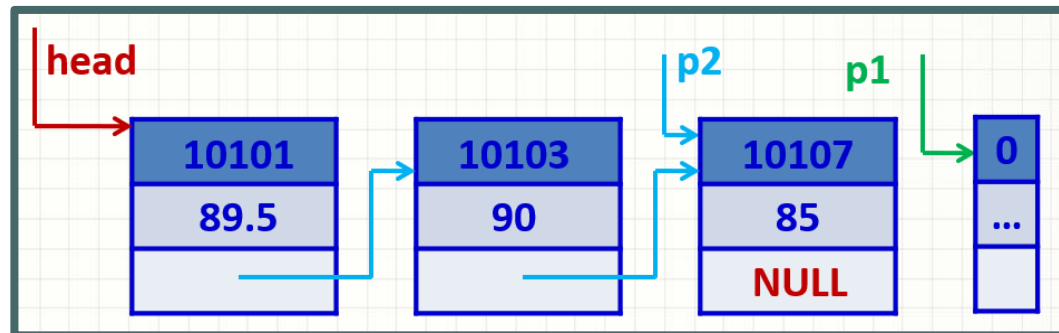




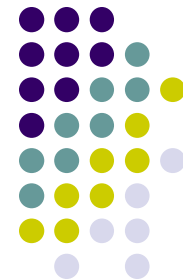
```
struct Student *creat(void)
{ struct Student *head,*p1,*p2; n=0;
  p1=p2=( struct Student*) malloc(LEN);

  scanf("%ld,%f",&p1->num,&p1->score);
  head=NULL;
  while(p1->num!=0)
  { n=n+1;
    if(n==1) head=p1;
    else p2->next=p1;
    p2=p1;
    p1=(struct Student*)malloc(LEN);
    scanf("%ld,%f",&p1->num,&p1->score);
  }
  p2->next=NULL;
  return(head);
}
```

**p1**总是开辟新结点  
**p2**总是指向最后结点  
用**p2**和**p1**连接两个结点



# 输出链表



【示例】 编写一个输出链表的函数print()

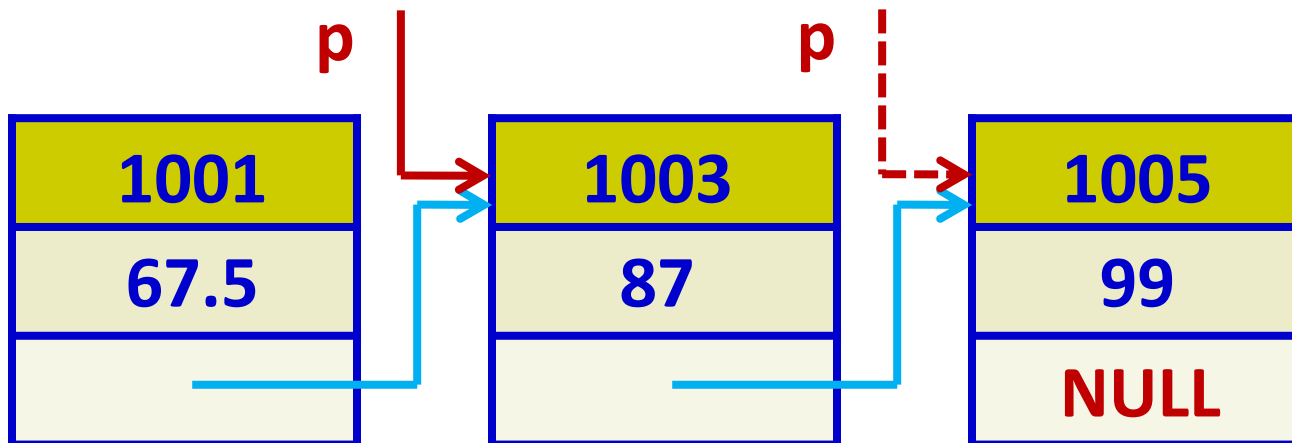
【解题思路】：


◆输出p所指的结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

◆使p后移一个结点

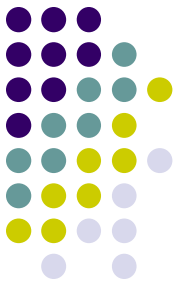
```
p=p->next;
```





1001	67.5
1003	87.0
1005	99.0

```
void print(struct Student *p)
{
    printf("\nThese %d records are:\n",n);
    if(p!=NULL)
        do
        { printf("%ld %5.1f\n",
                p->num,p->score);
          p=p->next;
        }while(p!=NULL);
}
```



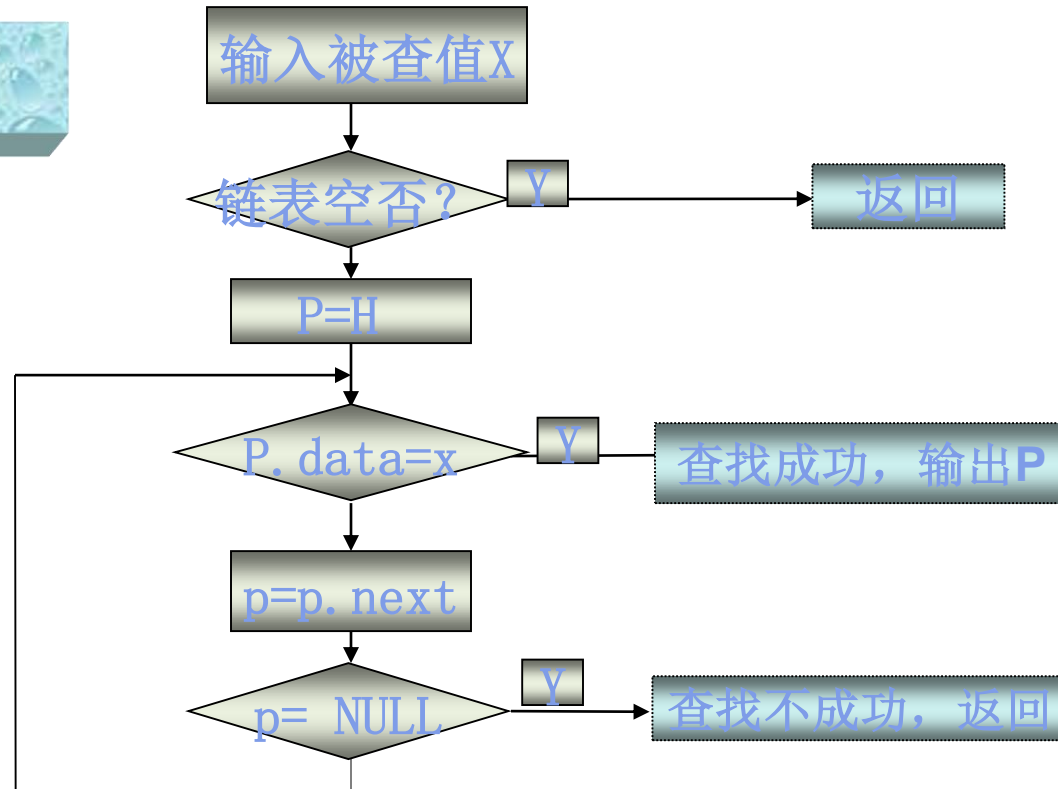
# 链表的查找操作

## 功能：

在链表中查找某结点，  
若存在则返回结点的地址，  
否则给出不存在的信息。



## 算法框图:



**方法:** 首先看链表空否?

不空则从表头结点的地址开始查找, 找到结束;  
找不到再取下一个结点的地址, 看其空否?  
若空则给出不存在的信息, 否则继续查找。



# 实验四：计费管理系统的链表基本操作

## 实验任务：

- 1) 动态内存分配与释放
- 2) 定义链表结构，建立链表
- 3) 进行链表的插入、删除、查询、输出等基本操作
- 4) 将卡信息保存在链表，

实现卡管理：**添加卡**、**查询卡**（链表）



# 实验4：实现卡管理：添加卡、查询卡（链表）

卡管理 (链表)	添加卡	定义链表结构。 将卡的信息保存到链表中。	1、链表 2、链表结点类型定义 3、添加链表结点 4、动态内存分配： malloc/free
	查询卡	根据用户输入的卡号， 在链表中查询卡号相同的卡信息，并以列表的格式显示在控制台中。	1、遍历链表 2、数组和指针作为函数参数

## 卡管理(链表)

### 添加卡(链表) ⊖

#### 使用链表实现添加卡信息



### 查询卡(链表) ⊕

#### 遍历链表，根据输入的卡号，查找符合条件的卡信息







## 实验4：实现卡管理：添加卡、查询卡（链表）

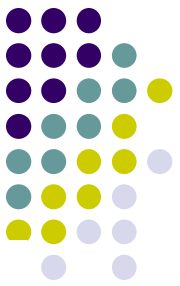
### 1、添加卡（链表）

上一次迭代：卡信息是保存在card\_service.c文件中的一个固定大小(50个元素)的结构体数组中。

数组在内存中是占据一片连续的存储单元，需要在内存中开辟一块很大的连续存储区域，来保存数组中的信息。如果添加卡信息不足50，就会造成了内存的浪费。而如果卡信息数量超过50之后，就不能再向数组中添加数据。

本次迭代：将采用一种动态的数据结构(链表)来保存卡信息。

根据添加的卡信息数量，动态分配内存，保存卡信息。使用链表实现添加卡功能。



# 添加卡

```
-----菜单-----
1.添加卡
2.查询卡
3.上机
4.下机
5.充值
6.退费
7.查询统计
8.注销卡
9.退出
请选择菜单项编号(0~8): 1
添加卡
请输入卡号<长度为1~18>: test
请输入密码<长度为1~8>: 123
请输入开卡金额(RMB): 50
-----添加的卡信息如下-----
卡号      密码      状态      开卡金额
test      123      0        50.0
-----菜单-----
1.添加卡
2.查询卡
3.上机
4.下机
5.充值
6.退费
7.查询统计
8.注销卡
9.退出
请选择的菜单项编号(0~8): 123456789123456789
输入的菜单编号错误!
```



## 实验4：实现卡管理：添加卡、查询卡（链表）

### 2、查询卡（链表）

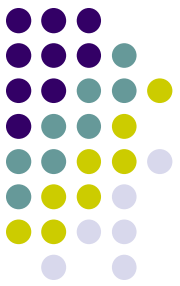
在“查询卡(结构体数组)”迭代中，实现了从结构体数组中查询卡号相同的卡信息。

上一次迭代“添加卡(链表)”中，修改了卡信息的保存方式，将添加的卡信息保存到card\_service.c文件中的卡信息链表cardList中。

本次迭代将实现两种方式的查找。

方式一：遍历卡信息链表cardList，从卡信息链表中查询卡号相同的卡信息，实现“查询卡”功能。

方式二：根据输入的卡号，采用模糊查询，从卡信息链表cardList中查询卡信息，并以列表的格式显示在界面上。



# 精确查询

```
-----菜单-----
1.添加卡
2.查询卡
3.上机
4.下机
5.充值
6.退费
7.查询统计
8.注销卡
9.退出
请选择菜单项编号(0~8): 2
-----查询卡-----
请输入查询的卡号<长度为1~18>: test
卡号      状态      余额      累计使用      使用次数      上次使用时间
test      0          50.0      50.0          0            2013-08-26 11:38
```



# 模糊查询

```
-----菜单-----
1. 添加卡
2. 查询卡
3. 上机
4. 下机
5. 充值
6. 退费
7. 查询统计
8. 注销卡
9. 退出
请选择菜单项编号 (0~8) : 2
-----查询卡-----
请输入查询的卡号<长度为1~18>:t
卡号      状态      余额      累计使用      使用次数      上次使用时间
test1     0          50.0      50.0          0            2013-10-25 19:01
test2     0          20.0      20.0          0            2013-10-25 19:01
```

# 实验四：计费管理系统的链表基本操作

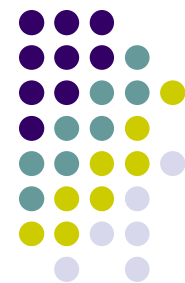


## 实现卡管理：添加卡、查询卡（链表）

（保留前面实验的代码，构造新函数编码实现）

	项目名称	实验内容	交付物
第4次	计费管理系统的链表基本操作	(1) 定义链表结构，建立链表。 (2) 进行链表的插入、删除和查询等基本操作。	运行 截图 + 代码 截图
第5次	计费管理系统的数据动态存储管理	(1) 定义链表结构，将卡的信息保存在链表中，然后写入卡信息文件。 (2) 查询卡时，先将卡信息文件中的卡信息保存到链表中，再根据用户输入的卡号，在链表中查询卡号相同的卡信息，并以列表的格式显示在控制台中。	

测试用例： 分别输入菜单编号1、 2； 进而  
添加卡（密码正常/超长/重复卡号）； 查询卡（存在/不存在）



# 实验四：计费管理系统的链表基本操作

## 测试用例：

### 一、输入菜单编号1，然后

1. 输入正常卡号、密码、金额
2. 输入超长卡号，正常密码、金额
3. 输入超长密码，正常卡号、金额
4. 输入非法金额，正常卡号、密码
5. 输入系统中已存在的卡号，正常密码、金额

### 二、输入菜单编号2，然后

1. 输入系统中存在的正常卡号（精确查询）
2. 输入系统中存在的正常卡号（模糊查询）
3. 输入超长卡号
4. 输入系统中不存在的正常卡号

验收要求：本次实验的正常运行需包含上述测试用例