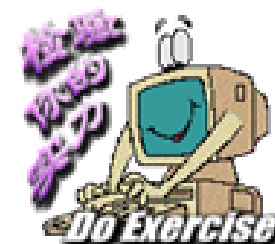
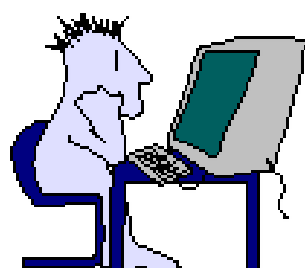


# C语言程序设计

主讲教师：秦琳琳

单位：信息学院自动化系

E-mail: [qinll@ustc.edu.cn](mailto:qinll@ustc.edu.cn)



# 第九章 结构体(6学时)

- ★ 结构体
- ★ 结构体数组
- ★ 指向结构体的指针
- ★ 动态存储分配函数
- ★ 结构体的应用—链表

## 9.1 结构体 (Structure)

- 结构体概述
- 结构体类型的声明
- 结构体变量的定义
- 结构体变量的初始化
- 结构体变量的引用

# 结构体概述

- 结构体
  - 将不同类型的数据组合成一个整体
  - 用来表示简单类型无法描述的复杂对象
  - 可以用结构体来定义用户自己的数据结构
- 举例
  - 描述学生(实体)信息(属性)

num	name	sex	age	score	addr
1301	Xu FeiFei	F	20	90.5	HeFei

# 结构体类型的声明

- 一般形式

- **struct** [结构体名] {  
    成员表列  
};

- “成员表列” 形式  
    类型名1 成员名1;  
    类型名2 成员名2;  
    ... ..

## 结构体类型的声明

- 类型定义只是为了说明一个实体相应的属性描述，只有通过定义相应的变量，并赋以一定的值才能构成一个实体的元素（记录）
- 结构变量的存储单元的大小为各成员所需容量的总合。
- 本例题，结构变量的存储单元的大小：  
 **$=4+20+1+4+4+32=65$**

# 结构体类型声明的说明 (1)

- 声明了一种**类型**，而不是定义**变量**
- **结构体名**可以**没有**，但是这样就**无法**再次使用该结构体类型了
- 成员表列中是成员 (Member) 的定义
- 成员的定义形式与变量定义相同
- **成员类型**可以是**另一结构体类型**，但**不可**直接或间接**递归嵌套**
- 成员表列**不可为空**，至少要有一个成员

## 结构体类型声明的说明 (2)

- 注意 {} 不表示复合语句，其后有分号
- 同一结构体的成员不能重名
- 不同结构体的成员可以重名
- 结构体成员和其他变量可以重名
- 结构体类型与其成员或其他变量可重名
  - `struct test { int test; } test;`
- 结构体类型名称是 `struct` 结构体名，注意 `struct` 关键字不能省略



## 结构体类型声明的说明 (3)

- 即使两个结构体声明中的成员类型、名称、顺序都完全一致，它们也是不同的类型
- 结构体类型也要“**先声明，后使用**”
- 如果结构体类型声明在函数内部，则该函数之外无法引用此结构体类型
- **一般**把结构体类型声明放到**文件最前面**
- 也可以把结构体类型声明放在**头文件里**

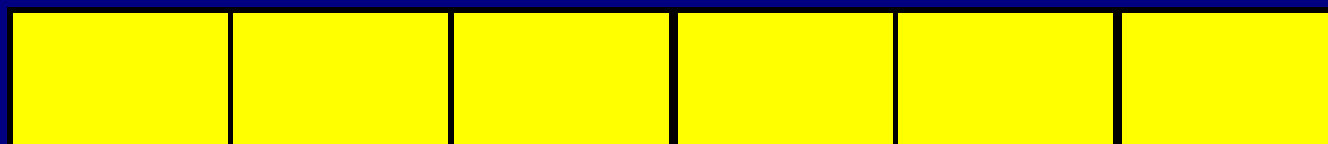
# 结构体变量的定义 (1)

- 先声明结构体类型再定义变量

```
struct student {
```

编译时系统为其分配存储空间:

student1



num    name[]    sex    age    score    addr[]

```
struct student stu1, stu2;
```

## 结构体变量的定义 (2)

- 在声明结构体类型的同时定义变量

```
struct student {  
    unsigned num;  
    char      name[20];  
    char      sex;  
    unsigned  age;  
    float     score;  
    char      addr[32];  
} stu1, stu2;
```

## 结构体变量的定义 (3)

- 直接定义无名结构类型变量

```
struct {  
    unsigned num;  
    char      name[20];  
    char      sex;  
    unsigned  age;  
    float     score;  
    char      addr[32];  
} stu1, stu2;
```

## 结构体声明和变量嵌套定义举例

```
struct date {  
    int year, month, day;  
};  
  
struct student{  
    unsigned    num;  
    char        name[20];  
    char        sex;  
    struct date birthday;  
    float       score;  
} stu1, stu2;
```

# 结构体变量的初始化

- 按照成员的顺序和类型对成员初始化

```
struct date date1 = {1988, 10, 20};  
struct student stu = {  
    1001,          /*unsigned num*/  
    "Tom",        /*char name[20]*/  
    'M',          /*char sex*/  
    {1983, 9, 20}, /*struct date birthday*/  
    93.5          /*float score*/  
};
```

# 结构体变量中成员的引用

- 一般形式
  - 结构体变量名. 成员名
- 成员运算符 .
  - 具有最高的优先级，自左向右结合

# 结构体变量中成员的引用举例

```
struct student stu;  
...  
scanf("%f", &stu.score);  
stu.num = 12345;  
stu.birthday.month = 11;  
stu.score = sqrt(stu.score) * 10;  
strcpy(stu.name, "Mike");  
printf("No.%d:", stu.num);
```



## 结构体变量引用规则

- 1) 如果结构变量为**嵌套结构**，就应使用成员运算符层层结合，一直找到最低一级的成员，只能对最低一级的成员进行赋值或存取运算。
  - `stu.birthday.month`
- 2) 成员属于变量，故可像变量一样进行各种操作。
  - `sum=stu1.score+stu2.score;`
  - `stu1.age++;`
- 3) 可以**引用成员的地址**，也可引用**结构变量的地址**
  - `scanf("%d", &stu1.num);`
  - `printf("%u %u \n", &stu1, &stu1.num);`

## 结构体变量整体引用

- 结构体类型变量之间可以直接相互赋值
  - 实质上是两个结构体变量相应的存储空间中的所有数据直接拷贝
  - 包括复杂类型在内的所有结构体成员都被直接赋值，如字符串、结构体类型等
- 函数的实参和形参可以是结构体类型，并且遵循实参到形参的单向值传递规则
- 为了提高程序的效率，函数的参数多使用结构体类型指针

## 结构体变量整体引用举例9\_1.c

```
struct student stu1, stu2={1002, "Eva", 'F',  
                           {1989, 11, 4}, 89.0};
```

```
void print(struct student s)
```

```
{
```

```
    printf("1002 , Eva , F, 1989.11.04 , 89.0\n");
```

```
    Press any key to continue
```

```
}
```

```
void main ()
```

```
{
```

```
    stu1 = stu2;          /* 直接赋值 */
```

```
    print(stu1);          /* 1002,Eva,F,1989.11.04,89.0 */
```

```
}
```

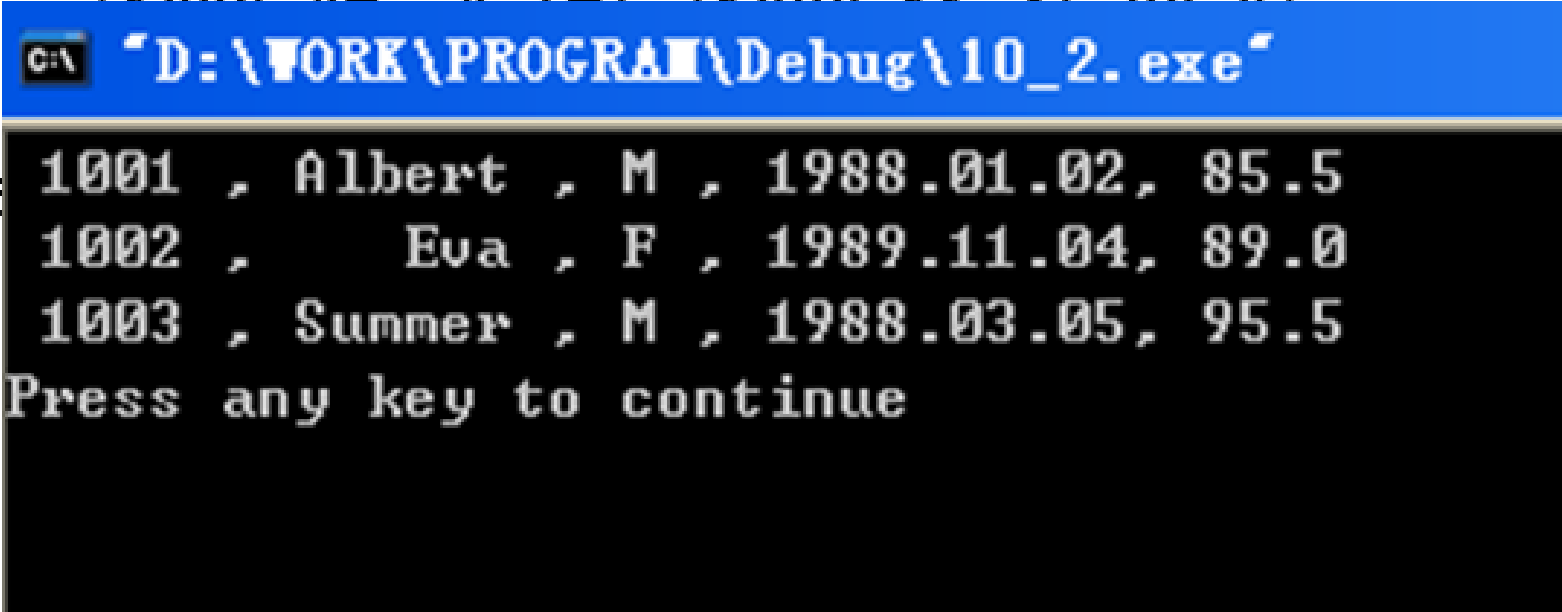
## 9.2 结构体数组

- 结构体数组的用法与基本类型数组类似
  - 定义、初始化、引用等
- 结构体数组可用于表示二维表格
- 举例

```
struct student s[10];  
for (i=0; i<10; i++)  
    scanf("%d %s %c %d %f",  
        &s[i].num, s[i].name, &s[i].sex,  
        &s[i].age, &s[i].score);
```

## 结构体数组初始化及应用举例9\_2.c

```
struct student stu[] = {  
    {1001, "Albert", 'M', {1988, 1, 2}, 85.5},  
    {1002, "Eva", 'F', {1989, 11, 4}, 89.0},  
    {1003, "Summer", 'M', {1988, 3, 5}, 95.5}
```



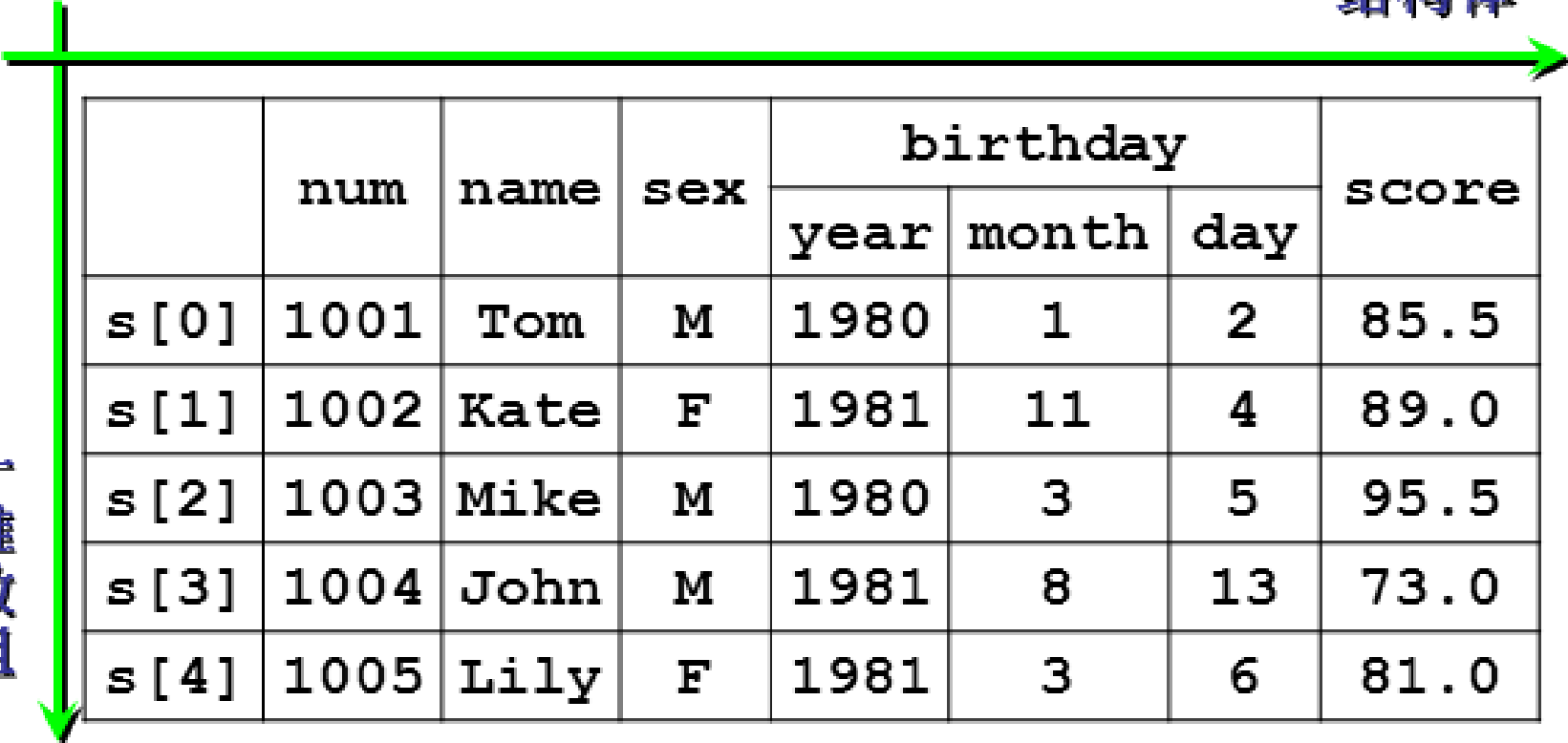
```
C:\ "D:\WORK\PROGRAM\Debug\10_2.exe"  
1001 , Albert , M , 1988.01.02, 85.5  
1002 , Eva , F , 1989.11.04, 89.0  
1003 , Summer , M , 1988.03.05, 95.5  
Press any key to continue
```

```
stu[i].birthday.day, stu[i].score);
```

# 结构体数组与二维表

```
struct student s[5];
```

结构体

一  
维  
数  
组

	num	name	sex	birthday			score
				year	month	day	
s[0]	1001	Tom	M	1980	1	2	85.5
s[1]	1002	Kate	F	1981	11	4	89.0
s[2]	1003	Mike	M	1980	3	5	95.5
s[3]	1004	John	M	1981	8	13	73.0
s[4]	1005	Lily	F	1981	3	6	81.0

## 指向结构体的指针

- 结构体变量被定义后，编译时就为其在内存中分配一片**连续的存储单元**。
- 该片内存单元的**起始地址**称为该结构变量的**指针**。
- 可以设立一个**指针变量**，用来**存放这个地址**。当作一个结构变量的起始地址。
- 赋予一个指针变量时，就称该指针指向这个结构变量。

# 结构体和指针

- 1、指向某一结构体指针的说明方式:

`struct 结构名 *变量名;`

`struct {成员说明表} *变量名;`

- 2、将一个函数的返回值说明为指向某结构的指针:

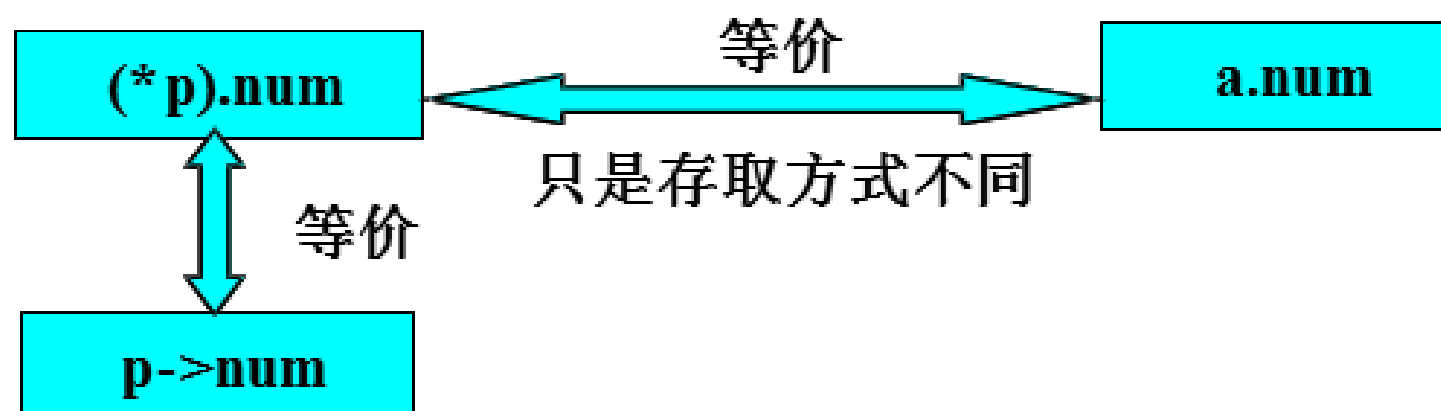
`struct 结构体 *函数名 (形参说明表)`  
`{ ... 函数体 }`



- 3、指向成员的运算符  $\rightarrow$ 
  - 结构体指针  $\rightarrow$  成员名

`struct student *p, a; p=&a; .....`

则根据指针的运算规则，对p所指向的结构变量a的分量num的引用方式可以如下：



## 指向结构体数组的指针

- 指向结构体数组的指针
  - 与指向其他基本类型数组的指针用法类似
  - 具有最高的优先级，自左向右结合
- 举例

```
struct student stu[10], *p=stu;  
++p->num; /* 同++(p->num); */  
p++->num;  
(++p)->num;  
(p++)->num;
```

## 4 结构与函数

早期的C不允许用结构变量作为函数的参数，  
解决结构变量的传递的办法：

- (1) 用结构变量的成员作参数，属“传值”方式；
- (2) 用取地址运算符‘&’取结构变量的地址，  
将指向结构变量（或数组）的指针作实参，属  
“传地”方式；
- (3) 函数的返回值也可以是一个指向结构的指针。

## 结构体指针作函数参数举例9\_3.c

```
void main ()
{
    int i;
    struct student stu[20], *p;
    p=stu;//p=&stu[0];
    scanf("%d%s%d.", &p->num, p->name, &p->birthday.y);
    p++;
    scanf("%d%s%d.", &p->num, p->name, &p->birthday.y);

}
```

## 结构体应用之一（编程要点、步骤与算法示例）

**题目：**定义一个结构数组，描述40个学生的学号、姓名、三门课程的成绩**及平均成绩**，并以函数形式实现以下功能：

- 读入学生的前五项数据；
- 计算平均成绩；
- 按平均成绩以递减顺序排序；
- 打印输出排序后的成绩。

## 程序设计： 分析题意明确要求

- 数据描述，定义一个结构数组，其元素可有六个成员组成：

学生 (num, name[16],score1,score2,score3,average)

Int char 类型相同可采用 int score[4]表示

- 定义四个功能函数，考虑确定相关的功能和函数传递方式：本例采用形实参数结合的形式。

形参一：采用指向结构体数组的指针或者结构体数组

形参二：学生个数

```
void read(STUDENT *p, int n);
```

```
void ave(STUDENT s[ ], int n);
```

- 结构类型应在外部定义，以便各函数应用定义相应的变量；
  - 可考虑适当的采用符号常数形式，以便程序的书写、阅读和调试。
- 2) 采用自顶向下逐步求精的算法编程思想与方法。

## 结构体应用举例9\_4.c

```
#include<stdio.h>
#define N 5
#define STUDENT struct student
STUDENT{
    int num;
    char name[16];
    int score[4];
};
```



```
void main()
{
    STUDENT stu[N];
    void read(STUDENT *p, int n); //函数声明
    void ave(STUDENT s[ ], int n);
    void sort(STUDENT s[ ], int n);
    void print(STUDENT *p, int n);
    read(stu,N);           //函数调用
    ave(stu,N);
    sort(stu,N);
    print(stu,N);
}
```

```
void read(STUDENT *p, int n)    //读入学生信息
{
    int i,j;
    for(i=0;i<n;i++,p++)
    {
        scanf("%d%s",&p->num,p->name);
        for(j=0;j<3;j++)
            scanf("%d",&p->score[j]);
    }
}

void ave(STUDENT s[ ], int n){    //求平均成绩
    int i,j,sum;
    for(i=0;i<n;i++){
        for(j=sum=0;j<3;j++)
            sum=sum+s[i].score[j];    //求总成绩
        s[i].score[3]=sum/3;    //求平均成绩并保存
    }
}
```

```
void sort(STUDENT s[ ],int n)  //按照成绩排序
{
    int i,j,k;
    STUDENT temp;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++)      //选择排序法
            if(s[k].score[3]<s[j].score[3])k=j; //记录位置
        if(k!=i)
            {
                temp=s[i];s[i]=s[k];s[k]=temp;
            } //每一个结构体变量可以整体引用,赋值
    }
}
```

```
void main(OSTUDENT *p, int n)
```

```
{
```

```
C:\ "D:\WORK\课程\Debug\10_4.exe"
```

```
051801zhang 90 90 90
```

```
051802qin 92 94 93
```

```
051803wang 98 98 96
```

```
051804sun 91 90 94
```

```
051805li 90 90 90
```

```
51803          wang      98      98      96      97
```

```
51802          qin       92      94      93      93
```

```
51804          sun       91      90      94      91
```

```
51801          zhang     90      90      90      90
```

```
51805          li        90      90      90      90
```

```
Press any key to continue_
```

```
}
```

## 9.3 动态存储分配函数

- 静态数据结构：
  - 前面讨论的各种基本类型和组合类型的数据都属静态数据结构，它们所占存储空间的大小在程序的说明部分就已经确定。
  - 如变量、数组、结构等，它们不能在程序的执行过程中加以改变。
- 动态数据结构：
  - 它是在程序的执行过程中动态建立起来的，故这种数据结构的规模大小在程序执行期间可动态地变化。

# 动态存储分配函数

- 利用动态数据结构可以解决一些静态数据结构难以解决的问题。如想在数组中**插入**或**删除**一个元素就比较困难,而动态数据结构就能方便的解决这类问题。
- 动态数据结构中最基本的形式是**链表**和**二叉树**,它们在应用软件和系统软件的设计中非常有用。

# 动态存储分配函数

- 动态分配存储
  - 根据需要开辟或释放存储单元
- 相关函数
  - malloc函数 (memory allocation)
  - calloc函数
  - free函数
- 说明
  - 应包含malloc.h或stdlib.h

```
float *p;  
p=(float *)malloc (sizeof(float))
```

- 函数原型 (Page 319)
  - void \* malloc (unsigned size);
- 参数
  - size: 分配存储空间的字节数
- 返回值
  - 若成功, 返回指向分配区域起始地址的指针
  - 若失败, 返回NULL



```
int *pb;  
pb=(int *) malloc (n*sizeof(int));  
if(pb==null)  
{  
    puts(“memory allocation error.”);  
    exit(1);  
}  
free(pb);
```

```
float *p;  
p=(float *)malloc (5 , sizeof(float))
```

- 函数原型

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

- 参数

- n :分配内存的项目数

- size:分配内存的每个项目的字节数

- 返回值

- 若成功, 返回指向分配区域起始地址的指针

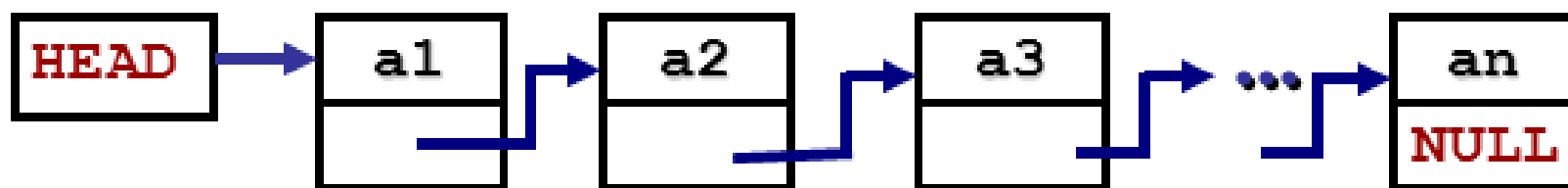
- 若失败, 返回NULL

# free函数

- 函数原型
  - `void free(void *ptr);`
- 参数
  - `ptr`: 要释放的内存区地址
- 说明
  - 释放`ptr`指向的内存区
  - 释放后的内存区能够分配给其他变量使用

## 结构体的应用—链表 (Link List)

- 链表是一种最简单的数据结构，利用它可以在一张表中随机的删除或插入元素。
- 例如：学生信息表 ( $a_1, a_2, a_3, \dots, a_n$ )



```
struct node {  
    int data;  
    struct node *next;  
};  
struct node *head;
```

# 链表的操作

- 链表的**建立**
  - 从链尾到链头：新结点插入到**链头**
  - 从链头到链尾：新结点插入到**链尾**
- 链表的**遍历**
- **删除**结点
  - 根据一定的条件，删除一个或多个结点
- **插入**结点
  - 根据一定的条件，把新结点插入到指定位置

## 尾插法建立链表算法

Head=NULL;           //置空链表

输入一个循环控制数据;

while(数据值不是结束标志){

    申请新结点，用指针p指向该结点;

    给新结点赋予相关数据值;

    if(链表为空){

        表示新插的结点是首结点

        头指针指向首结点 head = p;}

    else{

        将新结点插入到尾指针指向的结点之后 rear->next=p;

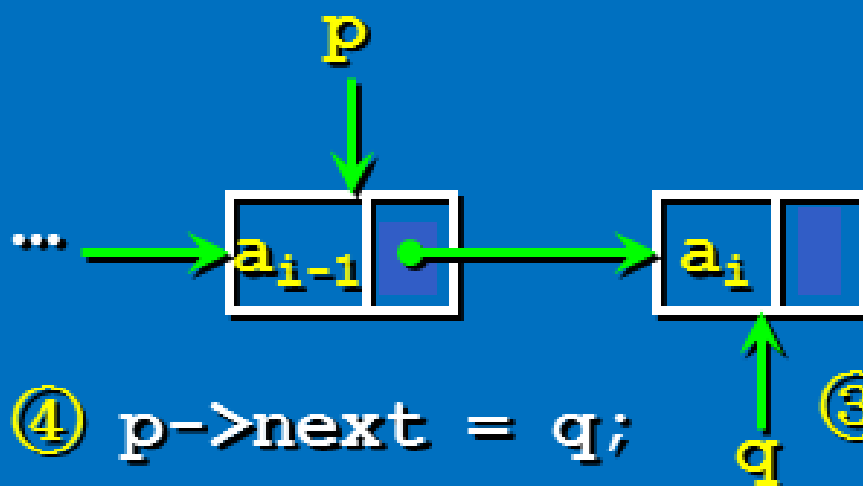
        使尾指针指向新结点, rear=p;}

    输入一个循环控制数据;

}

# 建立链表 (从链头到链尾)

① `for (i=0; i<n; i++)`



⑤ `p = q;`

④ `p->next = q;`

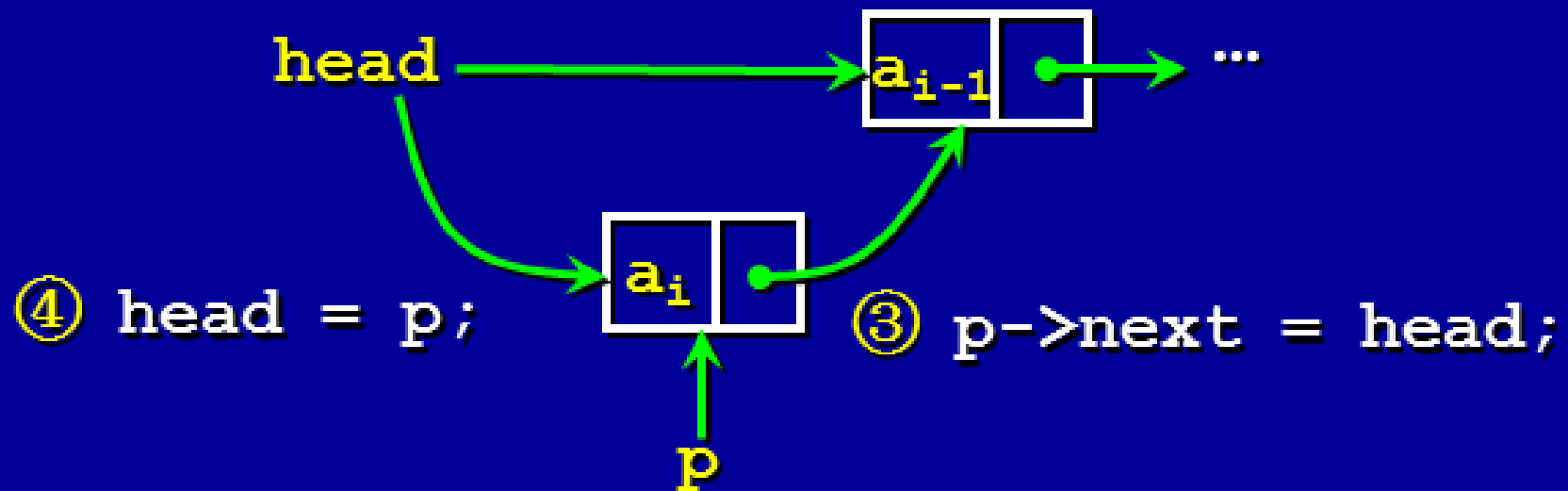
③ `q->next = NULL;`

② `q = malloc(sizeof (struct node));`

□ `q->data = a[i];`

# 建立链表 (从链尾到链头)

① `for(i=0; i<n; i++)`



② `p = malloc(sizeof (struct node));`  
    `p->data = a[i];`



## 结构体应用举例9\_5.c

```
#include<stdio.h>
#include<malloc.h>
#define STUDENT struct student
#define LEN sizeof(STUDENT)
STUDENT{           //定义结构体（结点）的类型
    long num;
    float score;
    STUDENT *next;
};
int  n;           //全局变量n用来统计链表结点的个数
```

```
STUDENT *create()
{
    //建立链表函数，表尾插入算法
    STUDENT *head, *p1, *p2;
    n=0;
    p1=p2=(STUDENT *)malloc(LEN);
    scanf("%ld%f",&p1->num,&p1->score);
    head=NULL;
    while(p1->num!=0)
    {
        n=n+1;
        if(n==1)head=p1;           //建立第一个结点
        else p2->next=p1;         //建立其余结点
        p2=p1;
        p1=(STUDENT *)malloc(LEN);
        scanf("%ld%f",&p1->num,&p1->score);
    }
    p2->next=NULL;                //NULL为尾节点结束标志
    free(p1); return(head);       //返回链表头指针
}
```

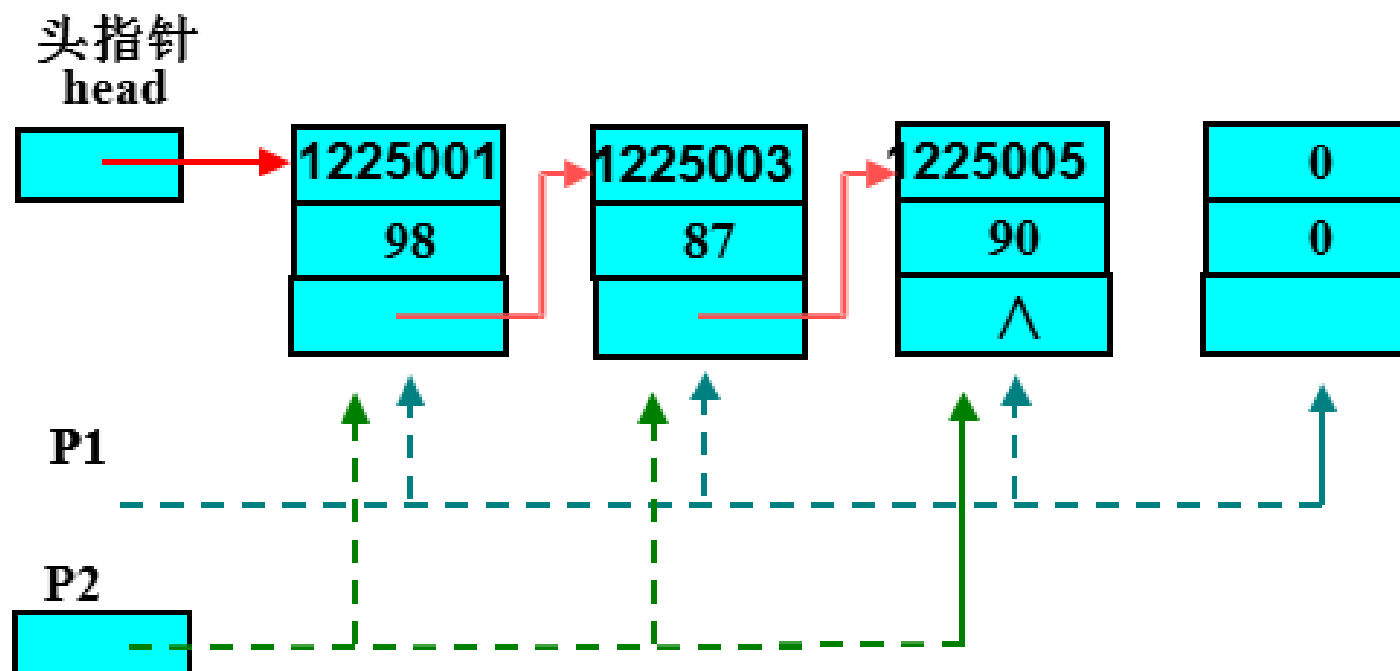
运行情况：//建立链表

输入：1225001 98

1225003 87

1225005 90

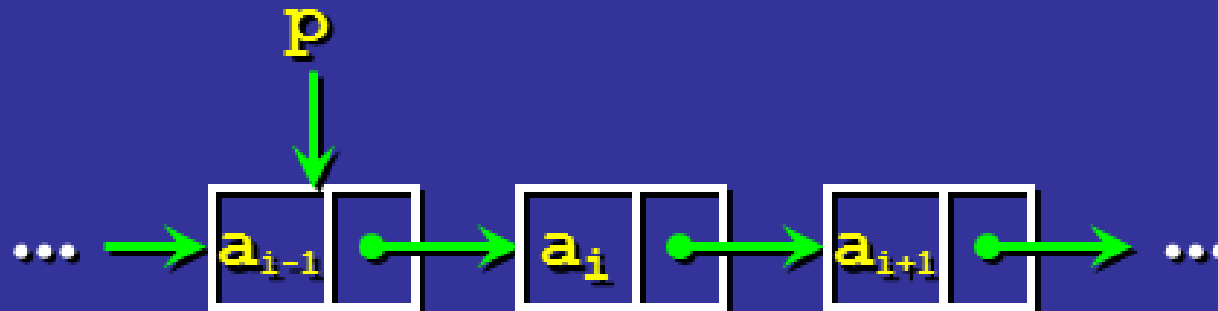
0 0



# 遍历链表

① while (p)

③ p = p->next;



② printf ("%d", p->data);

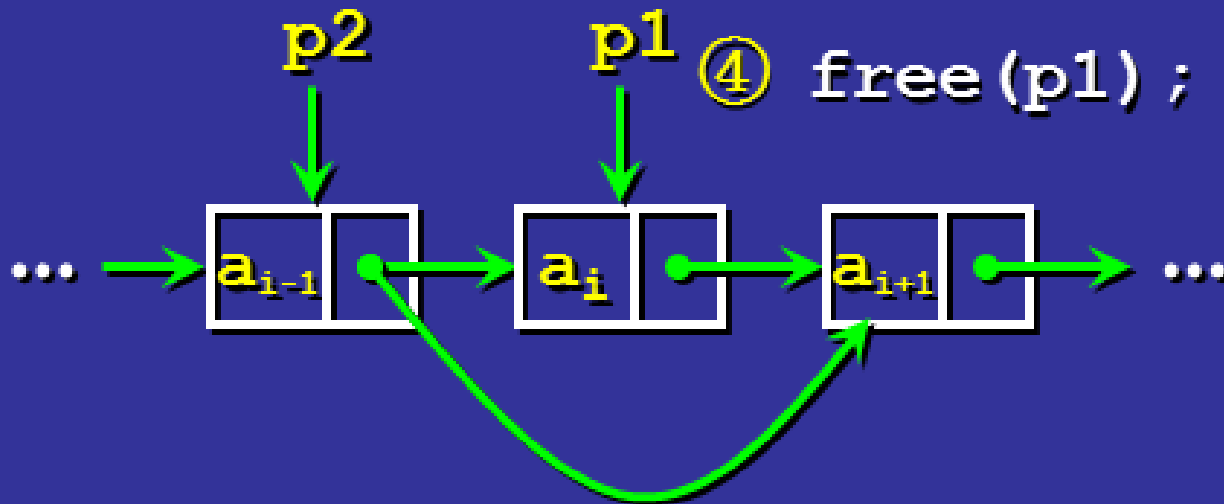
```
void print(STUDENT *head)    //遍历打印列表  
{  
    STUDENT *p;  
    p=head;  
    if(head!=NULL)  
        do{  
            printf("%ld %f\n",p->num,p->score);  
            p=p->next;  
        }while(p!=NULL);  
}
```

# 删除结点

① `if (p2->next` 满足删除条件)

② `p1 = p2->next;`

④ `free (p1);`



③ `p2->next = p1->next;`

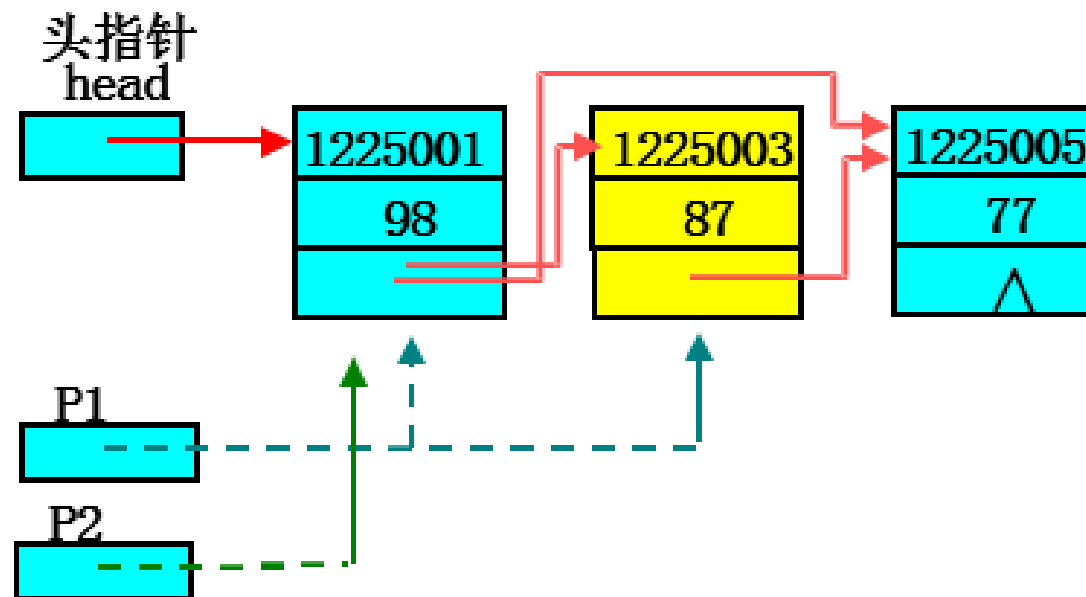
```
STUDENT *del(STUDENT *head, long num)
{
    STUDENT *p1,*p2;                //删除指定节点
    if (head==NULL)
        {printf("\nList null! \n"); return(head);}
    p1=head;
    while (num!=p1->num&& p1->next!=NULL) //查找定位
    {
        p2=p1;p1=p1->next;
        if (num==p1->num)
        {
            if (p1==head) head=p1->next; //删除表头元素
            else
                p2->next=p1->next;        //删除中间或尾元素
            printf("delete:%ld\n",num);
            n=n-1; free(p1);
        }
        else printf("%ld not been found\n", num);
    }
    return(head);
}
```

input the deleted number:1225003      //删除结点

delete:1225003

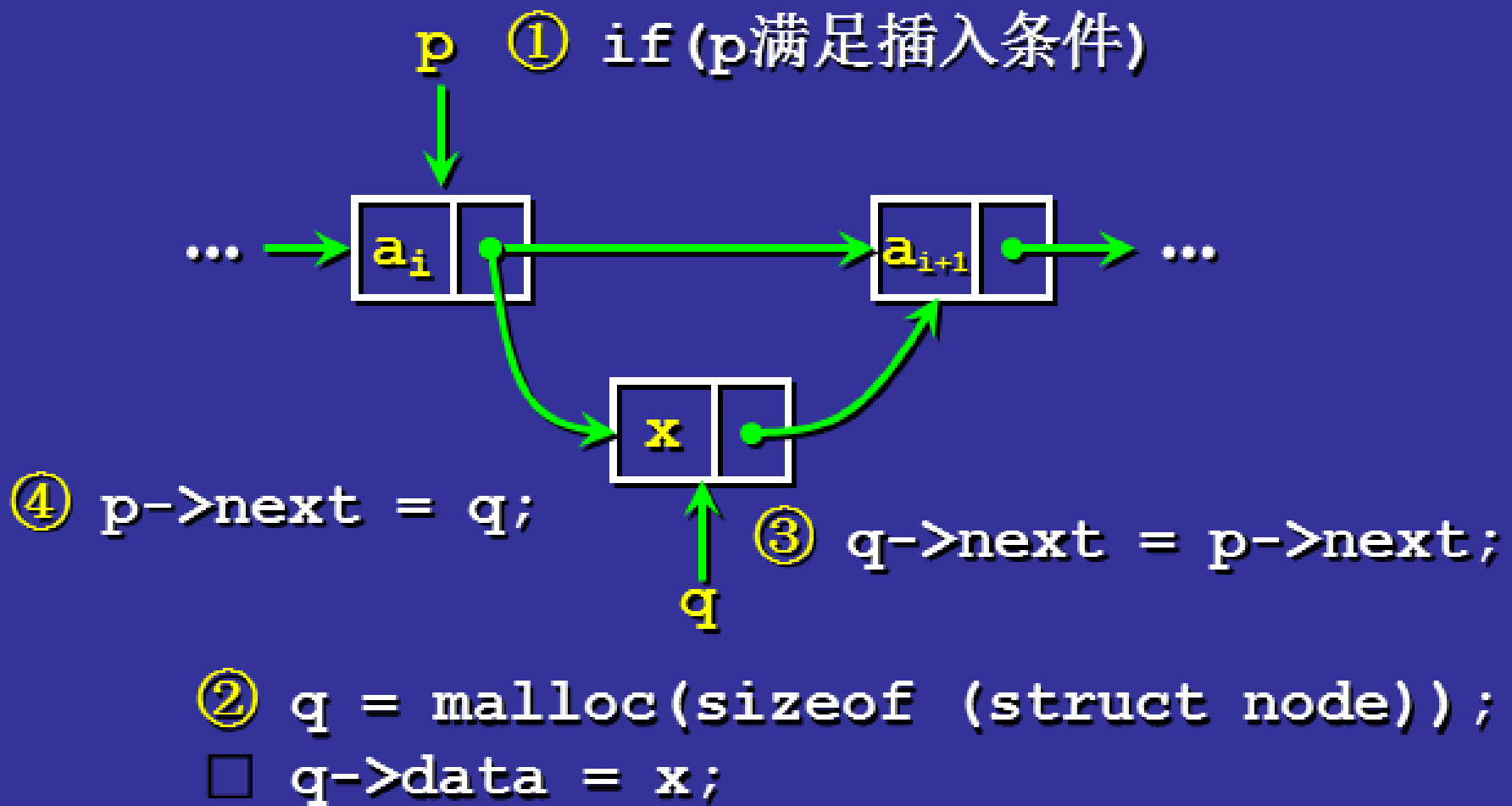
1225003 98.000000

1225005 77.000000



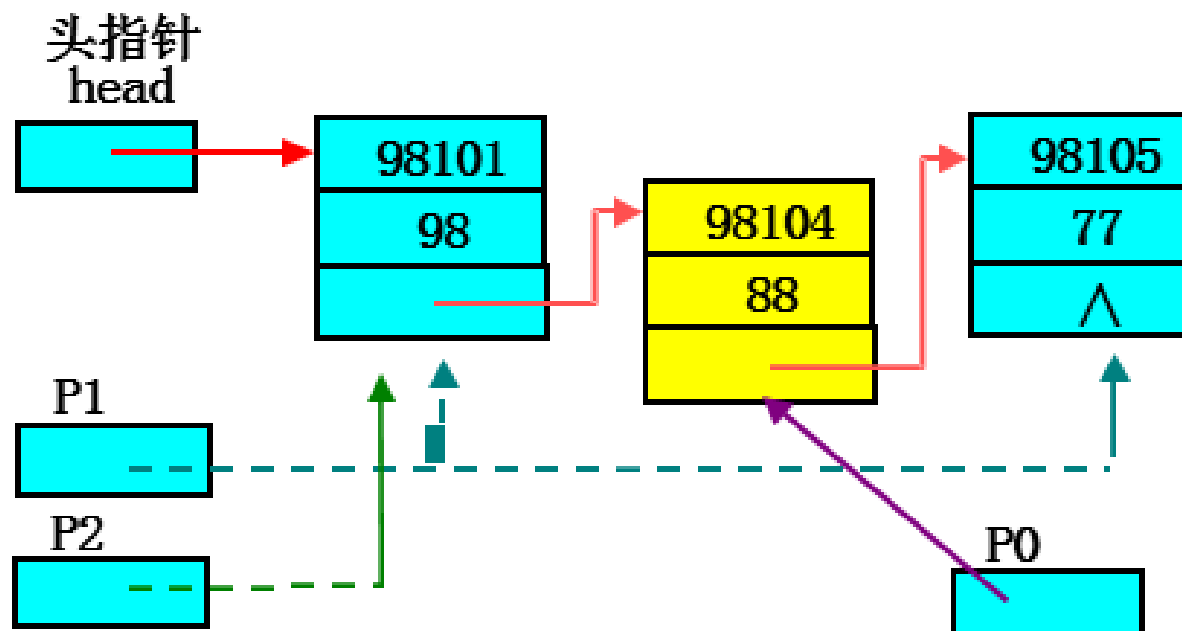


# 插入结点

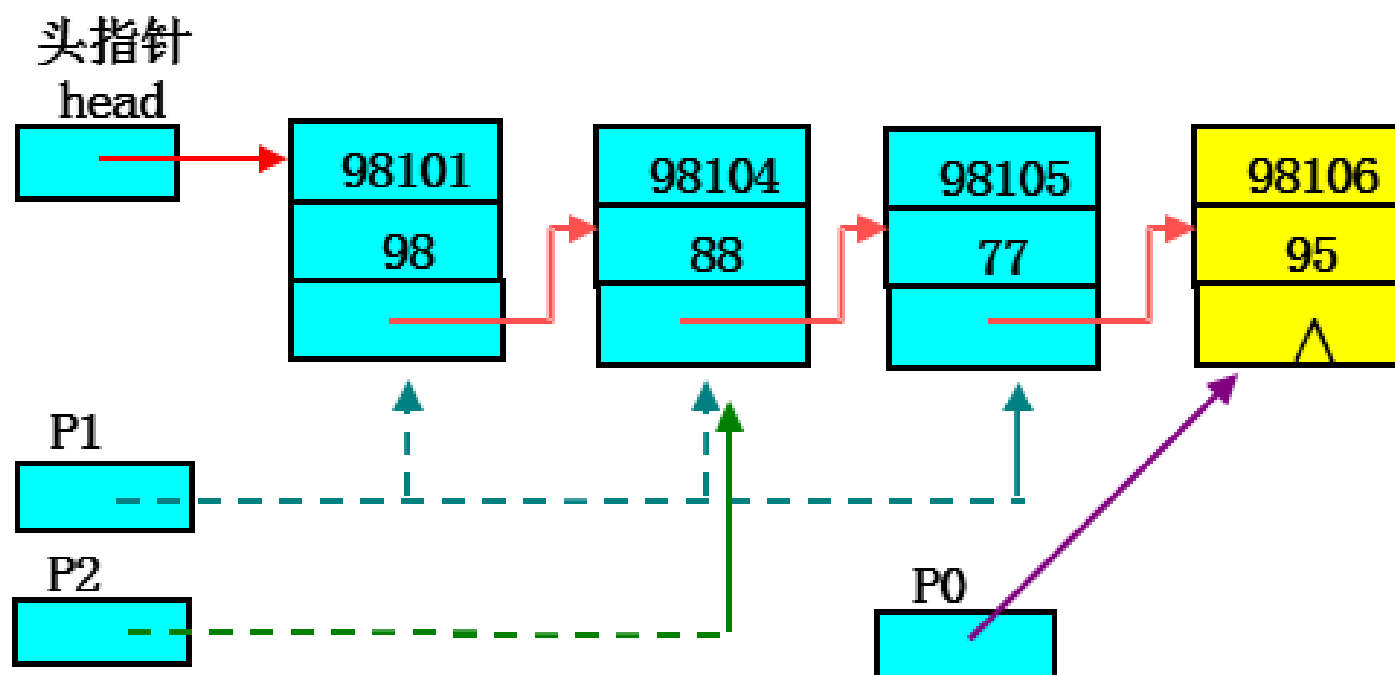


```
STUDENT *insert(STUDENT *head, STUDENT *stud)
{
    //按顺序插入元素算法，也可用于建立有序链表
    STUDENT *p0,*p1,*p2; p1=head; p0=stud;
    if(head==NULL)           //若链表为空，则建立头结点
        {head=p0; p0->next=NULL;} //否则查找定位
    else
        while( (p0->num>p1->num)&&(p1->next!=NULL) )
            { p2=p1; p1=p1->next; }
    if (p0->num<=p1->num)
        { if (head==p1) { head=p0; p0->next=p1; } //表头插入
          else
            { p2->next=p0; p0->next=p1; } //中间插入
          }
    else { p1->next=p0; p0->next=NULL; } //表尾插入
    n=n+1; return(head);
}
```

```
input the insert record:98104 88 //插入结点
98101 98.000000
98104 88.000000
98105 77.000000
```



```
input the insert record:98106 95
98101 98.000000
98104 88.000000
98105 77.000000
98106 95.000000
input the insert record:0 0
```



```
void main()                                //9_6.c
{  STUDENT *head,*s; long del_num;
  head=create();
  print(head); //建立链表并遍历打印
  printf("\n Input the deleted number:");
  scanf("%ld", &del_num);
  while(del_num!=0) //可按学号连续删除, 学号为0时结束
  {  head=del(head,del_num); //删除指定结点
    print(head);             //遍历打印检测运行状态
    printf("\n Input the deleted number:");
    scanf("%ld",&del_num);
  }
```

```
printf("\n Input the insert record:");  
s=(STUDENT *)malloc(LEN);  
                                //申请插入节点的存储空间  
scanf("%ld%f",&s->num,&s->score);  
while(s->num!=0) //可连续插入结点，学号为0时结束  
{ head=insert(head,s);  
  print(head);  
  printf("\n Input the insert record:");  
  s=(STUDENT *)malloc(LEN);  
  scanf("%ld%f",&s->num, &s->score);  
}  
}
```

# 链表操作中需要注意的几个问题

- 注意考虑几个特殊情况下的操作
  - 链表为**空表** (head=NULL)
  - 链表只有一个结点
  - 对链表的**第一个**结点进行操作
  - 对链表的**最后一个**结点进行操作
- 最后一个结点的next指针应为NULL
- 可以定义一个结构体类型用于表示结点的数据部分，以便于对数据的操作

# 结构体作业

**Page279: 1、2、3、5**

**Page279: 6、8、9**

**(12月2日交)**



# 上机-结构体

- **Page180** 实验内容1,2,3,4
- **Page246** 综合测试三,五、程序设计1
- 范例3

重要补充：阅读**P183** 常见错误分析

```
void SelectSort(student* p[],int n){  
    int i,j,k;  student* temp;  
    for(i=0;i<n;++i){  
        k=i;  
        for(j=i+1;j<n;++j){  
            if(p[j]->grade<p[k]->grade)  
                k=j;        }  
        if(k!=i){  
            temp=*(p+k);  
            *(p+k)=*(p+i);  
            *(p+i)=temp;}  
    }  
}
```

```
void BubbleSort(student* p[],int n){  
    int i,j;  
    student* temp;  
    for(i=0;i<n;++i)  
        for(j=0;j<n-i-1;++j)  
            if(p[j]->grade>p[j+1]->grade)  
            {  
                temp=p[j];  
                p[j]=p[j+1];  
                p[j+1]=temp;  
            }  
}
```

