第 8 章　结构、联合与指针

主要内容

- 8.1 结构数据类型
- 8.2 * 联合类型
- 8.3 指针小结

8.1 结构数据类型

- 8.1.1 结构类型定义
- 8.1.2 结构类型变量声明及初始化
- 8.1.3 结构类型变量及其成员的表示与使用
- 8.1.4 结构数组
- 8.1.5 函数间结构类型数据的传递
- 8.1.6 链表

8.1 结构数据类型

- 在实际应用问题中经常要处理一些具有不同数据类型的相关数据。

例如，要建立学生成绩档案，对于每个学生有学号、姓名、成绩、考试日期等不同类型的数据

- C 语言支持的结构类型（ structure ），允许程序设计者将一个或多个不同类型的数据有序地组织在一起，并为之确定一个名字构成一种新的数据类型。

8.1.1 结构类型定义

结构类型定义的一般形式为：

```
struct 结构类型名 {
    成员数据类型 1  成员名 1;
    成员数据类型 2  成员名 2;
    ¼ ¼
    成员数据类型 n  成员名 n;
};
```

8.1.1 结构类型定义

- 一个结构类型中成员项的多少、顺序没有限制
- 成员数据类型的前面不可以指定存储类型
- 成员名和结构类型名可与函数或程序中的其他对象名及其他结构类型中的成员名相同
- 结构类型定义最后的“ }º 后要加上分号“ ;º
- 结构定义可以在函数的内部和函数的外部进行

8.1.1 结构类型定义

- 例如：

```
struct date {
                    unsigned short int day;
                    char month[4];
                    unsigned short int year;
    };
```

8.1.1 结构类型定义

- 处理学生成绩的问题可定义成如下的结构类型：

```
struct stu_score{
        long number;
        char name[20];
        unsigned short int score;
        struct date test_date;
};
```

8.1.1 结构类型定义

```
struct time{
        int h;
        int m;
        int s;
    };

struct time{
        int h,m,s;
};
```

8.1.1 结构类型定义

- 结构类型标识是 struct 与结构类型名的组合

  如： struct stu_score
- 经常在定义结构类型的同时使用 typedef 重新为结构类型取个新类型名
- 例如：

```
typedef struct stu_score{
        long number;
        char name[20];
        unsigned short int score;
        struct date test_date;
}STU_Score;
```

其中 STU_Score 与 struct str_score 的含义一样。

8.1.2 结构类型变量声明及初始化

- 结构类型定义相当于定义了一个模板，其中并无具体的数据，系统不分配实际内存单元
- C 编译器只为用它声明的结构类型变量分配存储空间。

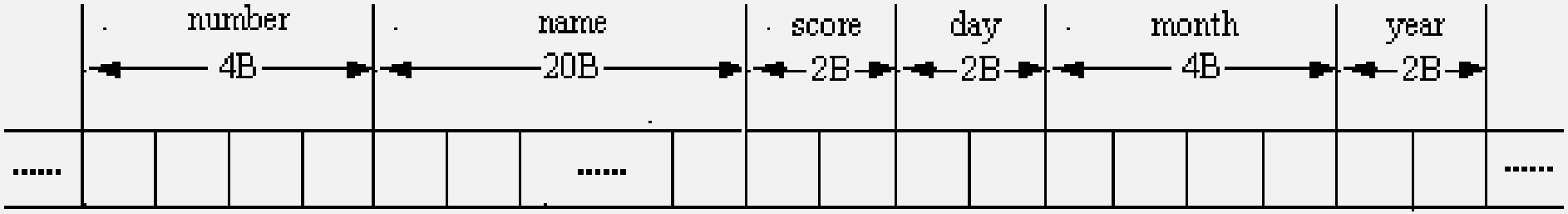
1. 结构类型变量声明

(1) 利用先前已经单独定义的结构类型声明变量。

- 声明的一般形式是：

  struct 结构类型名 结构类型变量；

- 例： **struct stu_score stu1, stu2;**



1. 结构类型变量声明

(2) 先利用 typedef 为已经存在的结构类型取一个别名，然后用该别名声明结构类型变量。

- 例如：

```
typedef struct stu_score{
                long number;
                char name[20];
                unsigned short int score;
                struct date test_date;
}STU_Score;
STU_Score stu1,stu2;
```

1. 结构类型变量声明

(3) 在定义结构类型的同时定义变量名。

- 例如：

```
struct stu_score{
        long number;
        char name[20];
        unsigned short int score;
        struct date test_date;
}stu1,stu2;
```

2. 结构类型变量初始化

带有初始化值的结构类型变量的声明形式为：

struct 结构类型名 结构类型变量名 1 ={ 初始化值表 1}, 结构类型变量名 2 ={ 初始化值表 2 },

¼ ¼

结构类型变量名 n ={ 初始化值表 n};

2. 结构类型变量初始化

- 或者

struct 结构类型名 {

成员数据类型 1 成员名 1;

成员数据类型 2 成员名 2;

¼ ¼

成员数据类型 n 成员名 n;

} 结构类型变量名 1 ={ 初始化值表 1},

结构类型变量名 2 ={ 初始化值表 2 },

¼ ¼

结构类型变量名 n ={ 初始化值表 n};

2. 结构类型变量初始化

- 例如：

```
struct stu_score{
                long number;
                char name[20];
                unsigned short int score;
                struct date test_date;
}stu1={1230001,ª Zhangsanº , 80,16,
   ª JUNº ,2009};
```

8.1.3 结构类型变量及其成员的表示与使用

- 结构类型的变量及其成员可以直接引用或使用指针变量间接引用
- 1. 直接引用结构类型变量及其成员
- (1) 引用结构类型变量
- 同类型的结构类型变量之间可以互相赋值。

例如：

```
struct stu_score stu1={1230001, ª Zhangsanº ,
   80,16, ª JUNº ,2009}, stu2;
stu2=stu1;
```

(1) 引用结构类型变量

- 结构类型变量作为整体只能对同类型变量之间相互赋值、作为函数参数传递、通过sizeof运算符求结构类型变量的存储空间大小及允许取结构类型变量的地址，如&stu1

- 其他将结构类型变量作为整体的操作都是不允许的
- stu1==stu2， stu1-stu2 ×
- scanf("%ld %s%u%d%d%d",&student1);×
- printf("%ld,%s,%u,%d,%d,%d",stu1); ×

(2) 引用结构类型变量的成员

- 引用结构类型变量成员的一般形式为：

  结构类型变量名 . 成员名

- 其中的“ .º 称为结构成员运算符，它有最高优先级，左结合性
- 例：结构类型变量 stu1 的成员可表示为：

  stu1.number

(2) 引用结构类型变量的成员

- “结构类型变量名.成员名” 是一个左值表达式，具有与普通变量完全相同的性质

stu1.number =4096; stu1.score++ ;

- 多级成员访问

例如： **stu1.test_date.year**

2. 使用指针间接引用结构类型变量及其成员

- 通过“ & 结构类型变量名”获得结构类型变量的起始地址。例如， &stu1 。
- 声明指向结构类型的指针变量的一般形式为：

结构类型标识 *指针变量名

例如： struct stu_score *p;

p=&stu1 ； /* 使 p 指向结构类型变量 stu1*/

- 指针变量 p 的基类型为 struct stu_score ， 它只能指向 struct stu_score 型的结构对象。

2. 使用指针间接引用结构类型变量及其成员

- 结构指针变量指向一个结构类型变量后，可以用它来存取所指向的结构类型变量中的成员，使用形式是：

  (* 结构指针名). 成员名

- 如：　struct stu_score stu1, *p=&stu1 ;

  (*p).number　/* 等价于 stu1.number */

2. 使用指针间接引用结构类型变量及其成员

例 8.1 结构指针变量的简单应用。

```
#include<stdio.h>
#include<string.h>
int main(void)
{  struct stu_score{
            long num;
            char name[20];
            unsigned score[3];
            float ave;
   } stu,*p=&stu;
  int i, sum=0;
  stu.num=1230001; strcpy(stu.name, ª Zhangsanº );
  for(i=0;i<3;i++){  scanf(ª %uº ,& stu.score[i]);    sum+= stu.score[i]; }
  stu.ave= sum/3.0;
  printf(ª %ld,%s,%u,%u,%u,%.1f \nº , stu.num,stu.name,stu.score[0], stu.score[1], stu.score[2],
     stu.ave);
  printf(ª %ld,%s,%u,%u,%u,%.1f \nº ,(*p).num,(*p).name,(*p).score[0], (*p).score[1],
     (*p).score[2], (*p).ave);
  return 0;
}
```

2. 使用指针间接引用结构类型变量及其成员

- (* 结构指针名). 成员名　不直观且易写错
- C 语言提供“ ->º 运算符
- (*p).ave 可以改写为　p->ave
- ª ->º 运算优先级与“ .º 结合性相同

例 8.1 最后的输出语句可改为：

```
printf(ª %ld,%s,%u,%u,%u,%.1f\nº ,p->num,p->name,
p->score[0], p->score[1], p->score[2], p->ave);
```

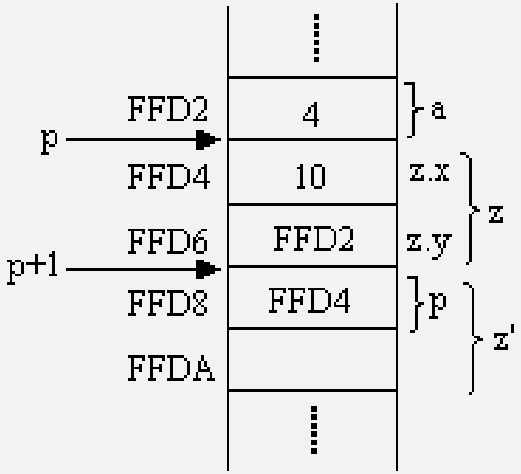
- ª ->º 运算符只用于使用指针变量访问成员

```
printf(ª %ld,%s \nº ,stu->num,stu->name);
```

2. 使用指针间接引用结构类型变量及其成员

- “->”、“.”与“++”、“*”等组合在一起使用时难于分析与理解。

```
int main(void)
{
    int a=4;
    struct {
          int x ,*y;
    }z,*p=&z;
    z.x=10;
    z.y=&a;
    ¼ ¼
}
```




```
*p->y ;
*p->y++ ;
*++p->y ;
*p++->y ;
(*p->y)++ ;
++*p->y ;
++*p->y++ ;
*(p->y++) ;
```

8.1.4 结构数组

- 结构数组即数组中的每个元素都是同一结构类型。
- 在实际应用中，经常使用结构数组表示具有相同数据类型的某个集合，例如一个班级学生的成绩信息。

8.1.4 结构数组

- 结构数组的声明与普通数组的定义在语法上完全一样，也可以声明多维结构数组。
- 与声明结构类型变量一样，也有三种方法声明结构数组。

声明方式1

先定义结构类型，再声明结构数组

```
struct stu_score{
        long num;
        char name[20];
        unsigned score[3];
        float ave;
}

struct stu_score stu[3];
```

声明方式 2

定义结构类型的同时声明结构数组

```
struct stu_score{
        long num;
        char name[20];
        unsigned score[3];
        float ave;
} stu[3];
```

声明方式 3

用结构类型的别名声明结构数组

```
typedef  struct stu_score{
        long num;
        char name[20];
        unsigned score[3];
        float ave;
} STU ;

STU stu[3];
```

8.1.4 结构数组

- 在声明结构数组的同时也可以在数组的后面以“ ={ 初始化值表 }º 的形式
- 对结构数组进行初始化。初始化值表中列出的初始化值对应于每一个结构数组元素中的每一个成员项。
- C 语言也允许把对应于每个结构数组元素的初始化值括在一对大括号中
- 例如：

```
struct stu_score stu[3]={
    {1230001, ª Zhangsanº ,{80,75,82},79.0},
    {1230002, ª Lisiº ,{70,85,72},0.0},
    {1230003, ª Wangwuº ,{83,77,82},0.0}
};
```

8.1.4 结构数组

- 与结构类型变量一样，可以引用结构数组元素的成员。例如：

```
stu[1].num++;
scanf(ª %sº ,stu[2].name);
```

- 可以用指针变量间接引用结构数组的元素。
- 例 8.2 指向结构数组的指针应用。

```
……
struct stu_score *p;
for(p=stu;p<stu+3;p++)
        printf(ª %ld,%s,%u,%u,%u,%.1f \nº ,p->num,p->name,
         p->score[0],p->score[1], p->score[2], p->ave);
……
```

8.1.4 结构数组

例 8.3 从键盘输入学生成绩信息，计算每个学生的平均成绩。使用结构类型的指针数组，按平均成绩降序排列学生成绩信息并输出。

```
#include<stdio.h>
#define MAX 5
typedef struct {
        long num;
        char name[20];
        unsigned score[3];
        float ave;
}STU_Score;
```

例 8.3

```
int main(void)
{
    STU_Score stu[MAX], *stu_index[MAX],*p;
    int i,j;
    printf("a input %d students©data:\no ,MAX);
    for(i=0;i<MAX;i++){      /* 输入学生成绩信息   */
      scanf("%ld %s %u%u%u",&stu[i].num,stu[i].name,
              stu[i].score, stu[i].score+1, stu[i].score+2);
      stu[i].ave=(float)(stu[i].score[0]+stu[i].score[1]
                  + stu[i].score[2])/3;
    }
```

例 8.3

```
for(i=0;i<MAX;i++)  stu_index[i]=stu+i;
for(i=0;i<MAX-1;i++)
    for(j=i+1;j<MAX;j++)
        if(stu_index[i]->ave<stu_index[j]->ave) {
                p=stu_index[i]; stu_index[i]=stu_index[j];
                stu_index[j]=p;
        }
printf("output the ordered students©data:\n" );
for(i=0;i<MAX;i++){
      p=stu_index[i];
      printf("%ld,%s,%u,%u,%u,%.1f \n" , p->num,p->name,
                  p->score[0], p->score[1], p->score[2], p->ave);
  }
return 0;
}
```

8.1.5 函数间结构类型数据的传递

1. 结构类型数据作为函数参数

- 主调函数将一个结构类型的数据传递给被调函数有 3 种方法：
- ⑴ 用结构类型变量的成员作为实际参数。
  - 成员类型为基本数据类型，数据“单向值传递”；
  - 成员类型为数组时，传递的是数组的地址。
  - 被调函数的形式参数类型应与传入的结构类型变量的成员类型对应。

1. 结构类型数据作为函数参数

例 8.4 结构类型变量的成员作为实际参数。

```
#include<stdio.h>
#include<string.h>
typedef struct{
            long num;
           char name[20];
           unsigned short int score[3];
           float ave;
}STU_Score;
int fun(long num, char name[20], unsigned score[ ], float ave)
{´´´´´}
int main(void)
{   STU_Score  stu1={1230001, ª Zhangsanº ,{80,75,82},0.0};
    fun(stu1.num,stu1.name,stu1.score, stu1.ave);
    return 0;
}
```

1. 结构类型数据作为函数参数

- (2) 用结构类型变量作实际参数和形式参数。
  - 数据传递是“值传递”的方式，将结构类型的实参所占存储单元的全部内容（包括其中的数组成员的所有元素值）按顺序传递给对应的形参。

```
int fun(STU_Score  temp)
{
    printf(ª %ld,%s,%u,%u,%u,%.1f \nº , temp.num,temp.name,
     temp.score[0], temp.score[1], temp.score[2], temp.ave);
     return 0;
}
int main(void)
{
    STU_Score stu1={1230001,º Zhangsanº ,{80,75,82},0.0};
    fun(stu1);
    return 0;
}
```

1. 结构类型数据作为函数参数

- (3) 用指向结构类型变量的指针作为函数参数。
- 将结构类型变量（或数组）的地址作为实际参数，传递的是地址，形式参数只需要地址长度的存储空间，系统的开销较小。

```
int fun(STU_Score  *temp)
{
    printf("%ld,%s,%u,%u,%u,%.1f \n",temp->num,temp->name,
    temp->score[0], temp->score[1], temp->score[2], temp->ave);
    return 0;
}
int main(void)
{
   STU_Score     stu1={1230001, "Zhangsan",{80,75,82},0.0};
   fun(&stu1);
   return 0;
}
```

1. 结构类型数据作为函数参数

- 用指向结构类型变量的指针作为函数参数时，若在函数中通过形参结构指针变量改变结构数据的值，则主调函数对应的结构类型变量的存储单元的内容将被改变。

例如：

```
int fun(STU_Score  *temp)
{
    temp->num++;
    return 0;
}
int main(void)
{
    STU_Score  stu1={1230001, ª Zhangsanº ,{80,75,82},0.0};
    fun(&stu1);
    printf(ª %ld\nº ,stu1.num);
    return 0;
```

1. 结构类型数据作为函数参数

- 指向结构类型变量的指针变量作为函数的形式参数，可以接收主调函数传递的结构数组的起始地址，从而可在被调函数中访问主调函数结构数组的存储空间。

- 例 8.5 从键盘上输入学生成绩信息，计算每个学生的平均成绩，使用指针数组建立结构数组按平均成绩降序排列的索引，并输出排序后的学生成绩信息。

```
#include<stdio.h>
#define MAX 5
typedef struct{
        long num;
        char name[20];
        unsigned score[3];
        float ave;
}STU_Score;
```

1. 结构类型数据作为函数参数

```
void  inputdata(STU_Score stu[ ]);  /*  输入学生成绩信息   */
void  sort(STU_Score *stu_index[ ]);  /*  排序函数   */
void  outputdata(STU_Score *stu_index[ ]);  /*  输出排序后的学生
    成绩信息   */
int main(void)
{
    STU_Score stu[MAX], *stu_index[MAX],*p;
    int i;
    inputdata(stu);
    for(i=0;i<MAX;i++)
          stu_index[i]=stu+i;
    sort(stu_index);
    outputdata(stu_index);
    return 0;
}
```

1. 结构类型数据作为函数参数

```
void inputdata(STU_Score stu[ ])
{   int i;
    printf("input %d students©data:\n",MAX);
    for(i=0;i<MAX;i++){
      scanf("%ld %s %u%u%u",&stu[i].num,stu[i].name,
                    stu[i].score, stu[i].score+1, stu[i].score+2);
      stu[i].ave=(float)(stu[i].score[0]+stu[i].score[1]+ stu[i].score[2])/3;
    }
}
void outputdata(STU_Score *stu_index[ ])
{   STU_Score *p; int i;
    printf(ª output the ordered students©data:\nº );
    for(i=0;i<MAX;i++){
          p=stu_index[i];
          printf(ª %ld,%s,%u,%u,%u,%.1f \nº ,p->num,p->name,
                p->score[0], p->score[1], p->score[2], p->ave);
    }
}
```

1. 结构类型数据作为函数参数

```
void sort(STU_Score *stu_index[ ])
{
    STU_Score *p;
    int i,j;
    for(i=0;i<MAX-1;i++)
          for(j=i+1;j<MAX;j++)
                if(stu_index[i]->ave<stu_index[j]->ave) {
                      p=stu_index[i];
                     stu_index[i]=stu_index[j];
                     stu_index[j]=p;
                }
}
```

2. 将结构类型的数据作为返回值

- 函数必须定义成返值为特定结构类型的函数
- 接收该函数返回值的对象也必须是该结构类型的变量

函数返回结构类型的数据也有三种方式。

- (1) 返回值为结构类型数据的成员。

例如：

```
typedef struct{
        long num;
,,,,,,
}STU_Score;
long fun(void)
{
    STU_Score temp;
    temp.num=1230001;
    return (temp.num);
}
```

```
int main(void)
{
    STU_Score stu1;
    stu1.num=fun();
    printf("%ld\n",stu1.num);
    return 0;
}
```

2. 将结构类型的数据作为返回值

- (2) 返回结构类型变量。
- 返回结构类型变量采取的是“值传递”方式，将被调函数 return 语句中的结构类型变量所占存储单元的全部内容（包括其中的数组成员的所有元素）返回给主调函数。

例如：

```
STU_Score fun(void)
{
    STU_Score temp;
    temp.num=1230001;
    strcpy(temp.name,º Zhangsanº );
    temp.score[0]=80;
    temp.score[1]=75;
    temp.score[2]=82;
    temp.ave =79.0;
    return (temp);
```

```
int main(void)
{
    STU_Score  stu1;
    stu1=fun();
    printf(……);
    return 0;
}
```

2. 将结构类型的数据作为返回值

- (3) 返回结构类型变量的地址。
  - 传递的是地址
  - 主调函数只需要使用指针变量接收地址即可，系统的效率较高
  - 函数必须定义成返值为特定结构类型的指针型函数
  - 接收该函数返回值的对象也必须是该结构类型的指针变量
  - 函数中返回的地址量最好为 static 型对象的地址

2. 将结构类型的数据作为返回值

例如：

```
STU_Score * fun(void)
{
    static STU_Score temp;
    temp.num=1230001; strcpy(temp.name, "Zhangsan" );
    temp.score[0]=80, temp.score[1]=75, temp.score[2]=82;
    temp.ave=79.0;
    return (&temp);
}
int main(void)
{
    STU_Score *p;
    p=fun();
    printf("%ld,%s,%u,%u,%u,%.1f \n" , p->num,p->name,
            p->score[0], p->score[1], p->score[2], p->ave);
    return 0;
}
```

2. 将结构类型的数据作为返回值

- 程序中经常将动态存储分配函数与指向结构数据的指针结合起来使用，以实现“动态结构数组”。
- 例 8.6 从键盘上输入学生人数，并输入成绩信息，使用动态存储分配函数开辟相应的存储空间，以存储学生成绩信息，并在主函数中输出学生成绩信息。

```
#include<stdio.h>
#include<stdlib.h>
int Num;
typedef struct {
long num;char name[20];
unsigned score[3];
float ave;
}STU_Score;
STU_Score *creatarray( );  /*  新建结构数组   */
```

2. 将结构类型的数据作为返回值

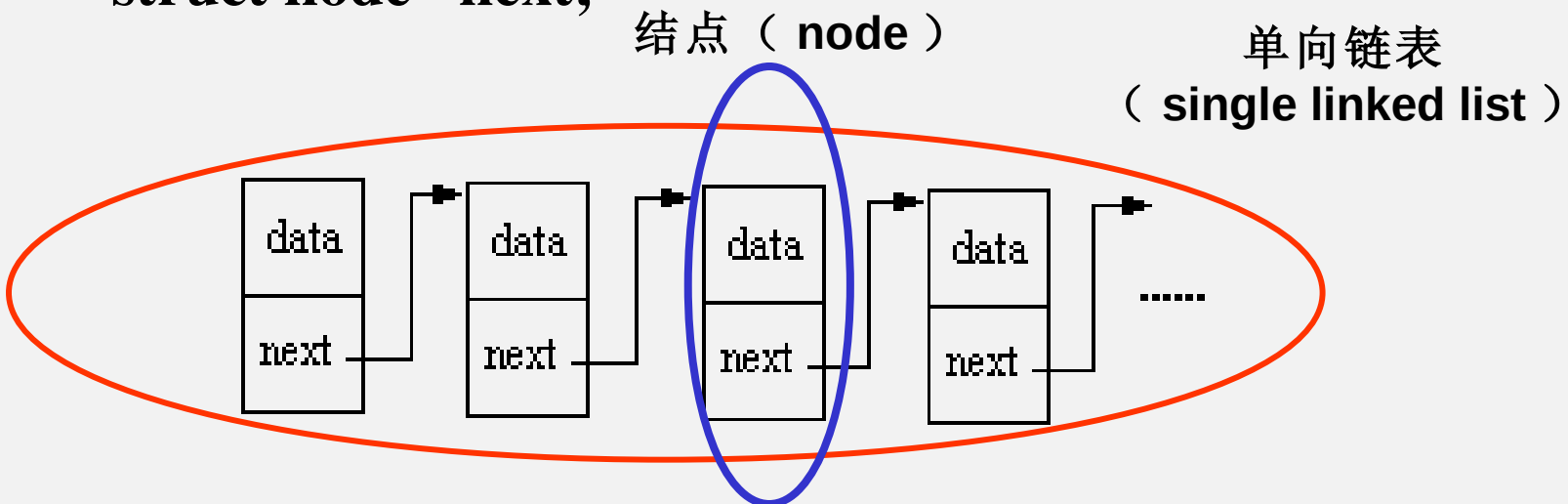
```
int main(void)
{   STU_Score *stu; int i;
    stu=creatarray();
    for(i=0;i<Num;i++)
        printf("%ld,%s,%u,%u,%u,%.1f\n",stu[i].num,stu[i].name,
            stu[i].score[0], stu[i].score[1],stu[i].score[2], stu[i].ave);
    return 0;
}
STU_Score *creatarray(void)
{   STU_Score *head, *p;
    scanf("%d",&Num);
    head=(STU_Score *)calloc(Num, sizeof(STU_Score));
    for(p=head;p<head+Num;p++){
        scanf("%ld%s%u%u%u",&p->num,p->name, p->score,
                            p->score+1, p->score+2);
        p->ave=(float)(p->score[0]+p->score[1]+ p->score[2])/3;
    }
    return head;
}
```

8.1.6 链表

- 1. 链表的基本概念
- 自引用结构：结构的成员项是本结构的一个结构指针变量。

例如：

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



1. 链表的基本概念

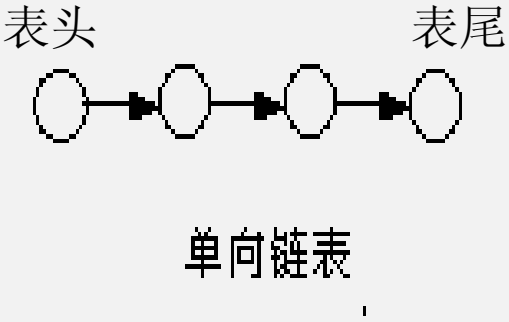
单向链表比结构数组优越：

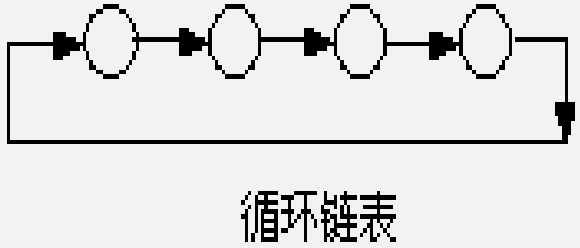
- 结构数组中的元素要连续存放，链表则不必；
- 数组中的元素个数是确定的，链表中的元素个数却没有限制；
- C 编译程序对数组必须给其分配存放它的全部元素的存储空间，对链表则不必也不可能预先分配全部存储空间。

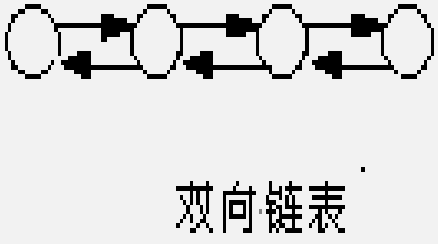
1. 链表的基本概念

根据链表结点之间的相互关系分：

- 单向链表
- 循环链表
- 双向链表







2. 链表结点结构的定义

- 一个单向链表结点中包含两类成员域：
- 数据成员域：存放结点值的数据
- 链接成员域：存放直接后继结点的地址

2. 链表结点结构的定义

- 假定结构类型名为 node，链接成员名为 next，可以定义如下的结构类型：

```
struct node{
        成员数据类型 1   成员名 1;
        成员数据类型 2   成员名 2;
        ¼ ¼
        成员数据类型 n   成员名 n;
                struct node *next;
};
```

2. 链表结点结构的定义

- 例如，若要创建学生单门课程成绩表，每个结点包括学生的学号和成绩，可定义如下的结点类型：

```
struct node{
        long  num;
        unsigned  score;
        struct node *next;
};
```

2. 链表结点结构的定义

- 为方便使用，可用 typedef 为 struct node 结点类型取一个新的名字：

```
typedef struct node{
        long  num;
        unsigned  score;
         struct node *next;
}NODE;
```

3. 单向链表的操作

- 链表的基本操作：
  - 创建链表
  - 撤消链表
  - 遍历链表
  - 查找结点
  - 插入结点
  - 删除结点
  - 结点排序
- 下面以实现学生单门成绩表为例，讨论链表的这几种操作。

3. 单向链表的操作

(1) 创建链表

用自引用结构实现链表有三个问题要解决：

- 第一，必须指出链表第一个结点的位置，否则无法存取该链表中的结点。

  只要定义一个 NODE 型的指针变量（简称头指针变量），使其指向链表的第一个结点。例如： NODE *p;

- 第二，链表结点是动态产生的，即在需要时才开辟一个结点的存储单元，如何获得一个新结点的存放空间

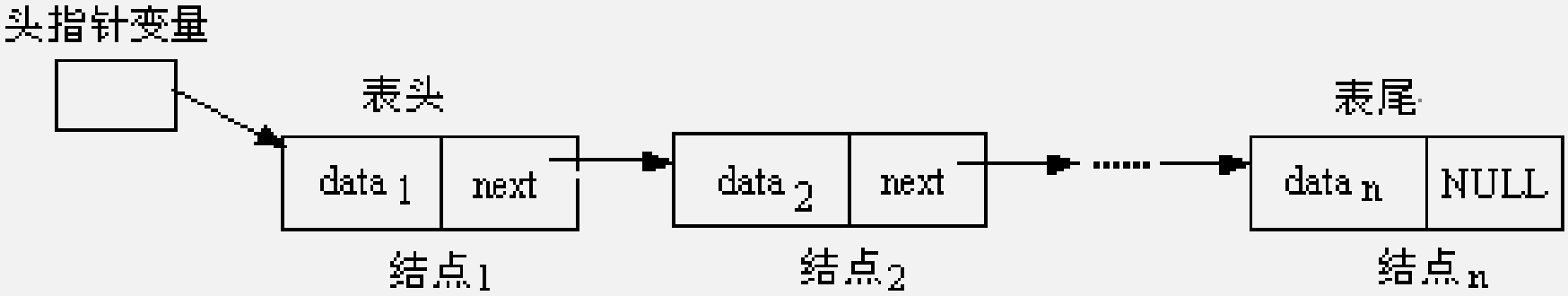
  。可以调用如下形式的内存分配库函数达到：
  (NODE *)malloc (sizeof(NODE))
  程序中只要把由 malloc 函数返回的地址赋给上一结点的成员项 next，便链接到下一个结点的存储位置。

- 第三，要指出链表的链尾。

  通常只要把最后结点中的成员项 next 置为空指针即可

3. 单向链表的操作

- 单向链表建立后，链表第一个结点的地址存放在“头指针”变量中，表尾的链域为NULL。



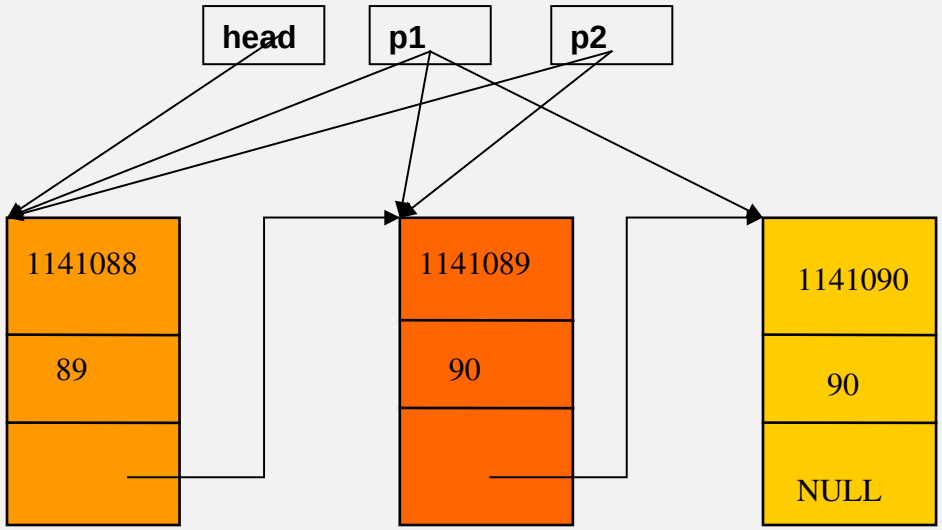
3. 单向链表的操作

- 假定，链表结点成绩数据从键盘上输入，当输入的成绩数据为 0 时，链表创建完成。

```
NODE *creat(void)   /* void 表示无参函数   */
{
   NODE *head=NULL,*p1=NULL,*p2=NULL;
    long num;
   unsigned score;
   int n=0;
```

3. 单向链表的操作

```
do{
   scanf("%ld%u",&num,&score);
   if(num==0) break;
   n++;
   p1=(NODE *)malloc(sizeof(NODE));
   p1->data.num=num,
   p1->data.score=score;
   p1->next=NULL;
   if(n==1)
        head=p2=p1;
   else{
        p2->next=p1;
        p2=p1;
   }
 }while(1);
 return head;
}
```

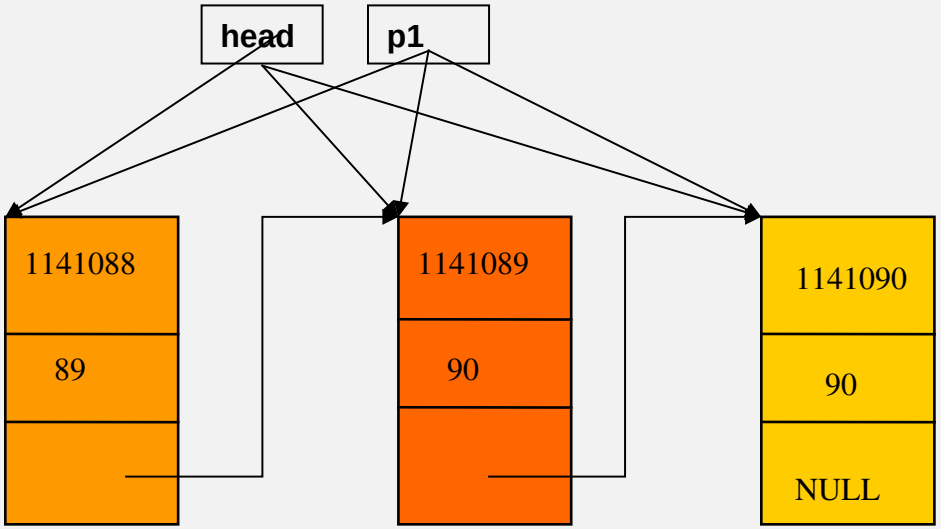


3. 单向链表的操作

(2) 撤消链表

- 撤消链表意味着回收链表中的结点使用的存储空间。
- 可利用 free 函数归还结点空间。

```
void delete_List(NODE *head)
{
   NODE *p1=NULL ;
   if(head==NULL)
        return;
   while(head!=NULL){
        p1=head;
        head = head->next;
        free(p1); }
}
```

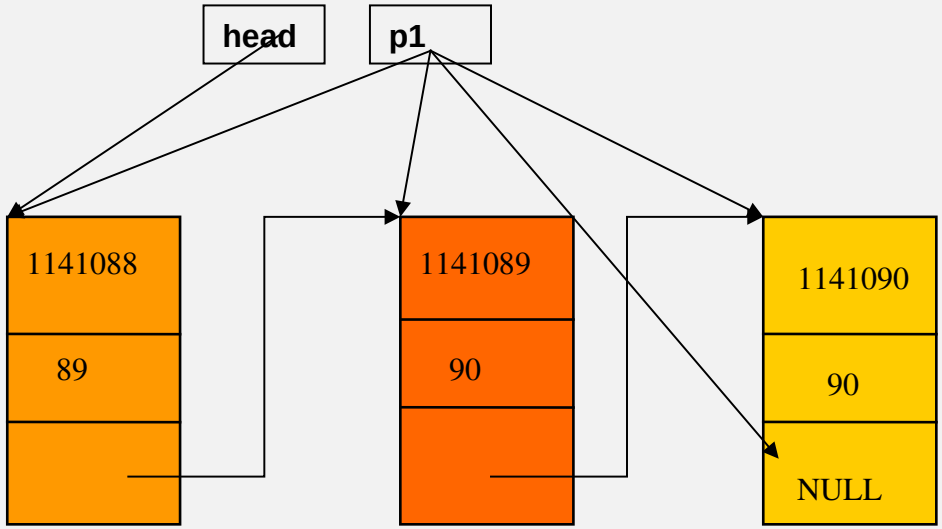


3. 单向链表的操作

(3) 遍历链表

- 在链表中查找指定的数据、显示链表全部结点中的数据、统计链表中的结点数等操作都需要遍历链表操作。
- 例如，下列函数输出给定链表中每个学生的数据。

```
void print(NODE *head)
{
    NODE *p=NULL;
    if(head==NULL) {
        printf("\nlist null!\n");
        return;
    }
    p=head;
    while(p!=NULL) {
        printf("%ld,%u\n",p->data.num,p->data.score);
        p=p->next;
    }
}
```



3. 单向链表的操作

(4) 查找结点

- 链表建立以后，经常需要查找链表内的某个数据，如查询链表中指定学号的学生信息，若查到则返回结点的地址，否则返回 NULL 。

```
NODE * search(NODE *head,long key_num)
 {
    NODE *p=NULL;
    if(head==NULL){
          printf("\nlist null!\n");
          return NULL;
    }
    p=head;
    while(p!=NULL) {
          if(p->data.num==key_num)
                return p;
          p=p->next;
    }
    printf("Not Found!\n");
    return NULL;
}
```

3. 单向链表的操作

(5) 插入结点

- 对链表的插入操作是指将一个结点按某种规律插入到一个已存在的链表中。
- 例如，已有一个学生单门成绩链表，各结点按成员项 **num**（学号）的值从小到大顺序排列。现要插入一个新生的结点，要求按学号的顺序插入。
- 必须解决两个问题：
  - 如何找到插入的位置
  - 怎样实现插入

3. 单向链表的操作

- 若 head 是链表的头指针，变量 p0 指向待插入的结点，为在 head 指向的链表中寻找插入位置，可设立 p1 和 p2 两个指针变量， p2 指向链表当前寻找位置的前一个结点， p1 指向 p2 的后继结点。
- 如何找到插入的位置
  - 当 head==NULL，插入位置为表头；
  - 否则，遍历链表寻找位置满足条件：

    p2->data.num≤p0->data.num≤p1->data.num

3. 单向链表的操作

```
p1=head;
while((p0->data.num>p1->data.num)&&(p1->next!
   =NULL)){
p2=p1;
p1=p1->next;
}
```

- 插入位置：
  - 如果 p1==head ， 表示 p0 插入在表头；
  - 如果 p0->data.num <p1->data.num ， 表示 p0 插入在 p2 与 p1 之间 ；
  - 如果 p1->next==NULL, 表示 p0 插到链表末尾。

3. 单向链表的操作

- 怎样实现插入：
  - 插入在表头；

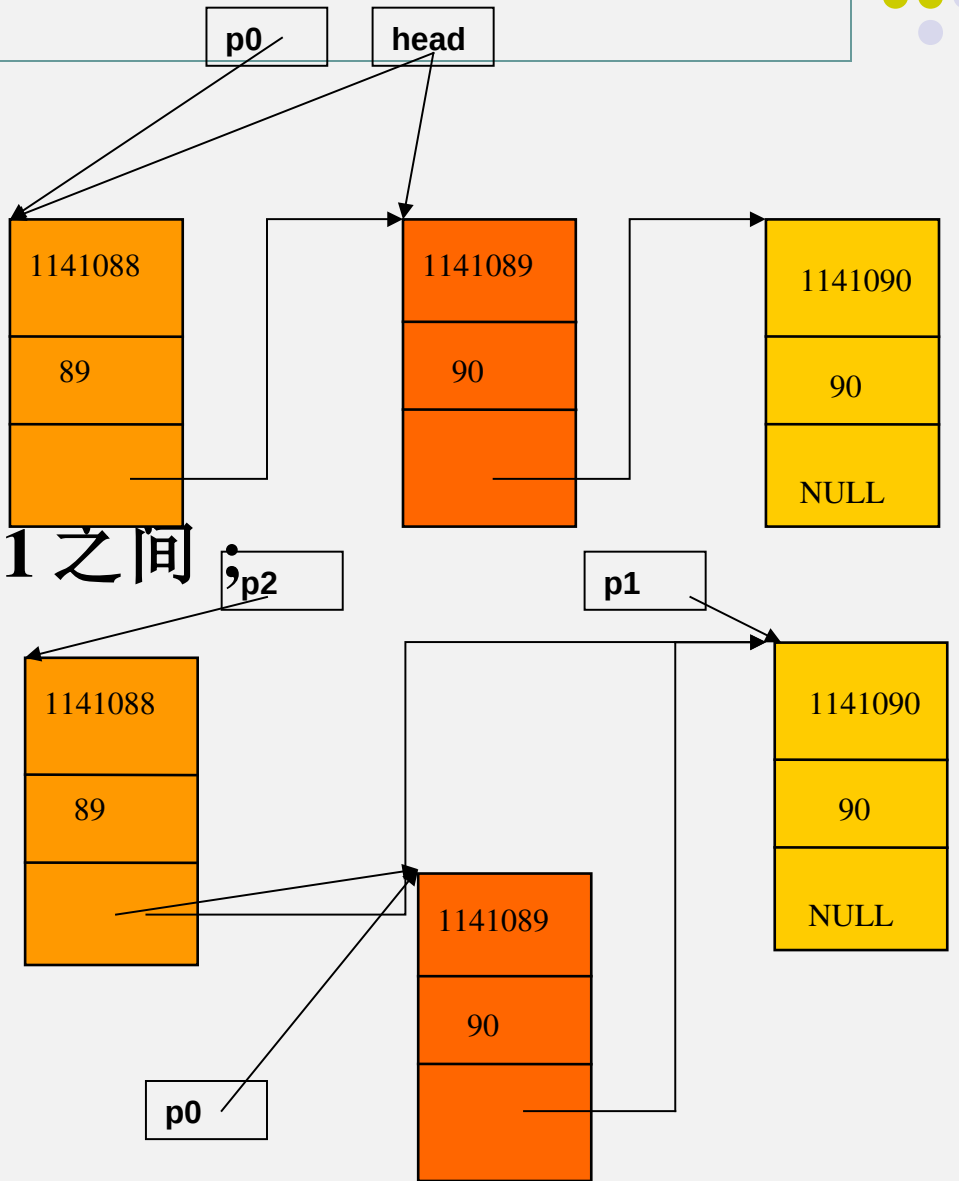
p0->next=head;

head=p0;

  - **p0 插入在 p2 与 p1 之间；**

p2->next=p0;

p0->next=p1;

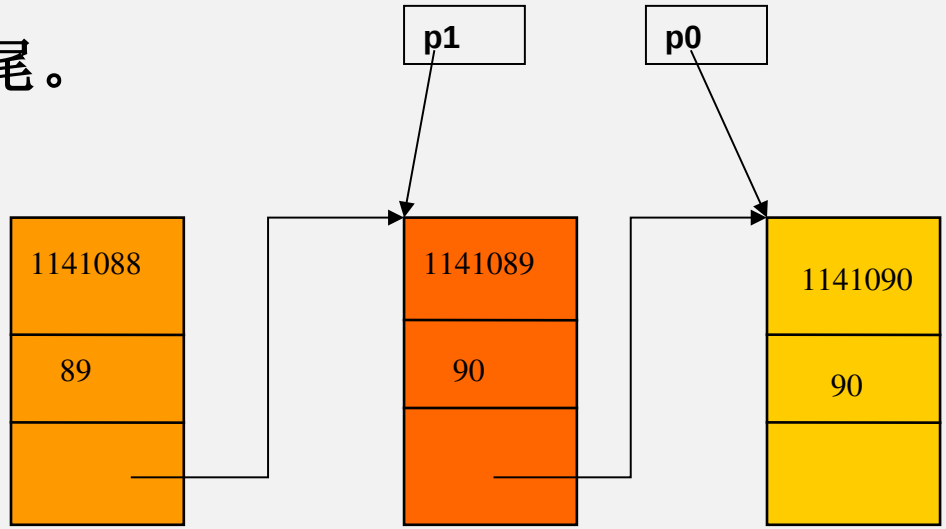


3. 单向链表的操作

- p0 插到链表末尾。

p1->next=p0;

p0->next=NULL;



- 最后要返回 head, 作为插入以后的新链表的表头

```
NODE *insert(NODE *head,NODE *p0)
{       ,,,,,,
        return head;
}
```

3. 单向链表的操作

- 有了 insert 函数，便可以利用它方便地创建一个有序的链表。

```
NODE *creat_seq(void)
{
    NODE *head=NULL,*p=NULL;
     long num;unsigned score;
    do{
          scanf("%ld%u",&num,&score);
          if(num==0) break;
          p=( NODE *)malloc(sizeof(NODE));
          p->data.num=num, p->data.score=score;
          p->next=NULL;
          head=insert(head,p);
    }while(1);
    return head;
}
```

NODE stu

p=&stu

3. 单向链表的操作

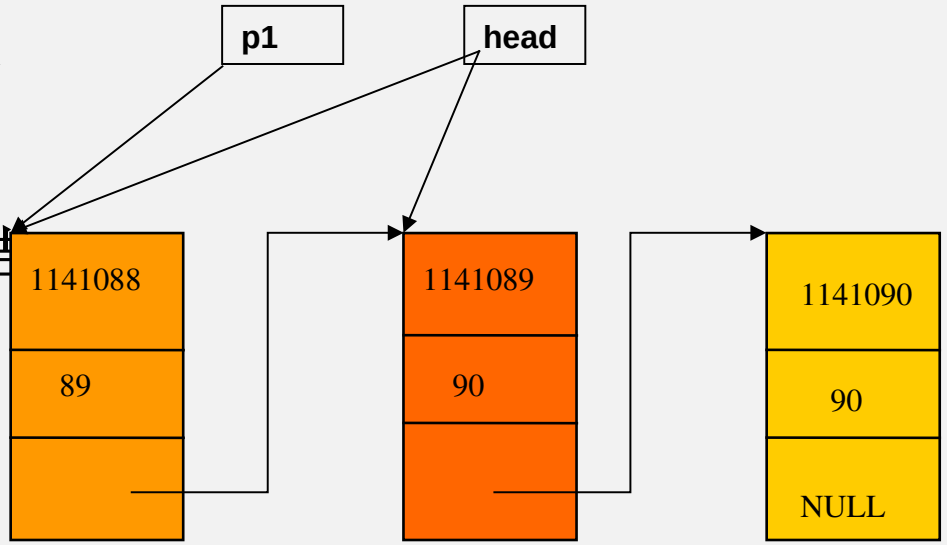
(6) 删除结点

- 从已存在的链表中删除一个符合条件的结点。
- 主要工作：
  - 在链表中查找符合删除条件的结点
  - 调整链接关系
  - 归还结点空间　　free(p1)
- 遍历链表，查找符合条件的结点 p1 时，保存它的前驱结点为 p2

3. 单向链表的操作

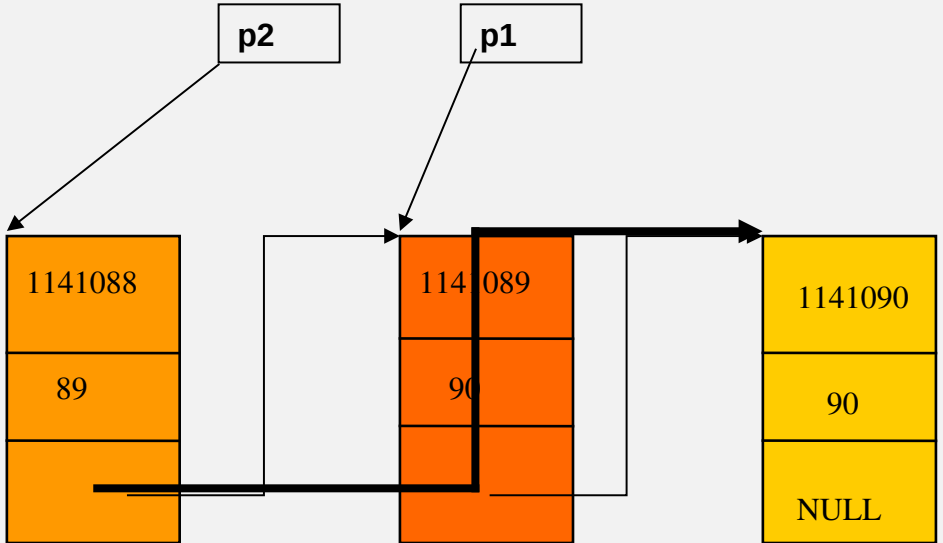
- 若找到符合条件的结点：
  - p1==head，即第一个结点即为要删除的结点

```
head=p1->next;
free(p1);
```



  - p1!=head，即要删除结点在中间或表尾

```
p2->next=p1->next
free(p1);
```



3. 单向链表的操作

```
NODE *delete_Node(NODE *head,long key_num)
{
    NODE *p1=NULL,*p2=NULL;
     if(head==NULL) {
            printf("\nlist null!\n");
            return(head);
    }
    p1=head;
    while((p1->data.num!=key_num)&&p1->next!=NULL) {
            p2=p1;
            p1=p1->next;
    }
    if(p1->data.num==key_num) {  /*  找到符合条件的结点   */
            if(p1==head)
                    head=p1->next;  /*  结点为表头   */
            else
                    p2->next=p1->next; /*  结点在中间或表尾   */
            free(p1);
            printf("delete:%ld\n",key_num);
    }
    else
            printf("%ld not been found! \n",key_num);
   return head;
}
```

3. 单向链表的操作

(7) 结点排序

- 对已建立的一个无序链表，若要按照某种规律将它排序，如按学号从小到大排序，最简单的方法是使用前面介绍的插入函数。

3. 单向链表的操作

```
NODE *insert(NODE *head,NODE *p0);
NODE *sort(NODE *head)
{
    NODE *p=NULL,*temp=NULL;
    if(head==NULL) {
          printf("\nlist null!\n" );
          return(head);
    }
     p=head->next;;
     while(p!=NULL) {
           temp=p->next;
           head=insert(head,p);
           p=temp;
    }
    return head;
}
```

3. 单向链表的操作

例 8.7 链表的综合操作。

对链表的综合操作只需在主函数中调用前述建立，输出，删除，插入等函数，就能完成所需的功能。本例实现如下的程序功能：

- 创建学生单门成绩链表，并输出；
- 将链表排序后输出；
- 输入一个学号，查找学生的成绩；
- 在链表中插入一个新的学生结点；
- 从键盘上输入的学号，并从已排序的链表中删除该学号的结点；
- 删除链表。

3. 单向链表的操作

```
NODE *insert(NODE *head,NODE *p0);
NODE *sort(NODE *head);
NODE *delete_Node(NODE *head,long key_num);
NODE * search(NODE *head,long key_num);
void print(NODE *head);
void *delete_List(NODE *head);
NODE *creat(void);
```

3. 单向链表的操作

```
int main(void)
{
    NODE *head,*p;
    long key_num;
    printf("input records(number  score):\n");
    head=creat();  /*  创建无序链表    */
    printf("print records:\n");
    print(head);   /*  输出全部结点    */
    head=sort(head);  /*  排序    */
    printf("print the sorted records:\n");
    print(head);
    printf("\ninput the number for search:");
    scanf("%ld",&key_num);
    p=search(head,key_num);
    printf("\nthe score is %u",p->data.score);
```

3. 单向链表的操作

```
    printf("\ninput the inserted records:");
    p=(NODE *)malloc(sizeof(NODE));
    scanf("%ld%u",&p->data.num,&p->data.score);
    p->next=NULL;
    head=insert(head,p);
    printf("print records:\n");
    print(head);
    printf("\ninput the  number for delete:");
    scanf("%ld",&key_num);   /* 输入要删除的学号   */
    p=delete_Node(head,key_num);
    printf("print records:\n");
    print(head);
    printf("\ndelete all records:\n");
    head=delete_List(head);
    print(head);
    return 0;
}
```

8.2 * 联合类型

- 联合类型对应的变量中，不同的时刻可以存储不同类型的数据，或者说允许利用同一存储区域来存储、处理不同类型的数据。

8.2 * 联合类型

- 联合数据类型的定义一般形式为：

```
union 联合类型名 {
        成员数据类型 1 成员名 1;
        成员数据类型 2 成员名 2;
    ¼ ¼
        成员数据类型 n 成员名 n;
};
```

```
union data{
    int k;
    char ch;
    float f;
};
```

8.2 * 联合类型

- 联合类型的变量的声明形式三种方式：

声明方式 1

先定义联合类型，再声明联合对象

```
union data {
        int k;
        char ch;
        float f;
}
union data a,b,c;
```

声明方式 2

定义联合类型的同时声明联合对象

```
union data {
        int k;
        char ch;
        float f;
} a,b,c;
```

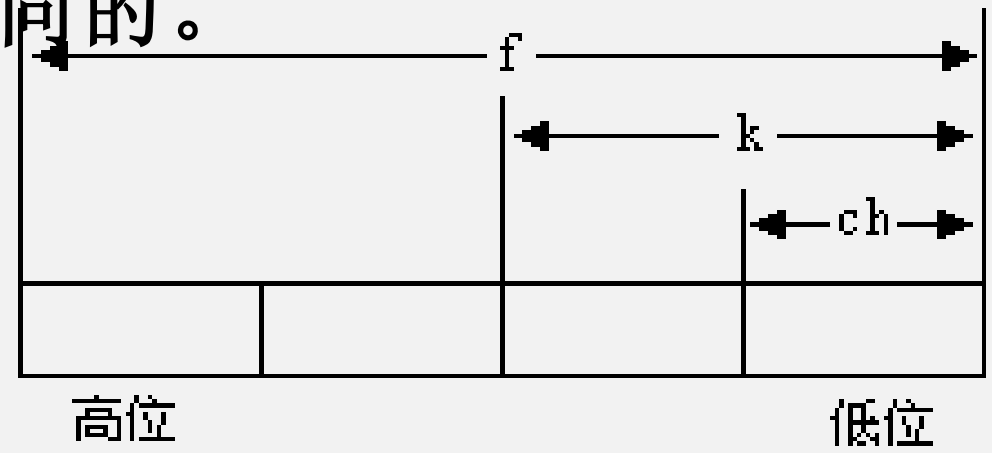
声明方式 3

用联合类型的别名声明联合对象

```
typedef union data {
        int k;
        char ch;
        float f;
} DATA ;
DATA a,b,c;
```

8.2 * 联合类型

- C 编译器将按三个成员中具有最大存储长度的成员的存储大小为每一个联合变量分配存储空间，且让这三个成员使用同一个存储区
- 一个联合类型变量的所有成员的存储起始地址都是相同的。



8.2 * 联合类型

- 联合类型与结构类型的区别：
  - 结构类型变量所占内存长度是各成员占的内存长度之和，每个成员分别占用其各自的内存单元。
  - 联合类型变量所占的内存长度等于最长的成员的长度。
- 如对于：

```
struct sdata
{       short int k;
        char ch;
        float f;
 } sa;
```

```
union udata
{       short int k;
        char ch;
        float f;
 } ua;
```

- 结构类型变量 sa 占 7 个字节 的存储空间，而联合类型变量 ua 则占 4 个字节。

8.2 * 联合类型

- 大多数 C 编译器允许在声明联合类型变量时对其指定初始化值，但只能有一个初值，且初始化的类型只能是第一个成员的类型。
- 例如：

```
union udata{
        int k;
        char ch;
        float f;
}a=123;
```

8.2 * 联合类型

声明了联合类型变量后可以引用变量的某个成员。

- 引用形式与结构类型变量成员的引用形式相同，利用成员运算符“ .º ，
- 对于指向联合类型变量的指针变量也可以用“ ->º 引用其成员。
- 一个联合类型变量，在不同时刻可以解释成不同类型的量来使用。
  - a.k 将把 4 个字节的前两个按为 int 型数据解释；
  - a.ch 将把 4 个字节的第一个字节按字符型数据解

8.2 * 联合类型

- 使用联合类型变量时需注意，联合类型变量中起作用的成员值是最后一次存放的成员的值。

如：

a.k=1;

a.ch='a';

a.f=1.5;

- 此时变量 a 的存储单元的值为 float 型数据 1.5 。

8.2 * 联合类型

- 联合类型变量的使用与限制与结构类型变量的使用与限制相同。
  - 只有相同类型的联合类型变量才可以相互赋值；
  - 联合可以作为函数参数类型，也可以作为函数的返回值类型等。
  - 可以声明指向联合类型变量的指针变量。

8.2 * 联合类型

```
#include<stdio.h>
union AA{
        long l;
        int k;
        char ch;
};
union AA f(union AA a)
{
        printf("%8lx",a.l);
        a.k++;
        return a;
}
```

```
int main(void)
{
   union AA ua=0x306141,ub;
   ub=ua;
   printf('a %c' ,ub);
   ub=f(ua);
   printf('a %8lx' ,ub);
   return 0;
}
```

8.2 * 联合类型

- 可以定义联合数组，数组也可以作为联合的成员。
- 可以定义指向联合类型变量的指针类型。
- 联合类型可作为结构类型的成员，反之亦然。

```
int main(void)
{
   union{
         int i[2];
         long k;
         char c[4];
   }r ,*s=&r;
   s->i[0]=0x39;
   s->i[1]=0x38;
   printf("%c\n",s->i[0]);
}
```

8.2 * 联合类型

例 8.8 联合类型的实际应用例。

- 假定某校学生会组织了一次向“希望工程”捐书的活动，组织者设计了如下的一张表格用来登记捐书者的有关信息：

姓名	职业	捐书量	工作单位／部队番号

- 其中，职业按工、农、商、学、兵分类，并利用字母 W、P、B、S、A 分别代表之。对军人而言“工作单位”栏填写的是该军人所在的部队番号（一个 long 型数）。

```
#define MAXSIZE 888
int main(void)
{
    struct person{
            char name[20];
            char job;
            int books;
            union {
                    long n;
                    char serv[30];
            } punit;
    }record[MAXSIZE];
    int  i , j ;
```

8.2 * 联合类型

```
for(i=0;i<MAXSIZE;i++){
        printf (ª Enter data please:º );
        scanf(ª %s%1s%dº ,record[i].name, &record[i].job ,
       &record[i].books );
        if(record[i].books==0)
             break;
        else if(record[i].job==`A'){
            printf(ª Enter designation:º );
            scanf(ª %ldº ,&record[i].punit.n);
         }
        else{
            printf(ª Enter unit of service:º );
            scanf(ª %sº ,record[i].punit.serv);
        }
}
```

```
    printf("\n%-20s%6s%6s%6s\n", "name"  ,
    "job","books","unit");
   for(j=0;j<i;j++){
        printf("%-20s %6c %6d",record[j].name ,
           record[j].job,record[j].books);
        if(record[j].job==`A')
            printf("%ld\n",record[j].punit.n);
        else
            printf("%s\n",record[j].punit.serv);
     }
}
```

8.3 指针小结

- 8.3.1 指针与指针变量
- 8.3.2 利用指针存取指向的数据对象
- 8.3.3 指针运算
- 8.3.4 在函数间传递数据对象的地址
- 8.3.5 指针的综合应用例

8.3.1 指针与指针变量

1. 指针

- 指针是 C 语言中的一种特殊的数据类型（指针类型）
- 指针的值是内存单元的地址。
- 不同的系统工程中指针型数据的存储空间大小不同。
- 指针不同于整型量，二者不相容。
- 程序中要获取数据对象的地址可用“ &”运算符实现。

1 指针

- (1) 获取变量 x 的地址。 &x
- (2) 获取一维数组中数组元素的地址。

  int x[N]; &x[i] 或 x+i
- (3) 获取二维数组中数组元素的地址。

  int x[N][M]; &x[i][j] 或 x[i]+j
- (4) 获取二维数组中的行地址。

  int x[N][M]; x[i] 或 x+i 。
- (5) 获取字符串常量中的字符的地址。

  "abcde"+i

  "ABC"=="ABC"

1 指针

- (6) 获取结构类型变量及其成员的地址。

```
struct data {
        int x;
        float y;
}z;
```

&z， &z.x， &z.y

- (7) 函数的地址。

```
double fun(double x)
{
    return x*x-x+1;
}
```

则 fun 函数的地址为 fun，是一个指针型常量。

- (8) 数组的地址。

int array[10]; 则数组 array 的地址为 array，与 &array[0] 等价

- 另外因寄存器无地址可言，所以不能获取 register 型变量的地址。register int x; 则用 scanf("%d",&x) 向变量 x 读入数据是错误的。

2. 指针型变量

- 程序中可以声明指针类型的变量，用于存放数据对象的地址。
- 当一个指针变量存储了某个对象的地址后，就称该指针变量指向那个对象。
- 指针变量定义的一般形式为：

  存储类型　数据类型　*变量名

- 基类型 (可以是各种数据类型名　)

2. 指针型变量

- (1) 指向变量的指针变量。例如： int *p;
- (2) 指向一维数组的指针变量。例如： int (*p)[N];
- (3) 指向函数的指针变量。例如： int (*p)(int,int);
- (4) 一维指针数组。例如： int *p[N];
- (5) 指向指针变量的指针变量。例如： char **p;
- C 语言允许定义如下形式的指针变量：

  void *变量名；

2. 指针型变量

- 使用 void 型指针变量中存储的地址值赋给另一个具有确定基类型的指针变量时，则应该用强制类型转换运算符将它先转换成相应的基类型指针后再赋给那个指针变量。例如：

```
int main(void)
{
    void *p;
    int a , *ip;
    p = &a;
    ip=( int*) p;
    ¼ ¼
}
```

2. 指针型变量

- 定义指针变量的同时可以给指针变量置初始值：

  存储类型　基类型　*变量名 = 初始化值；

- 若未给全局和 static 型指针变量指定初始化值，其初值自动为 0 (NULL)。
- 具有 0 值的指针变量未指向任何对象，称之为“空指针变量”。定义指针变量时可用如下的方法直接把其初始化成空指针。

  int *p=0;　或 int *p=NULL;

- 除数值 0 之外，不能简单地把一个整数当

2. 指针型变量

- 局部与非 static 型指针变量在没有指定初值之前其值无定义(其值不确定)。程序中不应直接使用无定义指针变量存取数据，否则破坏其他数据对象的危险性很大。例如：

```
int main(void)
{
    int *ip;
    scanf(ª %dº ,ip);
    printf(ª %d\nº ,*ip);
}
```

2. 指针型变量

- 指向指针变量的指针变量的定义形式如下：

  存储类型　数据类型　** 指针变量名；

- 可以定义更多重的指针变量，不过二重以上的间接访问用得较少。

2. 指针型变量

- 一旦定义了指针变量，便可用它来存放(指向)与其基类型相同的某个对象的地址。
- (1) 保存变量的地址。例如：

```
float x,*p=&x;
struct {
        int  x, y;
}s,*sp=&s;
int a[10],*ap=&a[3];
```

- (2) 用指向一维数组的指针变量保存行地址。例如：

```
int  y[3][10],  (*ptr1)[10]; ptr1= y+1;
struct {
        int  x, y[3][10];
    }s;
 int  (*ptr2)[10]=s.y;
```

2. 指针型变量

- (3) 用指向函数的指针变量保存函数的地址。例如：

```
#include<math.h>
double  (*p)(double);
p=cos;
```

- (4) 用指针数组保存对象地址。例如：

```
int  *p[3]={&a,&b,&c};  /*  其中的变量 a,b,c 必须是全局变量或 static 型局部变量   */
```

- (5) 用指向指针变量的指针变量保存指针数组元素的地址。例如：

```
int *a[3],**p;
p=a+1;
```

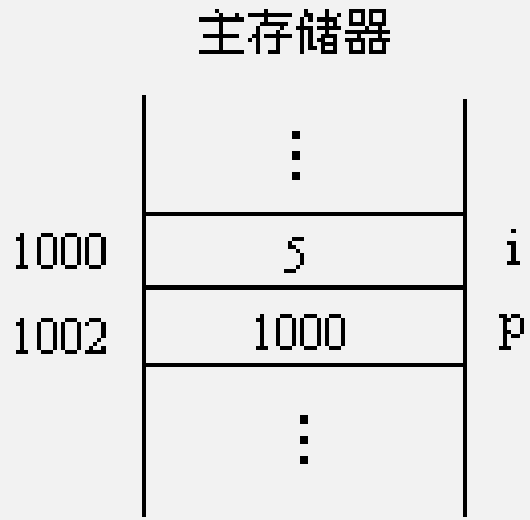
8.3.2 利用指针存取指向的数据对象

- 间接引用运算符： *
- 利用指针存取指向的数据对象一般形式是：*p

```
int main (void)
{
    int i=5,*p;
    p=&i ;
    *p=*p+10;
}
```

8.3.2 利用指针存取指向的数据对象

- ***p=*p+10**
- **等价于**
  i=i+10;



8.3.2 利用指针存取指向的数据对象

利用指针间接存取指向数据对象的简单应用。

- (1) 用指向变量的指针变量引用变量。

  例如：

  int a=10,*p=&a;

  使用 ***p**， ***&a**，都即可引用变量 **a**

- (2) 用指向变量的指针变量引用一维数组元素。

  例如：

  int a[N],*p=a,i;

  则使用 ***(p+i)** 或 **p[i]** 可以引用数组元素 **a[i]**

8.3.2 利用指针存取指向的数据对象

- (3) 用指向变量的指针变量引用二维数组元素。

  例如：

  int a[M][N],*p=a[0],i,j;

  则使用 ***(p+i*N+j)** 可以引用数组元素 **a[i][j]**

- (4) 用指向一维数组的指针变量引用二维数组元素。例如：

  int a[M][N],(*p)[N]=a,i,j;

  则使用 ***(*(p+i)+j)** 或 **p[i][j]** 可以引用数组元素 **a[i][j]**。

8.3.2 利用指针存取指向的数据对象

- (5) 用指针数组中的元素引用对象数据。

  例如：

  char *p[ ]={ª abcº , ª defgº , ª hijkº , ª lmnº };

  则可以用 **p[2][3]** 引用字符 ‘ **k'** ， 用 **p[3]** 引用字符串 “ **lmnº** 。

- (6) 用指向指针变量的指针变量引用指针数组中的元素。

  例如：

  char *a[]={ª abcº , ª defgº , ª hijkº , ª lmnº },p=a;**

  则使用 ***(p+i)** 或 **p[i]** 可以引用元素 **a[i]** 。 ***(p[2]+3)** 或 ***(*(p+2)+3)** 或 **p[2][3]** 引用字符 ‘ **k'** 。

8.3.2 利用指针存取指向的数据对象

- (7) 用指向函数的指针变量调用函数。

  例如：

  double (*p)(double),x=1,y;

  则执行

  p=sin;

  y=(*p)(x); /* 或 y=p(x) */

  则相当于执行“ y=sin(x);º

8.3.3 指针运算

- C 语言中允许对指针进行“ &º 求地址运算、间接引用运算“ *º 及赋值“ =º 运算、算术运算和关系运算。

8.3.3 指针运算

1. 指针的算术运算

- ? p±I

  例如：

  float a[4],*p=a;

  假设 a 的地址为 1000， p+2 的值类型为 float *，值 1008。

- ? p1- p2
  - p1、p2 同时指向同一个数组的元素，表明两个数组元素之间的元素个数，表达式值类型为 int。例如：

  float a[3][4],*p1=&a[2][3],*p2=&a[1][2];

  则 p1-p2 的值为 5。
  - 注意，不允许进行两个指针型数据的加法运算。

1. 指针的算术运算

- 作为 p1-p2 指针减法的实际应用，下面的例子实现 strlen 函数的功能。

```
int strlen (char *s)
{
    char *p=s;
    while (*p!= `\0')
    p++;
    return (p-s) ;
}
```

1. 指针的算术运算

- ? p++ 、 p-- 、 ++p 、 --p
  - p 不能是指针常量。
  - 表达式值类型与 p 相同，效果相当于 p=p+1 、 p=p-1 。 p++ 与 p-- 表达式的值为加减之前 p 的值，而 ++p 与 --p 表达式的值是加减运算以后 p 的值。

8.3.3 指针运算

2. 指针的关系运算

- 同类型的指针型数据 p1 、 p2 都指向同一数组中的元素。
  - 若 p1<p2 为真，表示指针 p1 指向对象的存储位置在 p2 所指向对象的存储位置的前面（在地址小的方向）；反之， p2 在 p1 的前面。
  - 若 p1==p2 为真，则 p1 、 p2 指向同一个对象。
- 允许将指针变量与 NULL 或数值 0 进行 == 或 != 的直接比较，这主要用于判定一个指针变量是否为空指针。

8.3.4 在函数间传递数据对象的地址

- 1. 指向变量的指针变量作函数形参

例如：

```
void f ( int  *p)   /*  int *p 可写成 int p[ ] 或 int p[N] */
 {
   ,,,,,,
}
```

假定主调函数中有如下声明：

```
static int x,y[3]={1,2,3},z[3]={4,5,6}, *q=&x,*p[10]={&x,y,&z[2]};
struct  { int  x,y;}s;
```

则在主调函数中如下的 f 函数调用都是正确的。

```
f(&x);
f(y);
f(&y[2]);
f(y+1);
f(&s.x);
f(q);
f(p[0]);
```

8.3.4 在函数间传递数据对象的地址

- 2. 行指针变量作函数形参

例如：

```
void f(int (*p)[N])  /* int (*p)[N] 可写成 int p[ ][N] 或 int p[M][N] */
{
  ,,,,,,
}
```

假定主调函数中有如下声明：

```
int  a[3][N] ，  (*q)[N]=a+1;
struct {
 int  x,y[4][N];
 }s;
```

则在主调函数中如下的 f 函数调用都是正确的。

```
f(a) ;
f(a+1) ;
f(s.y ) ;
f(q) ;
```

8.3.4 在函数间传递数据对象的地址

- 3．指向指针变量的指针变量作函数形参

例如：

```
void f (char **p)   /* char **p 可写成   char *p[N] 或 char *p[ ] */
 {
   ,,,,,,

 }
```

假定主调函数中有如下声明：

```
char *a[N], **q=a;
```

则在主调函数中如下的 f 函数调用都是正确的。

```
f(a) ;
f(a+1) ;
f(q) ;
```

8.3.4 在函数间传递数据对象的地址

- **4. 指向函数的指针变量作函数形参**

例如：

```
void f (double  (*p)(double ), double x)
{
……

}
```

在主调函数中调用 f 函数时，传给形参 p 的实参必须为函数的地址。例如：

```
f(sin,1.5);
```

8.3.4 在函数间传递数据对象的地址

- 5. 函数类型为指针类型

例如：

```
typedef struct {
            int day;
            char month[10];
            int year;
}TIME;
TIME *creat(void)
{
    TIME *head;
    ……
    return head;
}
```

8.3.5* 指针的综合应用例

- 例 8.9 银行叫号模拟系统。
- 通过编写一个简单的银行叫号模拟系统，综合应用所学习过的数组、结构、指针等知识。