9

结构体与共用体

9.1 引言

9.2 结构变量的定义、引用、初始化及赋值

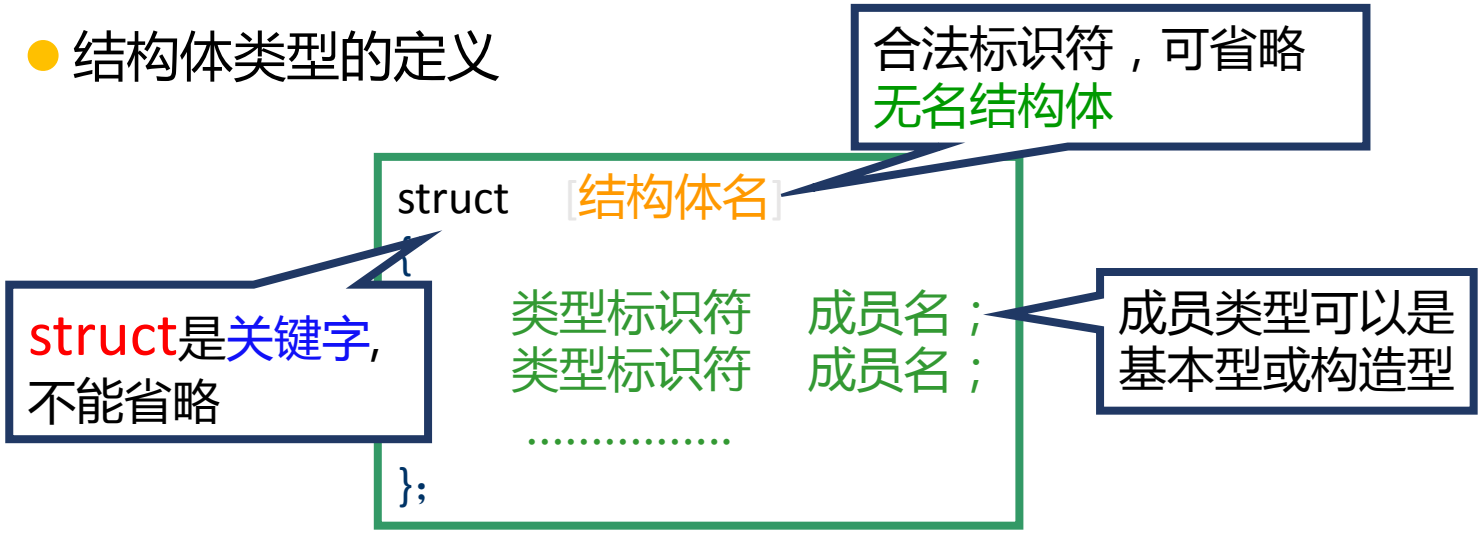
9.3 结构体数组

9.4 结构与链表

9.5 共用体和枚举类型

9.1 引言

- 构造数据类型，不同类型的数据需要统一处理
- 例如学生信息，需要学号、姓名、性别、年龄、班级等等，数据类型不一致，使用数组会难以实现
- 结构体是一种构造数据类型，把不同类型的数据组合成一个整体
- 结构体类型的定义



例
```
struct  student{
        long  no;
        char  name[16];
        char sex;
        int age;
        float score;
        char addr[30];
};
```

- 每个成员必须有自己的数据类型，定义后的需要有分号
- 相同类型、位置连续的结构成员可定义在一起，如

```
struct date{
int  month , day , year;
}
```

- 结构体类型定义的位置既可以在函数内部，也可以在函数外部
- 结构体类型的定义描述结构的组织形式，不分配内存

9.2 结构变量的定义、引用、初始化及赋值

结构体变量的定义

- 先定义结构体类型，再定义结构体变量

```
struct    结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
      ……
};
struct 结构体名  变量名表列；
```

```
例    struct  student{
            long  no;
            char  name[16];
            char sex;
            int age;
            float score;
            char addr[30];
      };
      struct student stu1,stu2;
```

结构体变量的定义

- 定义结构体类型的同时定义结构体变量

```
struct    结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
      ……
}变量名表列；
```

```
例   struct  student{
            long  no;
            char  name[16];
            char sex;
            int age;
            float score;
            char addr[30];
      }stu1,stu2;
```

结构体变量的定义

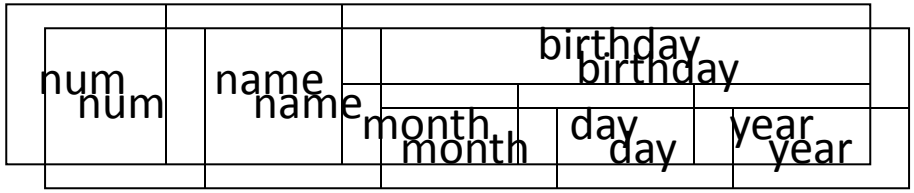
- 直接定义结构体变量（省略结构体名）

```
struct
{
        类型标识符    成员名；
        类型标识符    成员名；
        ……
}变量名表列；
```

例
```
struct
{       long  no;
        char  name[16];
        char sex;
        int age;
        float score;
        char addr[30];
}stu1,stu2;
```

- 结构体类型与结构体变量概念不同
  - 类型:不分配内存；　　变量:分配内存
  - 类型:不能赋值、存取、运算;　　变量:可以
- 结构体可嵌套
- 结构体成员名与程序中变量名可相同，不会混淆

```
struct  student
{   int  num;
    char name[20];
    struct  date
    {    int month;
         int day;
         int year;
    } birthday;
}stu;                          day;
}stu;
```



结构体变量的引用

- 结构体变量不能整体引用，只能引用变量成员

引用方式： 结构体变量名 . 成员名

- 可以将
- 结构体

例 struct student
{ long num;
char name[20];

例 struct student
{ long num;
char name[20];
struct date
{ int month;
int day;
int year;
} birthday;
} stu1,stu2;

一个结构体变量

u1.num=2016003;

if(stu1==stu2)
……..

printf("%d,%s,%c,%d,%f,%s\n",stu1);

u1.score=85.5;

stu1.birthday.month=12;

stu2=stu1;

re=stu2.score;

in","M",19,87.5,"DaLian" };

name++;

num		birthday		
		month	day	year

```
main()
{
      struct  student
      {
            int number;
            char name[6];
            char sex;
            int age;
            char address[20];
       } ;
      printf("%d\n ",sizeof(struct student));
}
```

结构体变量的初始化和赋值

- 形式一

```
struct    结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
       ……
};
struct  结构体名   结构体变量={初始数据}；
```

```
例    struct  student
      {       long num;
              char  name[20];
              char sex;
              int age;
              char addr[30];
      };
      struct  student  stu1={2016015,“Wang Lin”,‘M’,19, “200 Beijing Road”};
```

结构体变量的初始化和赋值

- 形式二

```
struct    结构体名
{
      类型标识符    成员名；
      类型标识符    成员名；
       ……
} 结构体变量={初始数据}；
```

例
```
struct  student
{       long num;
        char  name[20];
        char sex;
        int age;
        char addr[30];
} stu1={2016015,“Wang Lin”,‘M’,19, “200 Beijing Road”};
```

结构体变量的初始化和赋值

- 形式三

```
struct
{
    类型标识符    成员名；
    类型标识符    成员名；
    ……
} 结构体变量={初始数据}；
```

例
```
struct
{    long num;
     char  name[20];
     char sex;
     int age;
     char addr[30];
} stu1={2016015,“Wang Lin”,‘M’,19, “200 Beijing Road”};
```

- 通过键盘输入数据

```
main()
{
    struct
    {   char name[15];
        char class[12];
        long num;
    } stu;
    scanf("%s",stu.name);
    scanf("%s",stu.class);
    scanf("%ld",&stu.num);
    printf("%s,%s,%ld\n",stu.name,stu.class,stu.num);
}
```

亦可用以下赋值语句：

```
strcpy(stu.name,"wenli");
strcpy(stu.class,"Computer");
stu.num=2016015;
```

例.若有以下定义，则正确的赋值语句为____。

```
struct complex {
    float real;
    float image;
};
struct value {
    int no;
    struct complex com;
}val1;
```

A) com.real=1;　　B) val1.complex.real=1;

C) val1.com.real=1;　　D) val1.real=1;

9.3 结构体数组

结构体数组的定义

- 三种形式

```
struct  student
{    long num;
     char name[20];
     char sex;
     int age;
};
struct  student   stu[2];
```

```
struct  student
{    long  num;
     char name[20];
     char sex;
     int age;
}stu[2];
```

```
struct
{    long  num;
     char name[20];
     char sex;
     int age;
}stu[2];
```

结构体数组的初始化

引用方式：结构体数组名[下标].成员名

直接定义变量并初始化

```
struct  student
{
    long  num;
    char name[20];
    char sex;
    int age;
}stu[3];

struct  student  stu[3]={100,"Wang Lin",'M',20,
                         101,"Li Gang",'M',19,
                         110,"Liu Yan",'F',19};
```

strcpy(stu[0].name,"ZhaoDa");

stu[1].age++;

全部初始化时长度可省

例 用结构体编程实现统计候选人选票

```
struct person
{   char name[20];
    int count;
}leader[3]={“Li”,0,“Zhang”,0,”Wang“,0};
main()
{    int i,j;  char   leader_name[20];
   for(i=1;i<=10;i++)
   {   scanf("%s",leader_name);
       for(j=0;j<3;j++)
            if(strcmp(leader_name,leader[j].name)==0)
                leader[j].count++;
   }
   for(i=0;i<3;i++)
     printf("%5s:%d\n",leader[i].name,leader[i].count);
}
```

name	count
Li	0
Zhang	0
Wang	0

例9.1 定义结构体数组，然后赋值并输出

```c
#include<stdio.h>
struct person{
    long num;
    char name[20];
    float salary;
};
struct person ps[4];
void main()
{
    struct person *p=ps;
    int i;
    for(i=0;i<4;i++,p++){
        printf("第%d位员工的编号:",i+1);
        scanf("%ld",&(*p).num);
        printf("第%d位员工的姓名:",i+1);
        scanf("%s",(*p).name);
        printf("第%d位员工的工资:",i+1);
        scanf("%f",&(*p).salary);
    }
    for(p=ps;p<ps+4;p++)
        printf("%8ld%10s%10.2f\n",p->num,p->name,p->salary);
}
```

1、指针p的移动

2、利用指针p引用结构体数组元素的成员

例9.2 根据结构体成员的值对结构体数组进行排序

```c
#include<stdio.h>
struct person{
    long num;
    char name[20];
    float salary;
}ps[50];
void main()
{
    int i,j,k;
    struct person temp;
    for(i=0;i<5;i++){
        printf("第%d位员工的编号:",i+1);
        scanf("%ld",&ps[i].num);
        printf("第%d位员工的姓名:",i+1);
        scanf("%s",ps[i].name);
        printf("第%d位员工的工资:",i+1);
        scanf("%f",&ps[i].salary);
    }
    for(i=0;i<5;i++){
        k=i;
        for(j=i+1;j<5;j++)
            if(ps[k].salary<ps[j].salary) k=j;
        if(k!=i){
            temp=ps[k];
            ps[k]=ps[i];
            ps[i]=temp;
        }
    }
    printf("      no.        name.          salary.\n");
    for(i=0;i<5;i++)
        printf("%8ld%14s%15.2f\n",ps[i].num,ps[i].name,ps[i].salary);
}
```

排序的依据

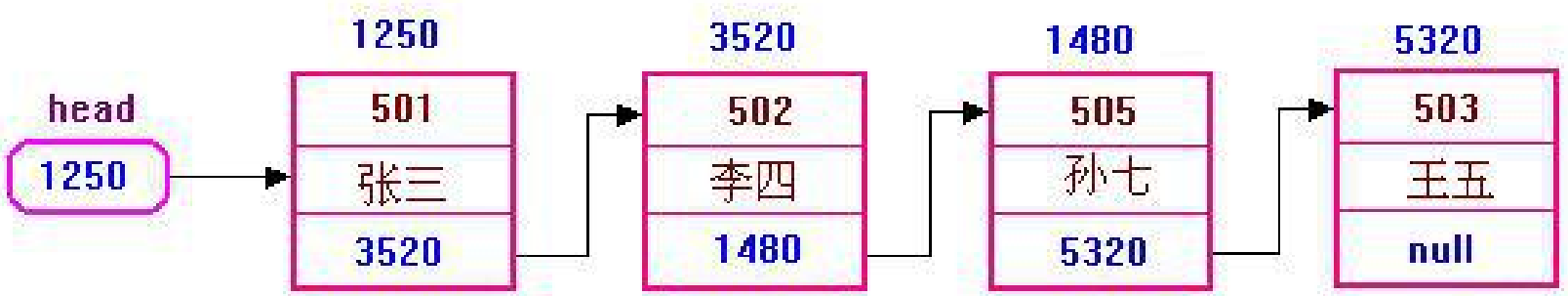
```c
#include<stdio.h>
#include<stdlib.h>
struct tm {int hours,minutes,seconds;};
delay(){
    long int t;
    for(t=1;t<=100000000;++t);
}
update(struct tm *t){
    (*t).seconds++;
    if((*t).seconds==60) {(*t).seconds=0;(*t).minutes++;}
    if((*t).minutes==60) {(*t).minutes=0;(*t).hours++;}
    if((*t).hours==24) (*t).hours=0;
    delay();
}
display(struct tm *t){
    system("cls");
    printf("%d:",(*t).hours);
    printf("%d:",(*t).minutes);
    printf("%d\n",(*t).seconds);
}
main(){
    struct tm time;
    time.hours=time.minutes=time.seconds=0;
    system("cls");
    printf("Now, press enter to begin my clock...");
    getchar();
    while(1){
        update(&time);
        display(&time);
    }
}
```

9.4 结构与链表

特点：

- 按需分配内存
- 不连续存放
- 有一个“头指针”（head）变量
- 每个结点中应包括一个指针变量，用它存放下一结点的地址。
- 最后一个结点的地址部分存放一个“NULL”(空地址）。

链表结点定义形式



定义形式：

```
struct student
{
    int number;
    char name[6];
    struct student *next;
};
```

```
struct student
{
    int number;
    char name[6];
    struct student st;
};
```

链表常用操作

p=p->next 在链表结点间顺序移动指针

将p原来所指结点中next的值赋给p，而p->next值即下一结点起始地址，故p=p->next 的作用是使p指向下一结点。

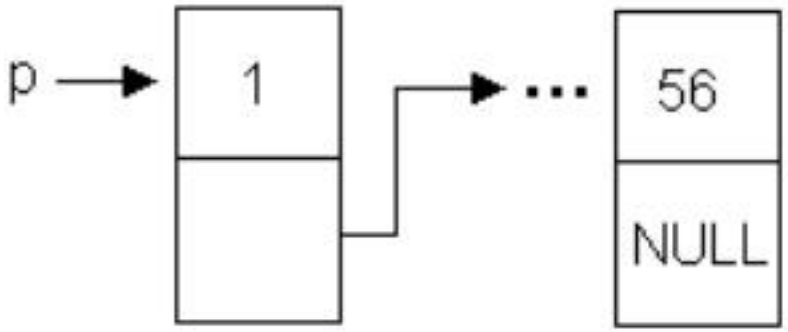
p2->next=p1 将新结点添加到现在链表中

如果p2是链表中的末结点，p1指新建结点，此句的功能是使p1所指新结点变成链表中的新的末结点。

p2->next=NULL 让p2所在结点成为链表中最后结点

例 若已建立下面的链表结构，指针p指向某单向链表的首结点

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



以下语句能正确输出该链表所有结点的数据成员data的是____。

A) for (;p!=NULL;p++) printf("%7d,",p->data);
B) for (;!p;p=p->next) printf("%7d,",(*p).data);
C) while (p)
{ printf("%7d,",(*p).data);
p=p->next;
}
D) while (p!=NULL)
{ printf("%7d,", p->data);
p++; }

链表指针的移动不能用p++

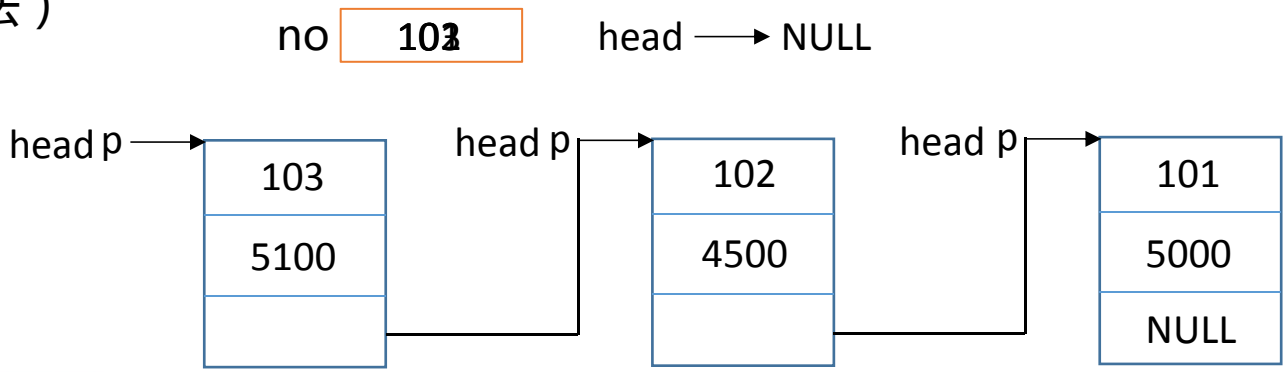
C

静态链表的建立

- 结点是如何定义的
- 结点是如何建立的
- 如何使诸结点形成链表
- 最后一个结点如何建立
- 如何从一个结点跳转到下一结点
- 如何遍历所有结点

```
#define NULL 0
struct student{
   long num;
   float score;
   struct student *next;
};
main() {
   struct student a,b,c,*head,*p;
   a.num=101;a.score=89.5;
   b.num=102;b.score=90;
   c.num=103;c.score=85;
   head=&a;
   a.next=&b;
   b.next=&c;
   c.next=NULL;
   p=head;
   while(p!=NULL);{
      printf("%ld,%.1f\n",p->num,p->score);
      p=p->next;
   }
}
```

动态链表的建立（头插法）




```c
#include<stdio.h>
#include<stdlib.h>
struct person
{
    int num;
    float salary;
    struct person *next;
};
struct person *CreateList(viod)
{
    struct person *head;
    struct person *p;
    int no;
    head=NULL;
    printf("\n输入一个职工号,输入0结束:");
    scanf("%d",&no);
    while(no!=0)
    {
        p=(struct person *)malloc(sizeof(struct person));
        p->num=no;
        printf("\n输入一个职工工资:");
        scanf("%f",&p->salary);
        p->next=head;
        head=p;
        printf("\n输入一个职工号, 输入0结束:");
        scanf("%d",&no);
    }
    return head;
}
```

动态链表的建立（尾插法）

```c
#include<stdio.h>
#include<stdlib.h>
struct person
{
    int num;
    float salary;
    struct person *next;
};
struct person *CreateList(viod)
{
    struct person *head;
    struct person *rear;
    struct person *p;
    int no;
    head=NULL;
    printf("\n输入一个职工号:");
    scanf("%d",&no);
    while(no!=0)
    {
        p=(struct person *)malloc(sizeof(struct person));
        p->num=no;
        printf("\n输入一个职工工资:");
        scanf("%f",&p->salary);
        if(head==NULL)head=p;
        else rear->next=p;
        rear=p;
        printf("\n输入一个职工号，输入0结束:");
        scanf("%d",&no);
    }
    if(rear!=NULL)
        rear->next=NULL;
    printf("\n建表结束！\n");
    return head;
}
```

遍历访问链表

遍历访问整个职员链表的函数

```
void PrintList(struct person *head)
{
    struct person *p=head;  //p指向表头
    while(p!=NULL)
    {
        printf("%d %f\n",p->num,p->salary); //输出下一个职工的信息
        p=p->next;   //使p指向下一个结点
    }
}
```

主函数，实现链表的建立和输出链表中的信息

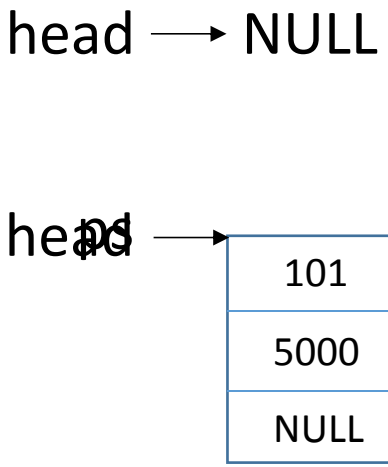
```
void main()
{
    struct person *head;
    head=CreateList();    //建立链表
    PrintList(head);      //遍历链表
}
```

链表结点的插入和删除

链表结点的插入

```
struct person *ListInsert(struct person *head,struct person *ps)
//head为链表头指针，ps指向要插入的结点，由调用函数生成好
{
    struct person *p,*q;
    if(head==NULL) //若为空表，使head直接指向插入结点皆可
    {
        head=ps;
        printf("%d\n",head->num);
        return head;
    }
    if(head->num>ps->num) //结点插入在表头
    {
        ps->next=head;
        head=ps;
        return head;
    }
    p=q=head; //使p，q指向表头结点
    while(p!=NULL&&p->num<ps->num) //查找插入位置
    {
        q=p;
        p=p->next; //使q指向p所指向的结点的前一个结点
    }
    q->next=ps; //插入新结点
    ps->next=p;
    return head;
}
```

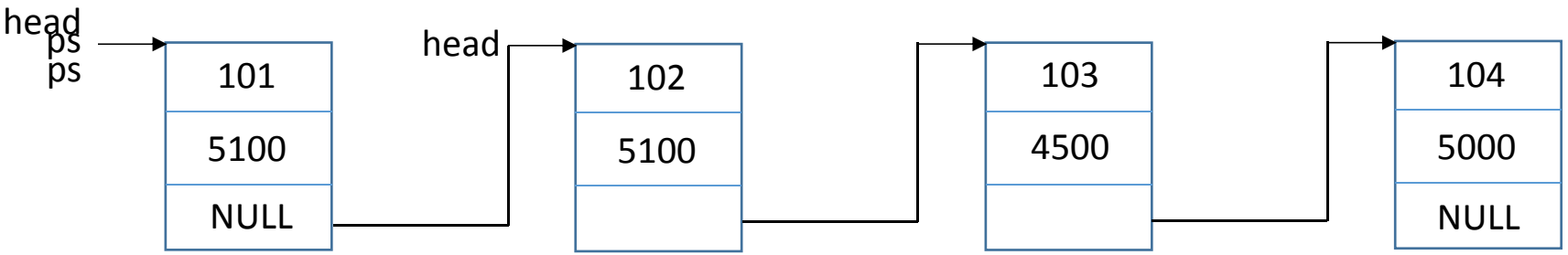
情况一：　　初始链表为空表

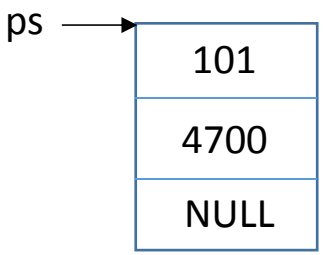


情况二：　节点插入在表头位置

① ps->next=head ;

② head=ps ;





条件判断：ps->num小于head->num

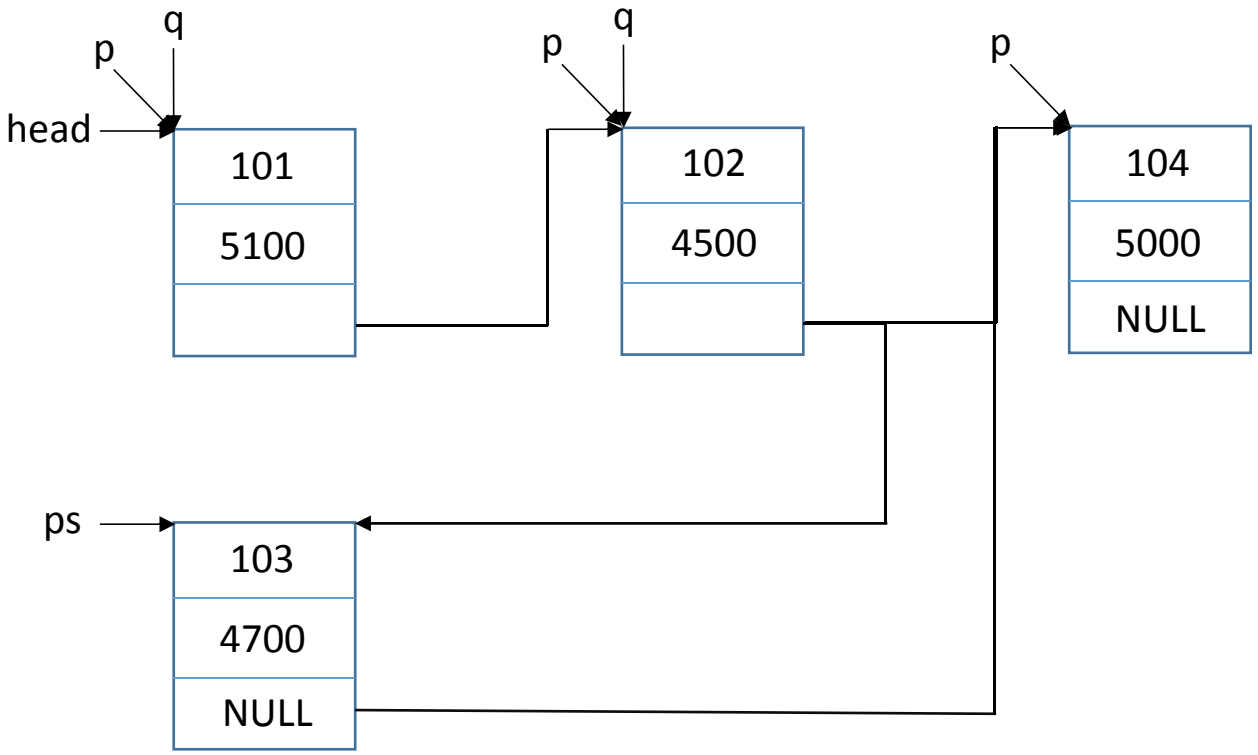
情况三： 节点插入不在表头位置

① p=q=head ;

② 查找位置：p!=NULL && p->num<ps->num ;

③ 移动指针：q=p ； p=p->next ;

④ 节点插入：q->next=ps ； ps->next=p ;



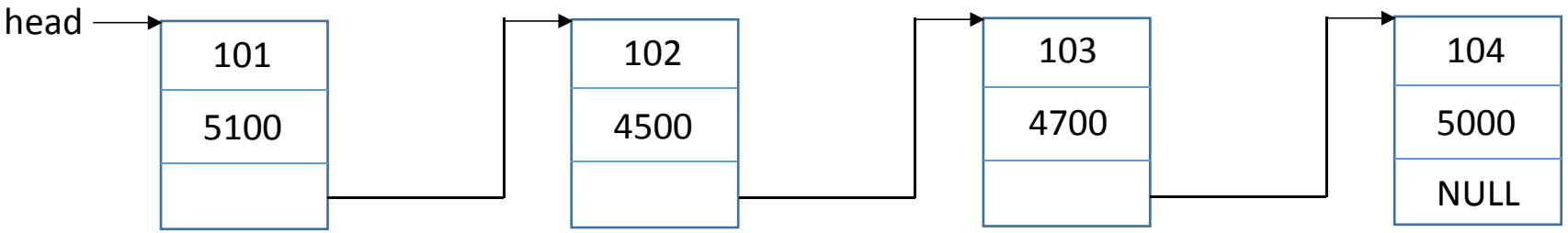
情况三： 节点插入不在表头位置

① p=q=head ;

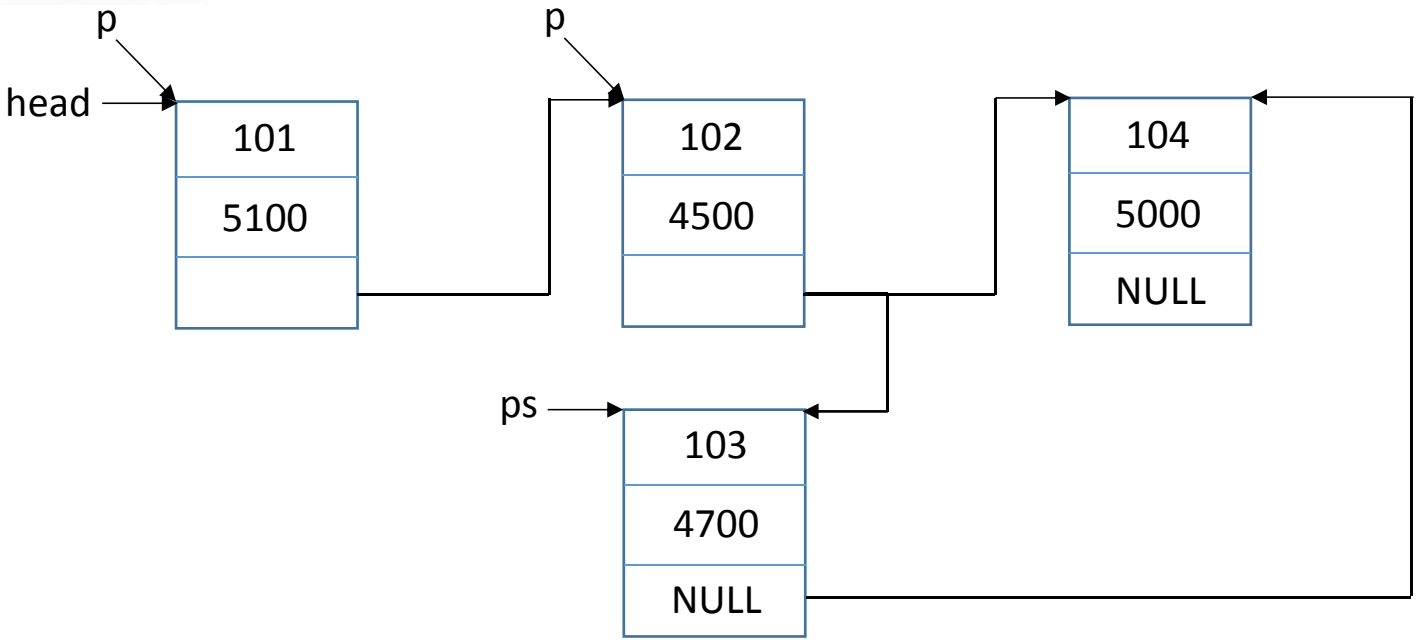
② 查找位置：p!=NULL && p->num<ps->num ;

③ 移动指针：q=p ； p=p->next ;

④ 节点插入：q->next=ps ； ps->next=p ;



如何只用一个指针实现节点的插入？



① p=head ;

② 如何查找位置？ p->next!=NULL && p->next->num<ps->num ;

③ 移动指针：p=p->next ;

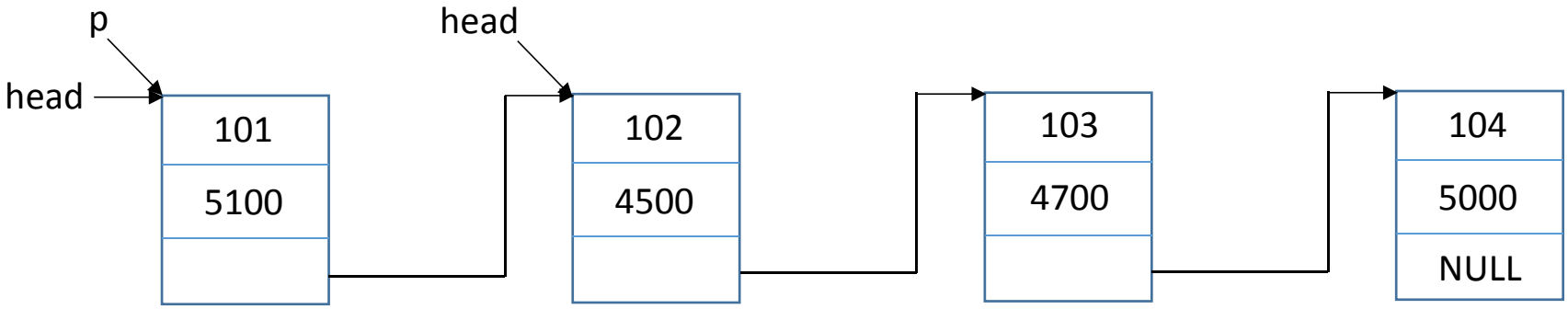
④ 如何插入节点： ps->next=p->next ; p->next=ps ;

链表结点的插入和删除

链表结点的删除

```
struct person *ListDelete(struct person *head,int num)
                    //删除职工号为num的结点
{
    struct person *q,*p;
    if(head->num==num)  //要删除的结点为链表首结点
    {
        p=head;
        head=p->next;
        free(p);
        return head;
    }
    q=p=head;
    while(p!=NULL&&p->num!=num) //查找要删除的结点
    {
        q=p; //用q指向刚访问过的结点
        p=p->next; //使p指向下一个结点
    }
    if(p!=NULL) //查到要删除的结点
    {
        q->next=p->next;  //将p指向结点的下一个结点连接到p->next
        free(p);  //删除p指向的结点
        return head;
    }
    printf("is not found!\n");
    return head;
}
```

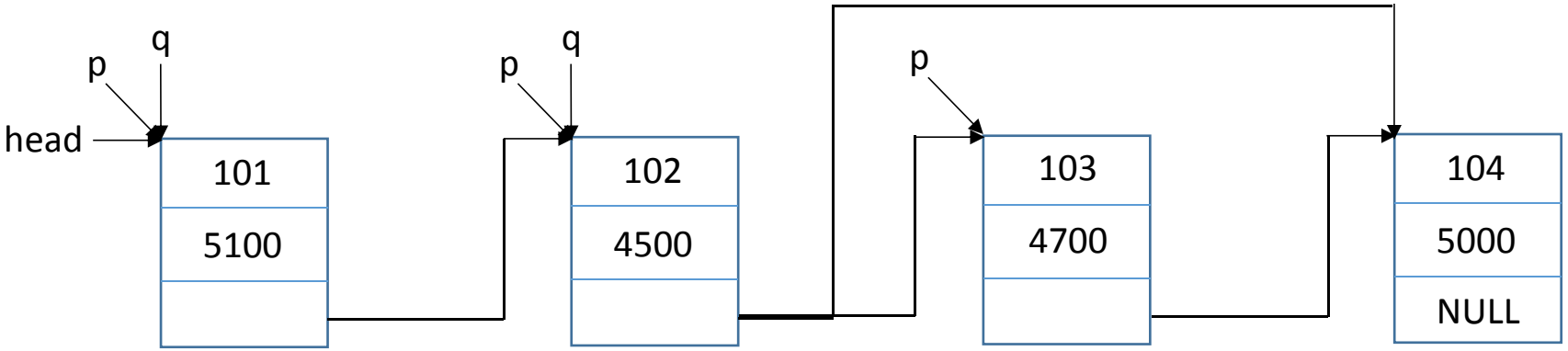
情况一：　要删除的节点为首节点　　num 101



条件判断：head->num等于num

① p=head ;
② head=p->next ;
③ free(p) ;

情况二：　要删除的节点不是首节点　　num 103



① p=q=head ;
② 查找位置：p!=NULL && p->num ! =num ;
③ 移动指针：q=p ;　p=p->next ;
④ 节点插入：q->next=p->next ;
⑤ free(p) ;

链表综合实例 — 职工工资链表

主函数

```c
#include<stdio.h>
#include<stdlib.h>
struct person{
    int num;
    float salary;
    struct person *next;
};
struct person *CreateList(void);
void PrintList(struct person *head);
struct person *ListDelete(struct person *head,int num);
struct person *ListInsert(struct person *head,struct person *ps);
void ListSearch(struct person *head,int num);
void main()
{
    int sn,num;
    struct person *head,ps;
    while(1)
    {
        printf("\t职工信息管理系统\n");
        printf("==============================\n");
        printf("\t1.职工信息链表建立\n");
        printf("\t2.职工链表节点的插入\n");
        printf("\t3.职工链表节点的删除\n");
        printf("\t4.职工链表节点的查询\n");
        printf("\t5.职工链表节点的输出\n");
        printf("\t0.退出\n");
        printf("==============================\n");
        printf("请选择菜单功能（0~5）:");
    while(1){
        scanf("%d",&sn);
        if(sn<0||sn>5)  printf("\n输入错误，请重新输入（0~5）:");
        else    break;
    }
    switch(sn){
        case 1:
            printf("职工信息链表建立\n");
            head=CreateList();
            break;
        case 2:
            printf("插入职工信息节点\n");
            printf("输入要插入的职工的工号:");
            scanf("%d",&ps.num);
            printf("输入要插入的职工的工资:");
            scanf("%f",&ps.salary);
            head=ListInsert(head,&ps);
            break;
        case 3:
            printf("职工节点的删除\n");
            printf("输入要删除的职工的工号:");
            scanf("%d",&num);
            head=ListDelete(head,num);
            break;
        case 4:
            printf("职工节点的查询\n");
            printf("输入要查找的职工的工号:");
            scanf("%d",&num);
            ListSearch(head,num);
            break;
        case 5:
            printf("职工信息链表的输出\n");
            PrintList(head);
            break;
        case 0:
            printf("再见！\n");
            exit(0);
    }
    }
}
```

生成链表函数
（尾插法）

```
struct person *CreateList(void)
{
    struct person *head;
    struct person *rear;
    struct person *p;
    int no;
    head=NULL;
    printf("\n输入一个职工号，输入0结束::");
    scanf("%d",&no);
    while(no!=0)
    {
        p=(struct person *)malloc(sizeof(struct person));
        if(p==NULL) {printf("error!");exit(0);}
        p->num=no;
        printf("\n输入一个职工工资:");
        scanf("%f",&p->salary);
        if(head==NULL)head=p;
        else rear->next=p;
        rear=p;
        printf("\n输入一个职工号，输入0结束:");
        scanf("%d",&no);
    }
    if(rear!=NULL)
        rear->next=NULL;
    printf("\n建表结束！\n");
    return head;
}
```

插入函数

```
struct person *ListInsert(struct person *head,struct person *ps)
//head为链表头指针，ps指向要插入的结点，由调用函数生成好
{
    struct person *p,*q;
    if(head==NULL) //若为空表，使head直接指向插入结点皆可
    {
        head=ps;
        printf("%d\n",head->num);
        return head;
    }
    if(head->num>ps->num) //结点插入在表头
    {
        ps->next=head;
        head=ps;
        return head;
    }
    p=q=head; //使p，q指向表头结点
    while(p!=NULL&&p->num<ps->num) //查找插入位置
    {
        q=p;
        p=p->next; //使q指向p所指向的结点的前一个结点
    }
    q->next=ps; //插入新结点
    ps->next=p;
    return head;
}
```

删除函数

```
struct person *ListDelete(struct person *head,int num) //删除职工号为num的结点
{
    struct person *q,*p;
    if(head->num==num)                                  //要删除的结点为链表首结点
    {
        p=head;
        head=p->next;
        free(p);
        return head;
    }
    q=p=head;
    while(p!=NULL&&p->num!=num)                         //查找要删除的结点
    {
        q=p;                                            //用q指向刚访问过的结点
        p=p->next;                                      //使p指向下一个结点
    }
    if(p!=NULL)                                         //查到要删除的结点
    {
        q->next=p->next;                                //将p指向结点的下一个结点连接到p->next
        free(p);                                        //删除p指向的结点
        return head;
    }
    printf("没有该职工!\n");
    return head;
}
```

查找函数

```
void ListSearch(struct person *head,int num)
{
    struct person *p=head;
    while(p!=NULL)
    {
        if(p->num==num) printf("节点找到: %d %f\n",p->num,p->salary);
        p=p->next;
    }
    printf("节点未找到\n");
}
```

遍历输出函数

```
void PrintList(struct person *head)
{
    struct person *p=head;
    while(p!=NULL)
    {
        printf("%d %f\n",p->num,p->salary);
        p=p->next;
    }
}
```

现有程序存在的问题 BUG？

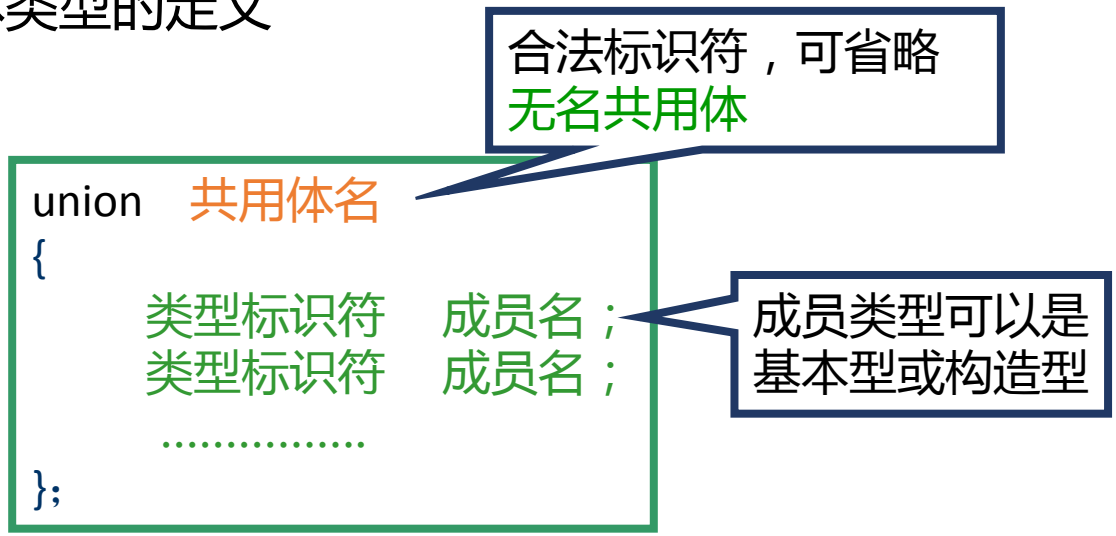
- 初始定义head指针时初始化为NULL
- 插入节点时使用malloc函数申请空间建立待插入节点
- 删除节点时加入对空链表的处理

- 节点信息逐个输入
- 增加清屏选项、输出格式显示形式的调整

9.5 共用体和枚举类型

共用体：构造数据类型，也叫联合体

- 用途：使几个不同类型的变量共占同一段内存（相互覆盖）
- 共用体类型的定义



共用体变量的定义

- 先定义共用体类型，再定义共用体变量

```
union  共用体名
{
      类型标识符    成员名；
      类型标识符    成员名；
      ……
};
union  共用体名  变量名表列；
```

```
union  ifcd{
    int  i;
    float  f;
    char  c;
    double  d;
};
union ifcd x1,y[5];
```

```
union  ifcd{
    int  i;
    float  f;
    char  c;
    double  d;
}x1,y[5];
```

```
union{
    int  i;
    float  f;
    char  c;
    double  d;
}x1,y[5];
```

共用体变量引用

- 引用方式： 共用体变量名.成员名

  共用体指针名->成员名

  (*共用体指针名).成员名

- 引用规则
  - 不能引用共用体变量，只能引用其成员
  - 共用体变量中起作用的成员是最后一次存放的成员
  - 不能在定义共用体变量时初始化
  - 可以用一个共用体变量为另一个变量赋值

```
例 float  x;
   union
    {   int i;   char ch;  float f;
    }a,b;
     a.i=1;   a.ch='a';   a.f=1.5;
     b=a;   ✓
     x=a.f;   ✓
```

```
例 union
   {    int i;
        char ch;
        float f;
   }a;
   a=1;        ×
```

```
例 a.i=1;
     a.ch='a';
     a.f=1.5;
     printf("%d",a.i);   ×
编译通过，运行结果不对
```

```
例  union
    {  int i;
       char ch;
       float f;
     }a={1,'a',1.5};      ×
```

例 结构体中嵌套共用体

```
struct
{   int num;
    char name[10];
    char sex;
    char job;
    union
    {  int class;
       char position[10];
    }category;
}person[2];
```

name	num	sex	job	class / position
Li	1075	F	S	501
Wang	537	M	T	prof

例 将整形数按字节输出

```
#include<stdio.h>
main()
{
    union  int_char
   {   int i;
       char ch[4];
   }x;
   x.i=65540;
   printf("x.1=%d,x.2=%d,x.3=%d,x.4=%d\n",x.ch[0],x.ch[1],x.ch[2],x.ch[3]);
}
```

枚举类型

- 基本数据类型,非构造数据类型
- 用途：列举出所有可能的取值，说明为该枚举类型的变量取值不能超过定义的范围
- 定义形式：enum 枚举名{ 枚举常量表 } ;

例　enum weekday{sun,mou,wed,thu,fri,sat};