第10章 复杂数据类型

—结构体、共用体、枚举、链表、声明新类型名

§1 结构体的概念

eg1:学生简历数据

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

eg2: 工资数据

序号	系	姓名	基本工资	津贴	应发	扣除	实发

1. 结构体(struct)是由若干不同类型数据所形成的组合项，其每个数据项称为“结构成员”或“结构分量”

2. 不能使用结构体，而应使用结构体类型来定义结构体变量

§1 结构体的概念

eg1:学生简历数据

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

```
struct  student
{  int num;
   char name[20];
   char sex;
   int age;
   float score;
   char addr[30]; };
 struct  student  stu1,stu2;
```

1. 结构体(struct)是由若干不同类型数据所形成的组合项，其每个数据项称为“结构成员”或“结构分量”

2. 不能使用结构体，而应使用结构体类型来定义结构体变量

§2 结构体变量的定义

语法I：先定义结构体类型，再定义结构体变量

```
struct 结构体名
{ 结构体成员定义;
  ……
  结构体成员定义;
};
struct 结构体名 结构变量1,…结构变量n;
```

语法II：定义结构体类型的同时定义结构体变量

```
struct 结构体名
{ 结构体成员定义;
  ……
  结构体成员定义;
}结构变量1,…结构变量n;
```

语法III：省略结构体名，直接定义结构体变量

```
struct
{ 结构体成员定义;
  ……
  结构体成员定义;
}结构变量1,…结构变量n;
```

§2 结构体变量的定义

eg1:学生简历数据

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

```
struct  student
{ int num;
   char name[20];
   char sex;
   int age;
   int score;
   char addr[30];
 };
struct  student  stu1,stu2;
```

```
struct  student
{ int num;
   char name[20];
   char sex;
   int age;
   float score;
   char addr[30];
 } stu1,stu2;
```

```
struct
{ int num;
   char name[20];
   char sex;
   int age;
   float score;
   char addr[30];
 } stu1,stu2;
```

§2 结构体变量的定义

eg2: 工资数据

序号	系	姓名	基本工资	津贴	应发	扣除	实发

```
struct salary
{   int  num;
    char  dep[30];
    char  name[20];
    float  base;
    float  other;
    float total;
    float cost;
    float real; };
struct  salary  w, z;
```

```
struct salary
{   int  num;
    char  dep[30];
    char  name[20];
    float base;
    float other;
    float total;
    float cost;
    float real; } w, z;
```

```
struct
{   int  num;
    char  dep[30];
    char  name[20];
    float base;
    float other;
    float total;
    float cost;
    float real; } w, z;
```

§3 结构体变量的引用

1.不能整体引用结构体变量之值，只能引用其成员之值：

结构体变量.成员

eg1: scanf(“%d%s%c%d%f%s”, &stu1);　　错误

eg1: scanf(“%d%s%c%d%f%s”, &stu1.num,stu1.name, &stu1.sex,&stu1.age,&stu1.score,stu1.addr);　　正确

eg2: printf(“%d%s%c%d%f%s\n”, stu1);　　错误

eg2: printf(“%d%s%c%d%f%s”, stu1.num,stu1.name, stu1.sex, stu1.age, stu1.score,stu1.addr);　　正确

eg1:学生简历 变量stu1

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

§3 结构体变量的引用

1.不能整体引用结构体变量之值，只能引用其成员之值：

结构体变量.成员

eg3: struct student stu1;
stu1={1001,”ZhangMing”,’M’,18,90,”Shanghai”};

错误

```
eg3: struct student stu1;
stu1.num=1001;
strcpy(stu1.name,” ZhangMing”);
 stu1.sex=‘M’;
stu1.age=18;
stu1.score=90;
strcpy(stu1.addr,” Shanghai”);
```

正确

eg1:学生简历 变量stu1

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

§3 结构体变量的引用

1.不能整体引用结构体变量之值，只能引用其成员之值：

结构体变量.成员

2. 结构体变量不可整体赋值，同类型结构体变量可以相互赋值

3.可引用结构体变量的地址，也可引用结构成员的地址

```
eg3: struct student stu1;
stu1.num=1001;
strcpy(stu1.name,” ZhangMing”);
 stu1.sex=‘M’;
stu1.age=18;
stu1.score=90;
strcpy(stu1.addr,” Shanghai”);
```

正确

```
eg4: struct student stu2;
stu2=stu1;
```

等价

```
st2.num=stu1.num;
strcpy(stu2.name,stu1.name);
…
```

eg1:学生简历 变量stu1

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

§3 结构体变量的引用

1.不能整体引用结构体变量之值，只能引用其成员之值：

结构体变量.成员

2. 结构体变量不可整体赋值，同类型结构体变量可以相互赋值

3.可引用结构体变量的地址，也可引用结构成员的地址

eg5: printf("%o",&stu1);/*struct student型*/

eg6: printf("%o",&stu1.num); /*int型*/

不同类的地址

eg1:学生简历 变量stu1

学号	姓名	性别	年龄	成绩	地址
num	name[20]	sex	age	score	addr[30]
int	char	char	int	float	char

§3 结构体变量的引用

1.不能整体引用结构体变量之值，只能引用其成员之值：

结构体变量.成员

2. 结构体变量不可整体赋值，同类型结构体变量可以相互赋值

3.可引用结构体变量的地址，也可引用结构成员的地址

4.若结构成员仍为结构体变量，应使用成员运算符.逐级找到最低一级成员进行操作

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

```
struct student
{ int num;  char name[20];
  char sex;   int age;
   struct date birthday;
   char addr[30];
 }s1,s2;
```

num	name	sex	age	birthday			addr
				month	day	year	

§3 结构体变量的引用

例1：以下程序的输出结果是 A 。

```
#include <stdio.h>
main( )
{ struct date
   { int year,month,day;
   }today;
printf("%d\n",sizeof(struct date));
}
```

A. 6　　B. 8　　C. 10　　D. 12

§3 结构体变量的引用

例2：已知学生记录描述为：

```
struct student
 { int no;  char  name[20];  char sex;
   struct { int year;
            int month;
            int day;
          }birth;
  } s;
```

设变量s中的生日应是“1984年11月11日”，下列对”生日”的正确赋值方式是 D 。

A. year=1984; month=11; day=11;

B. birth.year=1984; birth.month=11; birth.day=11;

C. s.year=1984; s.month=11; s.day=11;

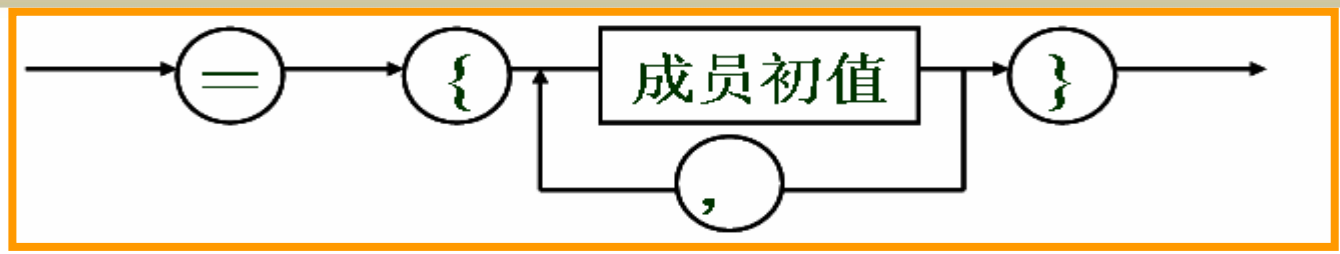
D. s.birth.year=1984; s.birth.month=11;s.birth.day=11;

§3 结构体变量的引用

例3：计算并输出学生5门课（整型）的平均成绩，最高分和最低分。学生的整个成绩信息用结构体变量表示。

```
#include <stdio.h>
struct score
{ int grade[5];
  float aver,max,min;
};
void main()
{ int i;
  struct score m;
  printf("请输入5门课成绩:\n");
  for(i=0;i<5;i++)
   scanf("%d",&m.grade[i]);
m.aver=0;
m.max=m.min=m.grade[0];
for(i=0;i<5;i++)
{ m.aver+=m.grade[i];
  if(m.grade[i]>m.max)
    m.max=m.grade[i];
  if(m.grade[i]<m.min)
    m.min=m.grade[i]); }
 m.aver/=5;
printf("%.1f,%.1f,%.1f\n",
 m.aver,m.max,m.min);  }
```

§4 结构体变量的初始化



例1：结构体变量a的初始化示例

```
main( )
{ struct student
  { long int num;
    char name[20];
    char sex;
    char addr[20];
  } a={89031, ”Li Lin”, ’M’, ”123 Beijing Road” };
  printf(“NO:%ld  name:%s  sex:%c  address:%s\n”,
              a.num, a.name, a.sex, a.addr);  }
```

num	89031
name	Li Lin
sex	M
addr	123 Beijing Road

NO:89031 name:Li Lin sex:M address:123 Beijing Road

§4 结构体变量的初始化

例2:分析下例结构体变量的存储布局,指出程序运行结果

```
main( )
{ struct st1{  char c[4];
               char *s;
             } s1={“abc”,”def”};
  struct st2{ char *cp;
              struct st1 ss1;
             }s2={“ghi”, {“jkl”, “mno”}};
  printf(“__1__%c*%c\n”, s1.c[0], *s1.s);
  printf(“__2__%s*%s\n”, s1.c, s1.s);
  printf(“__3__%s*%s\n”, s2.cp, s2.ss1.s);
  printf(“__4__%s*%s\n”, ++s2.cp, ++s2.ss1.s);
  printf(“__5__%s*%s\n”, s2.cp, s2.ss1.s);
  printf(“__6__%s*%s\n”, --s2.cp, --s2.ss1.s);}
```

__1__a*d

__2__abc*def

__3__ghi*mno

__4__hi*no

__5__hi*no

__6__ghi*mno

§4 结构体变量的初始化

例3:编程判定二维平面中的三点能否构成三角形

```
#include <stdio.h>
#include <math.h>
struct point
{ float x;
  float y;
};
 float length(float x1,float y1,float x2,float y2)
{  float len;
   len=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
   return len;
}
void main()
{   struct point p1,p2,p3;
    float len1,len2,len3;
    printf("请分别输入三点坐标\n");
    scanf("%f,%f",&p1.x,&p1.y);
    scanf("%f,%f",&p2.x,&p2.y);
    scanf("%f,%f",&p3.x,&p3.y);
    len1=length(p1.x,p1.y,p2.x,p2.y);
    len2=length(p2.x,p2.y,p3.x,p3.y);
    len3=length(p3.x,p3.y,p1.x,p1.y);
    if(len1+len2>len3&&len2+len3
    >len1&&len1+len3>len2)
     printf("三点可以构成三角形
    \n");
    else
     printf("三点不能构成三角形
    \n");
    }
```

§4 结构体变量的初始化

例3:编程判定二维平面中的三点能否构成三角形

```
#include <stdio.h>
#include <math.h>
struct point
{ float x;
  float y;
};
 float length(struct point a,struct point b)
{ float len;
  len=sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
  return len;
}
void main()
{   struct point p1,p2,p3;
    float len1,len2,len3;
    printf("请分别输入三点坐标\n");
    scanf("%f,%f",&p1.x,&p1.y);
    scanf("%f,%f",&p2.x,&p2.y);
    scanf("%f,%f",&p3.x,&p3.y);
    len1=length(p1,p2);
    len2=length(p2,p3);
    len3=length(p3,p1);
    if(len1+len2>len3&&len2+
    len3>len1&&len1+len3>len2)
    printf("三点可以构成三角形
    \n");
    else
    printf("三点不能构成三角形
    \n");
    }
```

§5 结构体数组

定义方法

- 先定义结构类型，再定义结构数组
- 定义结构类型的同时定义结构数组
- 缺省结构体名，直接定义结构数组

```
struct  name
{成员1定义;
 ……
 成员m定义;
};
struct name a[n];
```

```
struct  name
{element1定义;
 ……
 element m定义;
} a[n];
```

```
struct
{element 1定义;
 ……
 element m定义;
} a[n];
```

§5 结构体数组

数组初始化 ⇒ ={{ },{ },{ },……};

	num	name	sex	age	score	addr
stu[0]	10101	Li Lin	M	18	87.5	103 Beijing Road
stu[1]	10102	Zhang Fun	M	19	99	130 Shanghai Road
stu[2]	10103	Wang Min	F	20	78.5	1010 Zhongshan Road

```
struct  student
{ int num;
  char name[20];
  char sex;
  int age;
  float score;
  char addr[30];};
struct student stu[3];
```

```
struct  student
{ int num;
  char name[20];
  char sex;
  int age;
  float score;
  char addr[30];
}stu[3];
```

```
struct
{ int num;
  char name[20];
  char sex;
  int age;
  float score;
  char addr[30];
}stu[3];
```

§5 结构体数组

数组初始化 ➡ ={{ },{ },{ },……};

	num	name	sex	age	score	addr
stu[0]	10101	Li Lin	M	18	87.5	103 Beijing Road
stu[1]	10102	Zhang Fun	M	19	99	130 Shanghai Road
stu[2]	10103	Wang Min	F	20	78.5	1010 Zhongshan Road

```
struct  student
{ int num;
  char name[20];  char sex;
  int age;
  float score;
  char addr[30];
} stu[3]={{10101,"Li Lin",'M',18,87.5,"103 Beijing Road"} ,
          {10102,"Zhang Fun",'M',19,99, "130 Shanghai Road"},
{10103,"Wang Min",  'F',20,78.5,"1010 Zhongshan Road"}};
```

§5 结构体数组

数组初始化 ➡ ={{ },{ },{ },……};

	num	name	sex	age	score	addr
stu[0]	10101	Li Lin	M	18	87.5	103 Beijing Road
stu[1]	10102	Zhang Fun	M	19	99	130 Shanghai Road
stu[2]	10103	Wang Min	F	20	78.5	1010 Zhongshan Road

```
main( )
{ int i;
  printf("NO.\t name\tsex\tage\tscore\taddr\n");
  for(i=0;i<=2;i++)
  printf("%d\t%s%c%d%.2f%s\n",
                          stu[i].num,stu[i].name,stu[i].sex,
                          stu[i].age,stu[i].score,stu[i].addr);
}
```

§5 结构体数组

例2：根据以下定义，能打印出字母M的语句是 D 。

```
struct person { char name[9];
                int age;
              };
struct person class[10]={“John”,17,
                         “Paul”,19,
                         “Mary”,18,
                         “Adam”,16};
```

A. printf(“%c\n”,class[3].name);

B. printf(“%c\n”,class[3].name[1]);

C. printf(“%c\n”,class[2].name[1]);

D. printf(“%c\n”,class[2].name[0]);

§5 结构体数组

例3：有Li、Zhang、Wang三位候选人，编程根据选举情况，统计出各人得票结果。

```
#include <string.h>
struct person
 { char name[10];
   int count;
  }leader[3]={{“Li”,0},{“Zhang”,0},{“Wang”,0}};
main( )
{ int i, j, n;
   char leader_name[10];
   scanf(“%d”, &n);
```

§5 结构体数组

例3：有Li、Zhang、Wang三位候选人，编程根据选举情况，统计出各人得票结果。

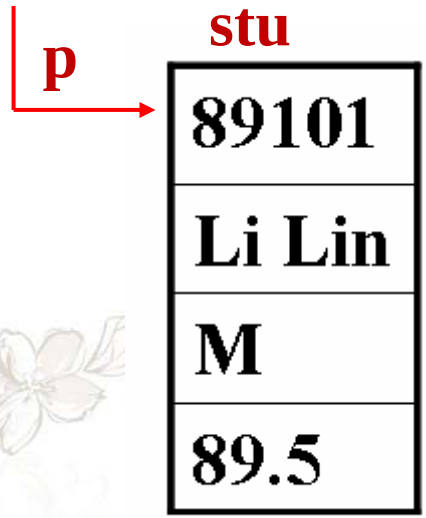
```
for( i=1; i<=n; i++)    /*唱票循环*/
 { scanf("%s", leader_name);  /*唱票*/
   for( j=0; j<=2; j++)
     if(strcmp(leader_name, leader[j].name)==0)
       leader[j].count++;   /*计票*/
 }
for( i=0; i<=2; i++)
   printf("%s:%d\n",leader[i].name,leader[i].count);
}
```

§6 结构体与指针变量

一、指向结构体变量的指针变量

struct 结构体名 *指针变量名;

```
struct student
   { long num;
     char name[20];
     char sex;
     float score;};
  struct student stu;
  struct student *p=&stu;
```



§6 结构体与指针变量

二、对结构体变量的访问

采用成员运算符：结构体变量名.成员名

访问方法

stu变量输出示例1：

```
#include <string.h>
main( )
{ struct student
   { long num;
     char name[20];
     char sex;
     float score;} stu;
```

```
stu.num=89101;
strcpy(stu.name, “Li Lin”);
stu.sex=‘M’;
stu.score=89.5;
printf(“%ld\n %s\n %c\n %f\n”,
            stu.num, stu.name,
            stu.sex, stu.score);
}
```

§6 结构体与指针变量

二、对结构体变量的访问

访问方法

- 采用成员运算符：结构体变量名.成员名
- 采用间访符：(*p).成员名
- 采用指向运算符：p->成员名

stu变量输出示例2：

```
#include <string.h>
main( )
{ struct student
   { long num;
     char name[20];
     char sex;
     float score;} stu,*p;
  stu.num=89101;
```

```
strcpy(stu.name, “Li Lin”);
stu.sex=‘M’;
stu.score=89.5;
p=&stu;
printf(“%ld\n %s\n %c\n %f\n”,
          (*p).num, (*p).name,
          (*p).sex, (*p).score);
}
```

§6 结构体与指针变量

二、对结构体变量的访问

访问方法

- 采用成员运算符：结构体变量名.成员名
- 采用间访符：(*p).成员名
- 采用指向运算符：p->成员名

stu变量输出示例3：

```
#include <string.h>
main( )
{ struct student
   { long num;
     char name[20];
     char sex;
     float score;} stu,*p;
  stu.num=89101;
strcpy(stu.name, “Li Lin”);
stu.sex=‘M’;
stu.score=89.5;
p=&stu;
printf(“%ld\n %s\n %c\n %f\n”,
          p->num, p->name,
          p->sex, p->score);
}
```

§6 结构体与指针变量

通式1：

若有： struct name a,*p=&a;

则 a.element

≡(*p).element

≡ p->element

≡(*(&a)).element

≡(&a)->element

§6 结构体与指针变量

例1：下面四个运算符中，优先级最低的是 D 。

A. ( ) B. . C. -> D. ++

例2(考题): 已知有如下的结构类型定义和变量声明：

```
struct student
{  int num;
   char name[10];
}stu={1,"Mary"},*p=&stu;
```

则下列语句中错误的是 C 。

A. printf("%d",stu.num); B. printf("%d",(&stu)->num);

C. printf("%d",&stu->num); D. printf("%d",p->num);

§6 结构体与指针变量

例3： 设有如下定义：

```
struct sk
 { int n;
   float x;
  }data,*p;
```

若要使p指向data中的n域，正确的赋值语句是 C 。

A. p=&data.n;　　B. *p=data.n;

C.p=(struct sk *)&data.n;　　D. p=(struct sk *)data.n;

§6 结构体与指针变量

三、指向结构体数组元素的指针变量

通式2：

若有：struct name a[n], *p=a;

则：p+1 ≡a+1 ≡ &a[1]→+sizeof(struct name)

p+i ≡a+i ≡ &a[i] →+i*sizeof(struct name)

通式3：

若有：struct name a[n], *p=a;

则: (++p)->element → a[1].element,p=&a[1]

++p->element → ++a[0].element,p=&a[0]

(p++)->element → a[0].element,p=&a[1]

p++->element → a[0].element,p=&a[1]

§6 结构体与指针变量

三、指向结构体数组元素的指针变量

通式4：

若有：struct name a[n], *p;

当：p=&a[i]时

有：a[i].element ≡(*p).element ≡p->element

≡(*(&a[i])).element ≡(&a[i])->element

例1(考题): 已知数据类型定义和变量声明如下：

```
struct sk
{ int a; float b;}data[2],*p=data;
```

则以下对data[0]中成员a的引用中错误的是__A__。

A. data[0]->a　　B. data->a　　C. p->a　　D. (*p).a

§6 结构体与指针变量

例2（考题）以下程序输出的两个数是 2 和 5 。

```
struct ks {
      int a;
      int *b;};
main( )
{  struct ks s[4], *p;
    int n=1, i;
    for( i=0; i<4; i++)
    {  s[i].a=n;
       s[i].b=&(s[i].a);
       n+=2; }
       p=&s[0];
       printf( “%d, %d\n”, ++(*p->b), *s[2].b); }
```

§6 结构体与指针变量

例3：若有以下程序段：

```
struct dent
  { int n,
    int *m;
   };
  int a=1,b=2,c=3;
  struct dent s[3]={{101,&a},{102,&b},{103,&c}};
  struct dent *p=s;
```

则以下表达式中值为2的是 _____。

A. (p++)->m　　B. *(p++)->m

C. (*p).m　　D. *(++p)->m

D

§6 结构体与指针变量

例4(考题): 若main函数中有以下定义、声明和语句:

```
struct test
{  int a; char *b;};
char x0[ ]="United states",x1[ ]="England";
struct test x[2],*p=x;
x[0].a=300; x[0].b=x0;
x[1].a=400;x[1].b=x1;
```

则不能输出字符串"England"的语句是 C 。

A. puts(x[1].b);　　B. puts((x+1)->b);

C. puts((++x)->b)　　D. puts((++p)->b);

§6 结构体与指针变量

例5(考题): 以下程序的输出结果是 575 。

```
#include <stdio.h>
struct s
{ int a;
  struct s *next;
};
main( )
{ int i;
  static struct s x[2]={5,&x[1],7,&x[0]},*ptr;
  ptr=&x[0];
  for(i=0;i<3;i++)
  { printf("%d",ptr->a);  ptr=ptr->next;}
}
```

§6 结构体与指针变量

例6(考题)分析以下程序的输出结果

```
#include <stdio.h>
struct s { int n,*m;}*p,*q;
int d[5]={10,20,30,40,50};
struct s arr[5]={{100,&d[0]},{200,&d[1]},{300,&d[2]},
                 {400,&d[3]},{500,&d[4]}};
main( )
{ q=p=arr;
  printf("%d\n",++p->n); p++;
  printf("%d\n",p++->n);
  printf("%d\n",++(*p->m));
  q+=3;
  printf("%d\n",*q->m);}
```

101
200
31
40

§6 结构体与指针变量

四、结构体变量的指针作函数实参

例：有4个学生，各学生包含学号、姓名和成绩三个数据，编写max找出最高分的学生，交由main打印出来。

```
struct student
{ int num;
   char name[20];
   int score;
};
```

```
struct student
{ int num;
   char name[20];
   int score;
}stu[4];
```

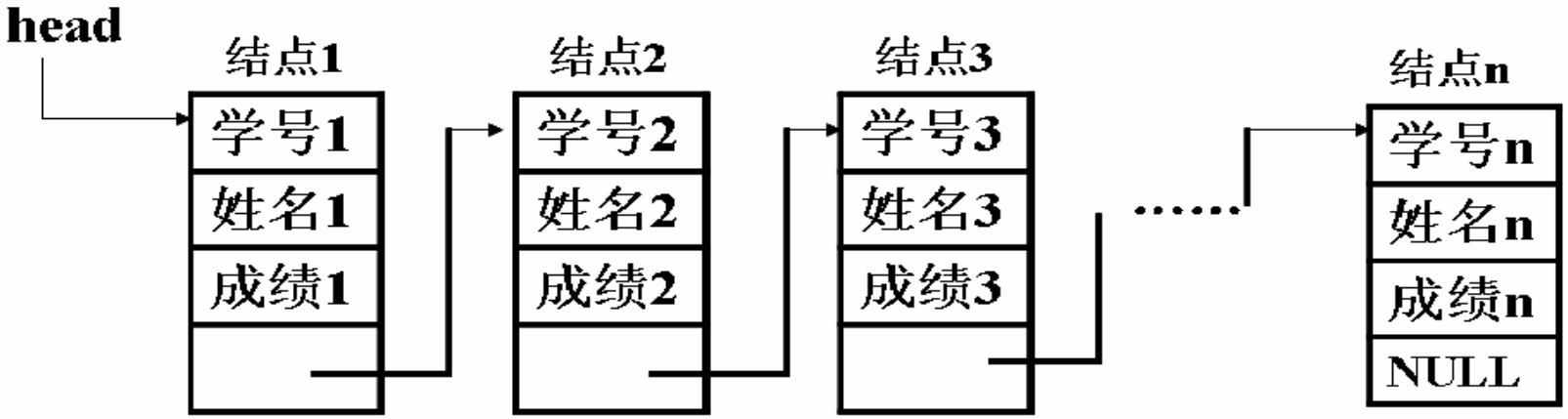
```
struct student *max(struct student *p,int n);
```

§6 结构体与指针变量

```
main( )
{ struct student stu[4], *pm;   int i;
  struct student *max(struct student *p, int n);
  for( i=0; i<4; i++)
  scanf("%d%s%d",&stu[i]. num, stu[i].name, &stu[i].score);
  pm=max( stu, 4);
  printf("\n The maximum score\n");
  printf("%d\t%s\t%d\n", pm->num, pm->name, pm->score);}
struct  student *max( struct student *p, int n )
{ struct student *pr, *p_end;   /*pr指向成绩最高的学生*/
  p_end=p+n-1;
  pr=p;              /*假设第一个学生成绩最高*/
  for( ; p<=p_end; p++)  if( p->score>pr->score)  pr=p;
  return pr;  }
```

§7 链表 (Linked List)

一、链表概述

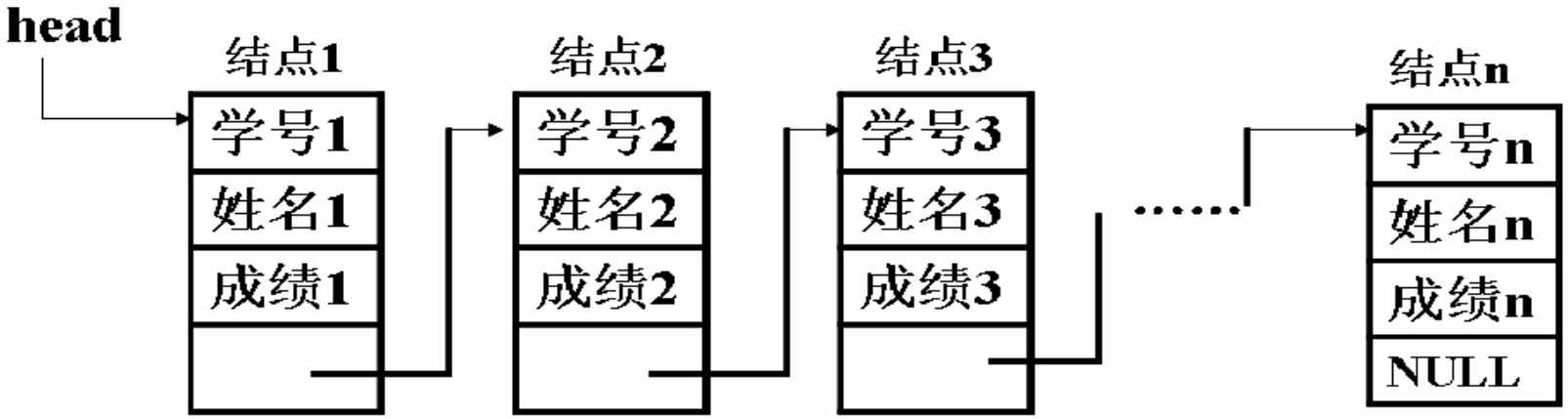


1.链表由若干结点用指针链接所组成。每个结点包含二部分内容：数据域——表示用户实际使用的数据；指针域——指向另一个结点的指针。由此通过指针将各结点链接起来形成链表，并由一头指针head指向首结点，而链尾指针为空NULL。

2. C中，链表各结点是同类型结构体变量，head是指向同类结构体的指针变量。NULL是定义在stdio.h中的符号常量（#define NULL 0)。

§7 链表 (Linked List)

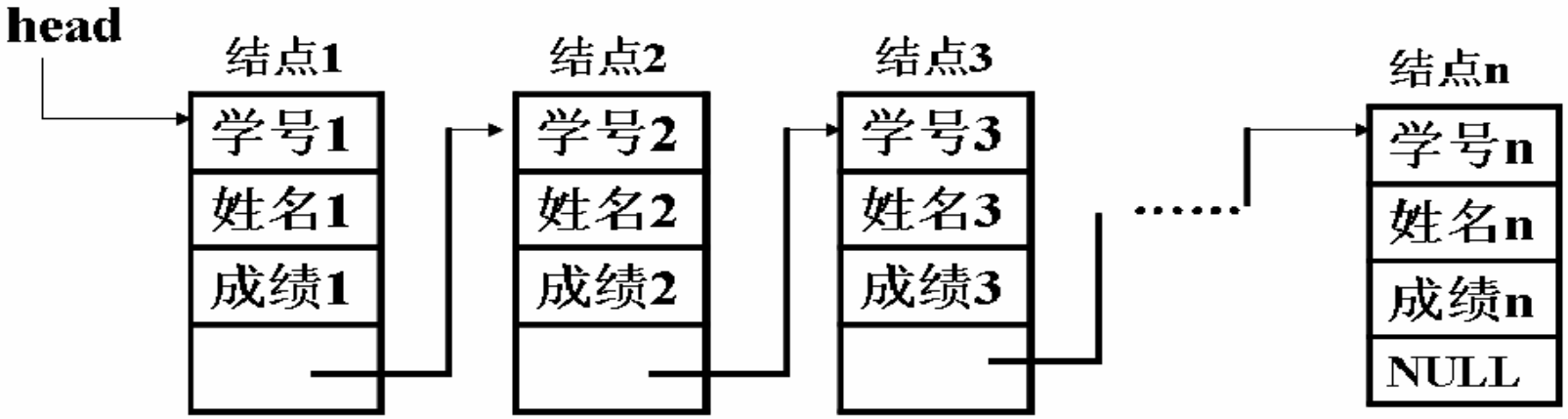
一、链表概述




```
struct  student
{ int num;
  char name[20];
   float score;
   struct student *next;  /*指向下一结点的指针*/
};
struct  student  *head;
```

§7 链表 (Linked List)

一、链表概述



eg1: head= =NULL； 表示什么 ？ /*空链表， n=0*/

eg2: 指针变量p=p->next的作用是什么 ？ /*p指针前移*/

eg3: 若指针p->next==NULL,则p指向链表什么位置 ？ /*尾结点*/

eg4: p=head;当p->next≠NULL时,p=p->next表示什么功能 ？

/*正向遍历链表*/

§7 链表 (Linked List)

二、动态存储分配函数

#include <stdlib.h>

1. malloc函数

- 函数原型： void *malloc(unsigned size);
- 功能： 在内存动态存储区分配一个长度为size 字节的空间
- 返回值： 若分配成功，返回分配区起始地址；若分配失败，返回空指针NULL

2. free函数: void free(void *p);

- 功能： 释放由指针p所指向的内存区

```
eg1: float *f;
      :
     f=( float *)malloc( sizeof(float));
     if( !f ) { printf("Allocation error!\n"); exit;}
    *f=1.38;
     printf("%.2f\n", *f); /*为实型数据动态分配主存单元*/
```

§7 链表 (Linked List)

二、动态存储分配函数

#include <stdlib.h>

1. malloc函数

- 函数原型：void *malloc(unsigned size);
- 功能：在内存动态存储区分配一个长度为size 字节的空间
- 返回值：若分配成功，返回分配区起始地址；若分配失败，返回空指针NULL

2. free函数: void free(void *p);

- 功能：释放由指针p所指向的内存区

```
eg2:  int *pi;
      pi=( int *)malloc( sizeof(int));
      if( !pi ) { printf("Allocation error!\n"); exit;}
     *pi=5;   printf("i=%d\n", *pi);
      free((void*)pi); /*释放pi所指对象的内存空间*/
```

§7 链表 (Linked List)

例2(考题):下列程序的功能是:首先输入二维数组的行数和列数,动态为该二维数组分配存储空间;其次,向二维数组中输入数据;最后,依次输出该数组中的所有元素。

```
main( )
{ int *p, i, j, k=1;
  int row,col;
  printf("Number of row:");
  scanf("%d", &row);
  printf("Number of column:");
  scanf("%d", &col);
  p=(int *)  (1)  ;
  if( p==NULL)
 {printf("Not memory!\n");exit(1);}
```

```
for( i=0; i<row; i++)
for( j=0; j<col; j++)
   p[  (2)  ]=k++;
for( i=0; i<row; i++) {
  for( j=0; j<col; j++)
    printf("%4d", p[  (3)  ]);
   printf("\n");}
 free(p);
}
```

(1)malloc(row*col*sizeof(int)) (2) i*col+j (3) i*col+j

§7 链表 (Linked List)

三、链表常用操作

1.建立链表

从无到有建立由n个结点组成的链表，其头指针由head指定，尾指针为NULL

2.输出（遍历）链表

将某时刻链表head中所有结点的数据依次输出出来

3.删除操作

从链表head中删除一个指定结点p

4.插入操作

向链表head中插入一个指定结点p

§7 链表 (Linked List)

四、建立链表

例1：已知学生数据包含学号、成绩，试建立由若干学生数据组成的单向链表。

学号 num	成绩 score	指针 next

图：学生数据结点图示

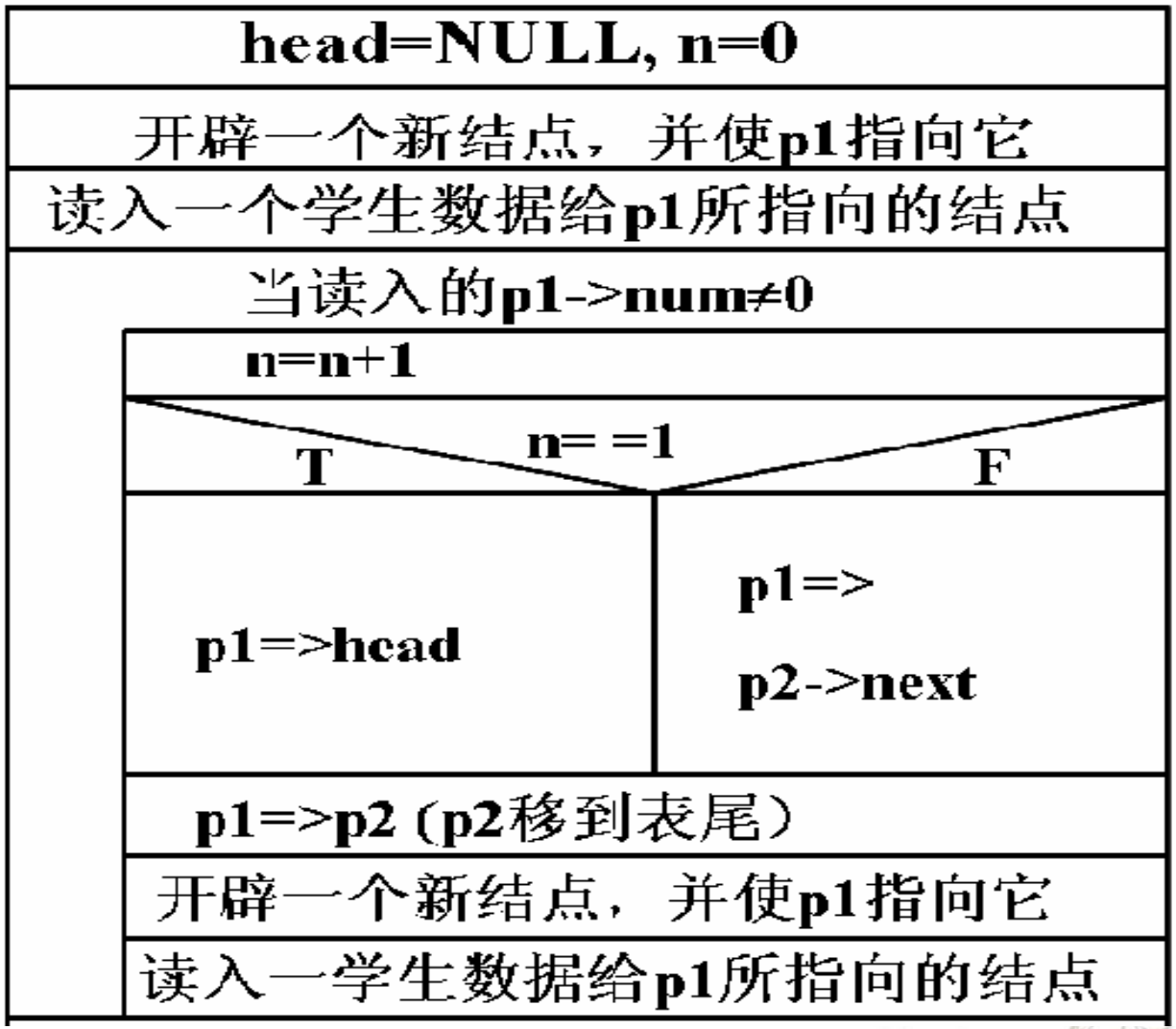
学生数据的结构体类型：

```
struct student {
        int num;
        float score;
        struct student *next;
        };
```

creat函数原型：

```
struct student *creat( );
```

§7 链表 (Linked List)



§7 链表 (Linked List)

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct student)
struct  student
{ int num;
   float score;
   struct student *next;
};
int n;        /*链表中结点数*/
struct student *creat( )
{ struct student *head,*p1,*p2;
   n=0;
   head=NULL;    /*初始链表为空链表*/
```

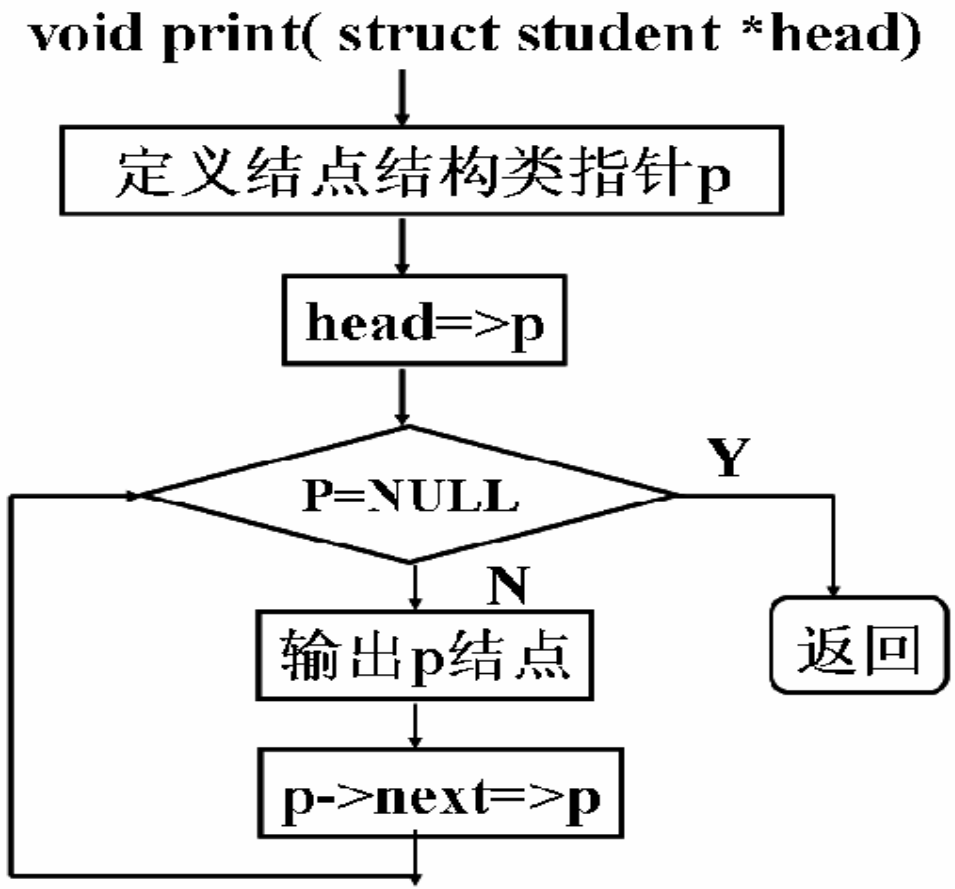
§7 链表 (Linked List)

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

§7 链表 (Linked List)

五、遍历链表

例2：有例1已经建立学生数据链表head，编程用遍历链表的方输出该链表中的若干学生数据。




```
void print(struct student *head );
```

§7 链表 (Linked List)

五、遍历链表

```
void print(struct student *head)
{ struct student *p;
  p=head;
  if(p==NULL) {printf("list is empty!\n"); exit(0);}
  printf("\n The %d nodes are:\n",n);
  while(p!=NULL)
  { printf("%d,%.1f\n",p->num,p->score);
     p=p->next; }
}
```

§7 链表 (Linked List)

考题实例：设已建立了一条链表，链表上结点的数据结构为：

```
struct node {
    float English,Math;    //英语和数学成绩
    struct node *next;
};
```

求出该链表上的结点个数、英语的总成绩和数学的总成绩，并在链首增加一个新结点，其分量English和Math分别存放这两门课程的平均成绩。若链为空链时，链首不增加结点。

以下函数ave( )的第一个参数h指向链首，第二个参数count存放求出的结点个数。

§7 链表 (Linked List)

```
struct  node *ave(struct node *h, int *count)
{  struct node *p1;
   float sume=0, summ=0;
   *count=0;
   if( h==NULL )     (1)     ;
   p1=h;
   while(    (2)    )
   { sume+=p1->English;   summ+=p1->Math; *count=*count+1;
      (3)    ; }
   p1=( struct node * )malloc(sizeof( struct node ));
   p1->English=sume/*count;  p1->Math=summ/*count;
    (4)    ;  h=p1; return h;
}
```

(1) return h

(2) p1!=NULL 或 p1

(3) p1=p1->next

(4) p1->next=h

§7 链表 (Linked List)

六、链表删除操作

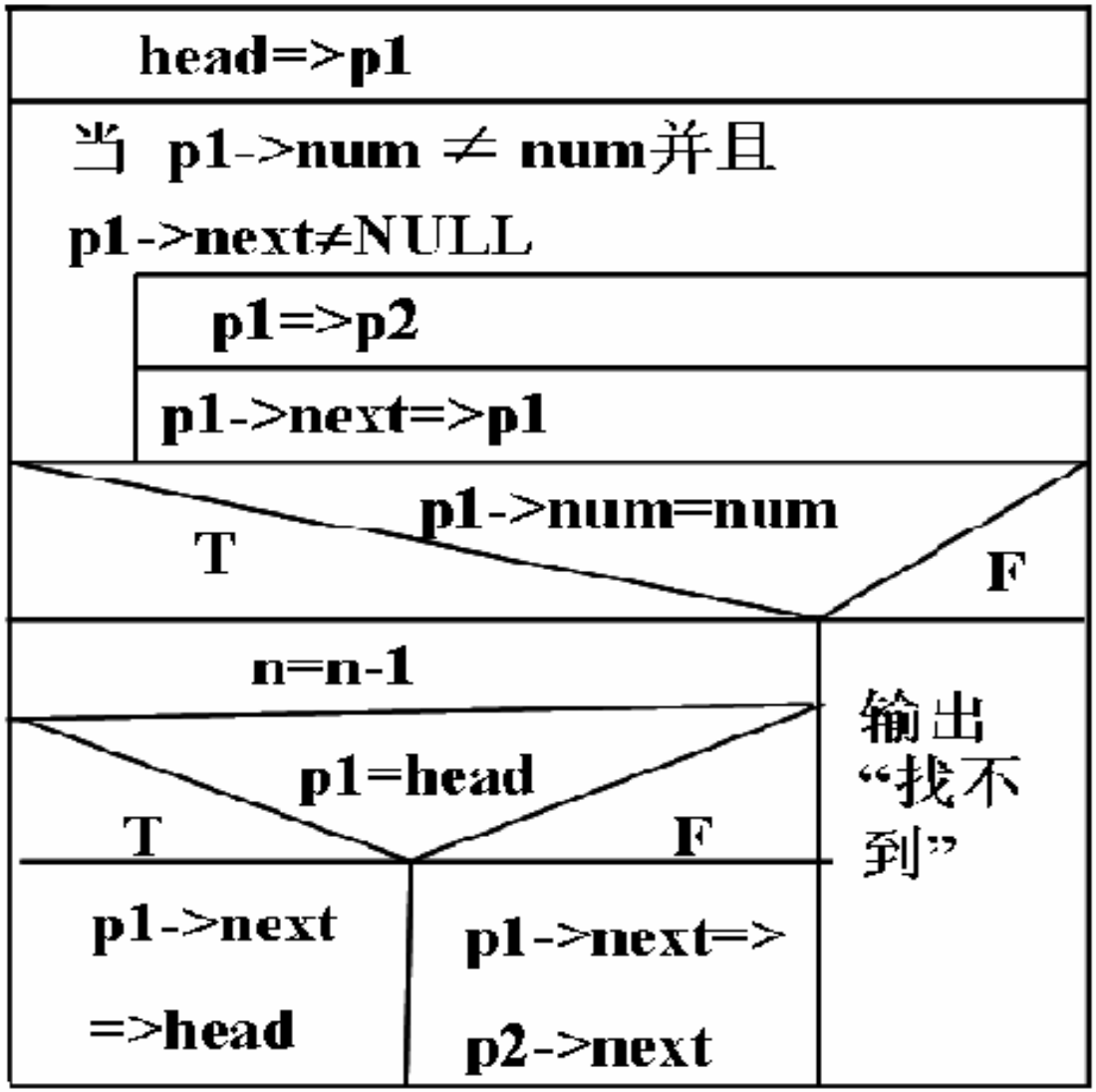
例3：从例1的学生链表head中删除指定学号为num的结点

```
struct student *del(struct student *head,int num);
```

删除操作逻辑步骤

- 查找欲删除结点在链表中的位置，并用p1指定
- 若找到p1结点，将p1结点删除

§7 链表 (Linked List)



§7 链表 (Linked List)

```
struct student *del( struct student *head, int num)
{ struct student *p1, *p2;
  p1=head;
  while(p1->num!=mun&&p1->next!=NULL)
   p2=p1,p1=p1->next;
 if(p1->num==num)     /*找到*/
   { n=n-1;
      if(p1==head)  head=p1->next; else p2->next=p1->next;
    }
else      /*未找到*/
   printf("%d not been found!\n",num);
 return(head);
}
```

§7 链表 (Linked List)

七、链表插入操作

例5：设有学生链表中各结点按学号由小到大顺序排列，现要将p0结点插入已有链表中。

插入操作步骤

- 确定p0的插入位置，并用p1指针指定
- 插入p0结点

```
struct student *insert(struct student *head,struct student *p0);
```

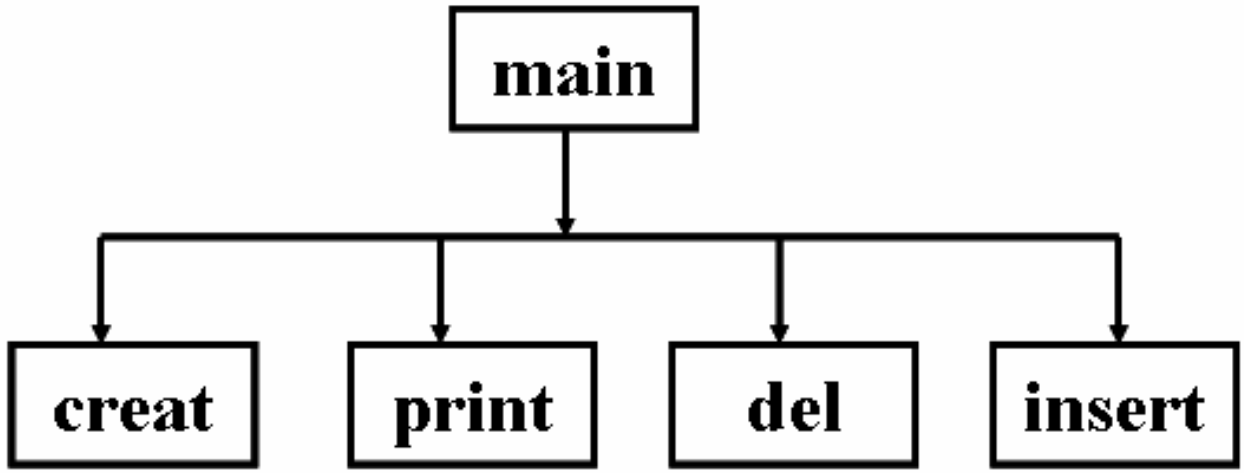
§7 链表 (Linked List)

```
struct student *insert( struct student *head, struct student *p0)
{ struct student *p1, *p2;
   n=n+1;  p1=head;
  while(p1->num<p0->num&&p1->next!=NULL)
              { p2=p1;  p1=p1->next;}        /*确定插入位置*/
  if( p1->num>p0->num)          /*在链首或链中插入*/
            { if(head==p1) head=p0; else p2->next=p0;
              p0->next=p1;
            }
 else  /*在链尾插入*/
 { p1->next=p0;  p0->next=NULL;}
  return( head);
}
```

§7 链表 (Linked List)

八、链表操作演示

例5：使用模块化设计思想，将链表建立、遍历、插入和删除功能集成为一个演示系统，演示链表的常用操作过程。



图：模块调用图

§7 链表 (Linked List)

```
#include <stdlib.h>
#include <stdio.h>
#define LEN sizeof(struct student)
struct student
{ int num;  float score;
   struct student *next;
 };
int n; /*链表中结点数*/
main( )
{ struct student *creat( );
   void print( struct student *head);
   struct student *del( struct student *head, int num);
   struct student *insert(struct student *head,struct student *p0);
```

§7 链表 (Linked List)

```
struct student *head, *p0;     /*头指针和插入结点指针*/
int  num;                      /*欲删除结点学号*/
head=creat( );
print(head);
printf("Input the deleted number:");
scanf("%d", &num);
head=del( head, num);
print(head);
printf("Input the inserted node:");
p0=(struct student *)malloc(LEN);
scanf("%d%f", &p0->num, &p0->score);
head=insert(head,p0);
print(head);
}
```

§8 定义类型名

用 typedef 定义新类型名

1. 写出变量定义语句；
2. 将变量名换为新类型名；
3. 在前面加关键字typedef，即为原类型增加了一个等价的新类型名

eg1: float x;

eg1: float REAL;

eg1: typedef float REAL;

REAL x, y, z; 等价 float x, y, z;

§8 定义类型名

eg2: typedef int ARRAY[10];
ARRAY a, b, c,d;

等价

eg2: int a[10], b[10], c[10], d[10];

eg4:typedef struct student
{ int num;
char name[20];
struct student *next;
} STUDENT;
STUDENT stu1,stu2,*p;

eg3: char *p; 》 char *POINTER; 》 typedef char *POINTER;

POINTER *p,*s[10]; 等价 char *p,*s[10];

typedef 用来定义类型别名，而非定义变量

typedef 只对已存在的类型增加新类型名,并没有产生新类型

§8 定义类型名

例（考题）：设有定义和声明语句如下：

```
typedef  struct dtype
{ int a;
  struct dtype *b;
} node;
 static node x[3]={5,&x[1],7,&x[2],9,'\0'},*ptr=&x[0];
```

则下列选项中，表达式值不为5的是 A 。

A. x[1].b->a-2　　B. ptr->b->a-2

C. (ptr+1)->a-2　　D. ptr->a