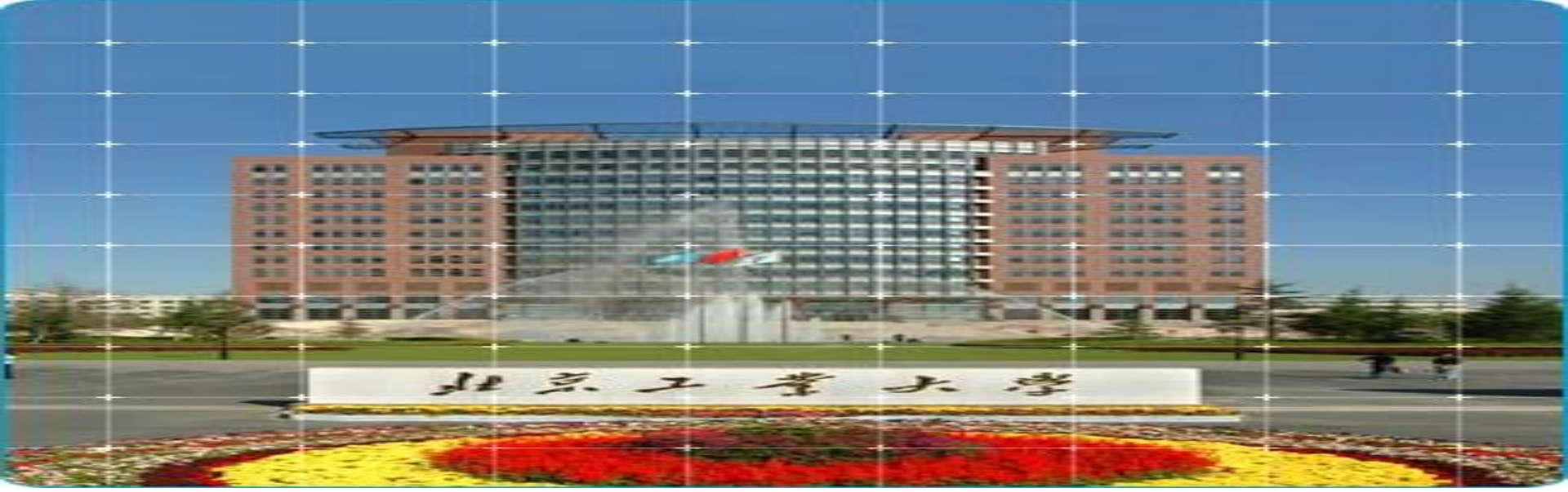


C语言程序设计
------结构

导例：手机类型

1. 问题描述

创建一种手机类型，其中包含手机品牌、手机颜色、手机厚度数据项，然后在主函数中创建该类型的一个普遍变量，一个数组，并显示它们的信息。

```
struct mobile{  /*定义手机结构类型*/
        char brand[20];     /*手机品牌 */
        char color[20];     /*手机颜色 */
        double thickness;           /*手机厚度 */
};
```

```
#include <stdio.h>
#include <string.h>
struct mobile{      /*定义手机结构类型*/
        char brand[20];       /*手机品牌 */
        char color[20];       /*手机颜色 */
        double thickness;     /*手机厚度 */
};
```

```
void main ( )
{ /*定义mobile类型的变量和数组，并初始化结构数组变量*/
    struct mobile m1,m2[3]={{"iphone","black",7.6},{"nokia","red",12.1},{"samsung","blue",8.5}};
    strcpy(m1.brand,"blackberry");  /*为m1赋值*/
    strcpy(m1.color,"black");
    m1.thickness=6.5;
    printf("m1的值为：\n");
    printf("%s\t%s\t%3.1f\n",m1.brand,m1.color,m1.thickness);       /*输出m1*/
    strcpy(m1.brand,"sony");                    /*修改m1的brand值*/
    printf("m1变量修改后的值为：\n");
    printf("%s\t%s\t%3.1f\n",m1.brand,m1.color,m1.thickness);       /*输出m1*/
    printf("m2数组的值为：\n");
    for(int i=0;i<3;i++)                /*输出m2*/
        printf("%s\t%s\t%3.1f\n",m2[i].brand,m2[i].color,m2[i].thickness);;
}
```

结构的概念与定义

结构与数组：
- 都是构造类型，是多个变量的集合
- 数组成员类型相同，结构成员类型不同

- 使用结构来表示学生信息：

```
struct mobile{     /*定义手机结构类型*/
    char brand[20];        /*手机品牌 */
    char color[20];        /*手机颜色 */
    double thickness;      /*手机厚度 */
};
```

- 结构是C语言中一种新的构造数据类型，它能够把有内在联系的不同类型的数据统一成一个整体，使它们相互关联
- 结构又是变量的集合，可以按照对基本数据类型的操作方法单独使用其变量成员。

结构的概念与定义

- 结构类型定义的一般形式为：

```
struct 结构名
{
 类型名 结构成员名1；
 类型名 结构成员名2；
 ● ● ●
 类型名 结构成员名n；
};
```

关键字struct和它后面的结构名一起组成一个新的数据类型名

结构的定义以分号结束，C语言中把结构的定义看作是一条语句

结构的概念与定义

➢ 例如，平面坐标结构：

```
struct point
 {
    float     x;
    float     y;
 };
```

- 虽然x、y的类型相同，也可以用数组的方式表示，但采用结构进行描述，更贴近事物本质，从而增加了程序的可读性，使程序更易理解

- 结构比较适合用于描述具有多个属性的实体或对象

结构的嵌套定义

- 在我们的实际生活中，一个较大的实体可能由多个成员构成，而这些成员中有些又有可能是由一些更小的成员构成。
- 在学生信息中可以再增加一项：“通信地址”，它又可以再划分为：城市、街道、门牌号、邮政编码。

学号	姓名	通信地址				计算机	英语	数学	平均成绩
		城市	街道	门牌号	邮编				

结构的嵌套定义

- 由此，我们可以对其结构类型进行如下重新定义：

```
struct address{
 char city[10];
 char street[20];
 int code;
 int zip;
};
```

```
struct student{
     int num;
     char name[10];
     struct address addr;
     int computer, english, math;
     double average;
};
```

•在定义嵌套的结构类型时，必须先定义成员的结构类型，再定义主结构类型。

计算学生平均成绩

输入n个学生的成绩信息，计算并输出每个学生的个人平均成绩。

```
struct student{
   int num;
   char name[10];
   int computer, english, math;
   double average;
};
```

```
int main(void)
{   int i, n;
    struct student s1;   /* 定义结构变量 */
    printf("Input n: ");
    scanf("%d", &n);
    printf("Input the student's number, name and course scores\n");
    for(i = 1; i <= n; i++){
       printf("No.%d: ", i);
       scanf("%d%s%d%d%d",&s1.num,s1.name,&s1.math,&s1.engli
        sh,&s1.computer);
       s1.average = （ s1.math + s1.english + s1.computer) / 3.0;
       printf("num:%d, name:%s, average:%.2lf\n", s1.num, s1.name, s
        1.average);
    }
    return 0;
}
```

为什么s1.name前面没有“&”

结构变量的定义和初始化

- 在C语言中定义结构变量的方式有三种：

1. 单独定义：先定义一个结构类型，再定义一个具有这种结构类型的变量

```
struct student{
   int num;                          /* 学号 */
   char name[10];                    /* 姓名 */
   int computer, english, math;      /* 三门课程成绩 */
   double average;                   /* 个人平均成绩 */
};
struct student s1,s2;
```

结构变量的定义和初始化

2. **混合定义**：在定义结构类型的同时定义结构变量

```
struct student{
   int num;                          /* 学号 */
   char name[10];                    /* 姓名 */
   int computer, english, math;      /* 三门课程成绩 */
   double average;                   /* 个人平均成绩 */
}s1, s2;
```

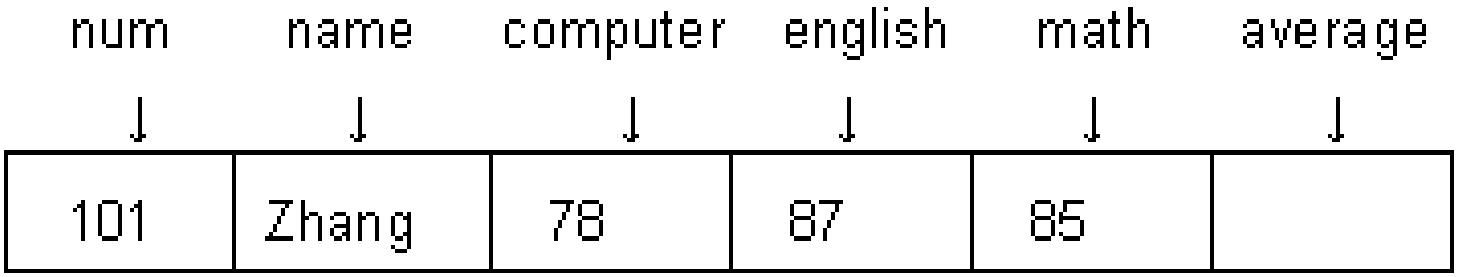
3. **无类型名定义**：在定义结构变量时省略结构名

```
struct {
   int num;                          /* 学号 */
   char name[10];                    /* 姓名 */
   int computer, english, math;      /* 三门课程成绩 */
   double average;                   /* 个人平均成绩 */
} s1, s2;
```

结构变量的定义和初始化

- 结构变量的初始化

```
struct student s1 = {101, "Zhang", 78, 87, 85};
```



结构变量的使用

1. 结构变量成员的引用

- 在C语言中，使用结构成员操作符“.”来引用结构成员，格式为：

  结构变量名 . 结构成员名

```
s1.num = 101;
strcpy(s1.name, "Zhang");
nest_s1.addr.zip = 310015;
```

结构变量的使用

2. 结构变量的整体赋值

- **具有相同类型的结构变量可以直接赋值**。赋值时，将赋值符号右边结构变量的每一个成员的值都赋给了左边结构变量中相应的成员。

```
struct student s1 = {101, "Zhang", 78, 87, 85}, s2;
s2 = s1;
```

学生类型

- 定义一个student结构类型，其中包含学号、姓名、使用的手机、3门课程成绩及总分，然后在主函数中创建该类型数组变量，计算每个学生的总分，并把总分放在成绩数组的第4个元素位置，最后显示每个学生信息。

- 程序首部定义了结构类型struct student，其中的成员分别代表学生的基本信息

```
struct mobile{              /*定义mobile结构类型*/
   char brand[20];
   char color[20];
   double thickness;
};
struct student{             /*定义student结构类型*/
   int id;
   char name[10];
   struct mobile phone;          /*包含mobile类型数据项*/
   int score[4];              /*包含数组类型数据项*/
};
```

```
void main ( )
{
        /*定义student类型的数组并初始化*/
         struct student s[3]={{10,"张三",{"iphone","black",7.6},{70,78,67}},
                              {11,"李四",{"nokia","red",12.1},{80,88,77}},
                              {12,"王五",{"samsung","blue",8.5},{70,98,67}}
        };
        int i,j;
        for(i=0;i<3;i++){      /*计算每个学生的总分，并放在成绩数组第4个元素位置*/
                for(j=0;j<3;j++)
                              s[i].score[3]+=s[i].score[j];
        }
        printf("id\tname\tbrand\tcolor\tthick\ts1\ts2\ts3\tsum\n");
        for(i=0;i<3;i++)       /*显示每个同学的信息*/
                printf("%d\t%s\t%s\t%s\t%3.1f\t%d\t%d\t%d\t%d\n",s[i].id,s[i].name,
                              s[i].phone.brand,s[i].phone.color,s[i].phone.thickness,
                              s[i].score[0],s[i].score[1],s[i].score[2],s[i].score[3]);
}
```

结构变量作为函数参数

- 如果一个C程序的规模较大，功能较多，必然需要以函数的形式进行功能模块的划分和实现
- 如果程序中含有结构数据，则就可能需要用结构变量作为函数的参数或返回值，以在函数间传递数据。

特点：可以传递多个数据且参数形式较简单

缺点：对于成员较多的大型结构，参数传递时所进行的结构数据复制使得效率较低

输入n个学生的成绩信息，计算并输出每个学生的个人平均成绩。

```
double count_average(struct student s)
{
    return (s.math + s.english + s.computer) / 3.0;
}
```

```
int main(void)
{
    int i, n;
    struct student s1;  /* 定义结构变量 */
    printf("Input n: ");
    scanf("%d", &n);
    printf("Input the student's number, name and course scores\n");
    for(i = 1; i <= n; i++){
       printf("No.%d: ", i);
    scanf("%d%s%d%d%d",&s1.num,s1.name,&s1.math,&s1.english,&s1.computer);
       s1.average = count_average (s1);
       printf("num:%d, name:%s, average:%.2lf\n", s1.num, s1.name, s1.average);
    }
    return 0;
}
```

```
#include <stdio.h>
struct date{      /*定义date结构类型*/
        int year;
        int month;
        int day;
};
void sub(date d){
        d.year--;              /* 修改形参的值 */
        d.month--;             /* 修改形参的值 */
        d.day--;               /* 修改形参的值 */
}
void main ( )
{
        date d={2013,3,3};
        sub(d);   /* 结构类型做函数的实参 */
        printf("%d-%d-%d\n", d.year,d.month,d.day);
                  /* 显示实参的值，看是否发生了变化 */
}
```

运行结果

2013-3-3

结构类型作函数的返回值

```
#include <stdio.h>
struct date{                    /*定义date结构类型*/
        int year;
        int month;
        int day;
};
date sub(date d){  /* 结构类型作为函数的形参和返回值类型 */
        d.year--;
        d.month--;
        d.day--;
        return d;  /* 返回结构类型 */
}
void print(date d){
        printf("%d-%d-%d\n", d.year,d.month,d.day);
}
void main ( )
{
        date d={2013,3,3};
        date d1=sub(d);        /* 调用返回结构类型的函数 */
        print(d);
        print(d1);
}
```

运行结果：

2013-3-3

2012-2-2

学生成绩排序

例 输入n（n<50）个学生的成绩信息，按照学生的个人平均成绩从高到低输出他们的信息。

```
struct student students[50], temp;    /* 定义结构数组 */
double count_average(struct student s); /* 函数,计算平均成绩 */

/* 输入 */
...
students[i].average = count_average( students[i] );
```

选择法排序 (程序段)

```
for(i = 0; i < n-1; i++){
        index = i;
        for(j = i + 1; j < n; j++)
           if(a[j] < a[index])      index = j;
        temp = a[index];
        a[index] = a[i];
        a[i] = temp;
}
```

```
/* 结构数组排序，选择排序法 */
 for( i = 0;  i < n-1;  i++ ){
    index = i;
    for (j = i+1;  j <n;  j++ )
      if (students[j].average > students[index].average)   /* 比较平均成绩*/
          index = j;
    temp = students[index];          /* 交换数组元素 */
    students[index] = students[i];
    students[i] = temp;
 }
 /* 输出排序后的信息 */
 printf("num  t name  t average  n");
 for (i = 0;  i < n;  i++ )
    printf("%d  t%s  t %.21f  n", students[i].num, students[i].name, students[i].average);
```

结构数组操作

- 结构数组的定义方法与结构变量类似

```
struct student students[50];
```

结构数组students，它有50个数组元素，从students[0]到students[49]，每个数组元素都是一个结构类型struct student的变量

结构数组操作

- 结构数组的初始化

```
struct student students[50] = {
{ 101,"zhang", 76, 85, 78 }, {102, "wang",
 83, 92, 86} };
```

students[0]	101	Zhang	76	85	78	
students[1]	102	Wang	83	92	86	
…	…	…	…	…	…	
students[9]						

结构数组操作

- 结构数组元素的成员引用 ，其格式为：

  结构数组名[下标] . 结构成员名

- 使用方法与同类型的变量完全相同：

```
students[i].num = 101;
strcpy(students[i].name, "zhang");
students[i] = students[k]
```

修改学生成绩

例　输入n (n<50) 个学生的成绩信息，再输入一个学生的学号、课程以及成绩，在自定义函数中修改该学生指定课程的成绩。

```
int main(void)
{
    int course, i, n, num, pos, score;
    struct student students[50];   /* 定义结构数组 */
    … /* 输入n个学生信息 */
    … /* 输入待修改学生信息 */
    /*调用函数，修改学生成绩*/
    pos = update_score(students, n, num, course, score);
    … /*输出修改后的学生信息*/ …
}
```

```
/* 自定义函数，修改学生成绩 */
int update_score(struct student *p, int n, int num, int course, int score)
{
    int i,pos;
    for(i = 0; i < n; i++, p++)          /* 按学号查找 */
        if(p->num == num)
            break;
    if(i < n)          /* 找到，修改成绩 */
    {
      switch(course){
            case 1: p->math = score; break;
            case 2: p->english = score; break;
            case 3: p->computer = score; break;
      }
      pos = i;                    /* 被修改学生在数组中的下标 */
    }
    else      /* 无此学号 */
       pos = -1;
    return pos;
}
```

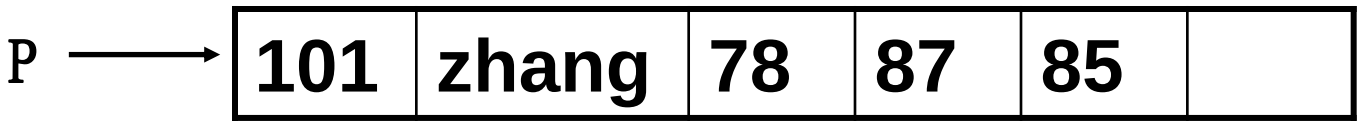
结构指针的概念

- 指针可以指向任何一种变量，而结构变量也是C语言中的一种合法变量，因此，指针也可以指向结构变量，这就是结构指针。

- 结构指针就是指向结构类型变量的指针

结构指针的概念

```
struct student s1 = {101, "zhang", 78, 87,
  85}, *p;
p = &s1;
```



结构指针的概念

- 结构指针的使用
- (1) 用*p访问结构成员。如:

```
(*p).num = 101;
```

- (2) 用指向运算符“->”访问指针指向的结构成员。如：

```
p->num = 101;
```

当p指向结构变量s1时，下面三条语句的效果是一样的：

```
s1.num = 101;
(*p).num = 101;
p->num = 101;
```

结构指针作为函数参数

- 结构指针的操作是非常灵活的，如果将结构指针作为函数的参数，可以完成比基本类型指针更为复杂的操作。
- 例

```
main:
pos = update_score(students, n, num, course, s
  core);
```

自定义函数：

```
int update_score(struct student *p, int n, int
  num, int course, int score)
```

函数update_score运行完毕返回主函数后，主函数中的结构数组students中的值已被修改

结构指针作为函数参数

- 结构数组名students作为函数参数时，其实质就是结构指针作为函数参数，因为数组名代表数组的首地址。

```
void new_student(struct student students[ ] );
void search_student(struct student students[ ], in
  t num);
void output_student(struct student students[ ]);
```

- 与结构变量作为函数参数相比，用结构指针作为函数参数的效率更高，因而是更佳的选择。