高级语言程序设计

第10章　结构、联合、枚举

内容提要

- **结构**
- 联合*
- 枚举*
- 链表*

- 基本数据类型：整型、实型和字符型
- 由基本数据类型构造而成的数组和指针类型

内容提要

结构体

- 结构体的定义
- 结构体变量
- 结构体指针
- 结构体数组
- 向函数传递结构体

内容提要

某学校学生成绩表

学号	姓名	性别	年龄	成绩	家庭住址
1	小张	男	17	90	江苏
2	小王	男	18	78	安徽
3	小李	女	17	89	山东
4	小赵	女	18	78	广东
5	……				
6	……				

结构

- 编程时，在描述或者表征某个对象时，若对象同时包含多个属性和特征，应如何描述呢？如学生对象，可能同时包含姓名、学号、性别和成绩等多方面的信息。
- 利用多个基本数据类型表示该数据对象。如用整型变量表示学号，字符数组表示姓名，字符型变量表示性别（'F'为女，'M'为男），实型变量表示成绩。

id	name	sex	age	score	addr
1	小张	M	17	90.0	江苏

结构

- 如何同时表示多个学生对象呢？
- 假设一共有10个学生，考虑使用数组进行刻画：

```
int ID[10];                    /* 学号数组 */
char Name[10][20];             /* 姓名数组，每行对应一位学生姓名 */
char Sex[10];                  /* 性别数组 */
double Score[10];              /* 成绩数组 */
```

- 将每个学生对象的信息分开表示和存储，类似于将机器每个零件拆开后分开保存，缺乏信息描述的完整性。

结构

- 对于这类数据，C语言提供了结构体机制来进行描述。
- 结构体类型的语法定义：

```
struct 结构类型名
{
    类型1    成员1;
    类型2    成员2;
    ……
    类型n    成员n;
};
```

注意：结构体定义的大括号后面一定要加上一个;号，否则编译时会报语法错误。

结构体定义说明

- “**struct**”是定义结构体的关键字，“结构类型名”是用户给新类型命名的名称。
- 大括号里包含的若干个变量，称为结构体的**成员**。每一个成员分别是用户用以描述对象的一个属性，各成员的类型任意，可以相同，也可以不同。
- 结构体类型定义之后的**分号不能省略**。
- 该定义结束后，“**struct 结构类型名**”就作为了一种新类型名。

结构体定义举例

- 示例1：日期类型的定义。
  - 一个日期由年、月、日组成，因此可定义一个日期类型如下：

```
struct Date
{
    int year;          /*年*/
    int month;         /*月*/
    int day;           /*日*/
};
```

结构体定义举例

- 示例2：学生类型的定义。
  - 假设学号、姓名、性别、一个成绩等信息可以较完整的表示一个学生的信息，则可以定义学生类型如下：

```
struct Student
{
    int ID;                 /*学号*/
    char Name[20];          /*姓名*/
    char Sex;               /*性别*/
    double Score;           /*成绩*/
};
```

结构体类型的嵌套

- 结构体类型中的成员可以是任何类型。
- 如果在struct Student类型中增加一个成员birthday表示学生的生日，则可修改为

```
struct Student
{
    int ID;                          /*学号*/
    char Name[20];                   /*姓名*/
    struct Date birthday;            /*生日，结构体类型*/
    char Sex;                        /*性别*/
    double Score;                    /*成绩*/
};
```

结构体嵌套

结构体类型的别名

- 用typedef为结构体起别名的方法
  - 方法一：先定义结构体类型，再使用typedef 语句
  - 语法

    typedef 原类型名 新类型名;

  - 如

    typedef struct Date Date;

    typedef struct Student Student;

以后再需要写日期类型时，可以省略struct

结构体类型的别名

- 用typedef为结构体起别名的方法
  - 方法二：定义结构体类型的同时给出其别名

```
typedef   struct Date
{
  int year;
  int month;
      int day;
} Date;                    /*这里的Date为别名*/
```

  - 使用别名的目的是使程序更为简洁。

小结

注意：结构体一般定义在main函数之外，是全局变量，方便别的函数访问。

结构体变量

- 结构体变量的定义
- 结构体变量的赋值和访问等

结构体变量

- 当定义一个结构体类型后，从本质上来说，它的使用与int、char一样，可以用于定义变量，也可以与指针、数组等结合，定义更复杂的结构体数组、结构体指针等。
- 结构体变量的定义：

结构体类型名　变量名;

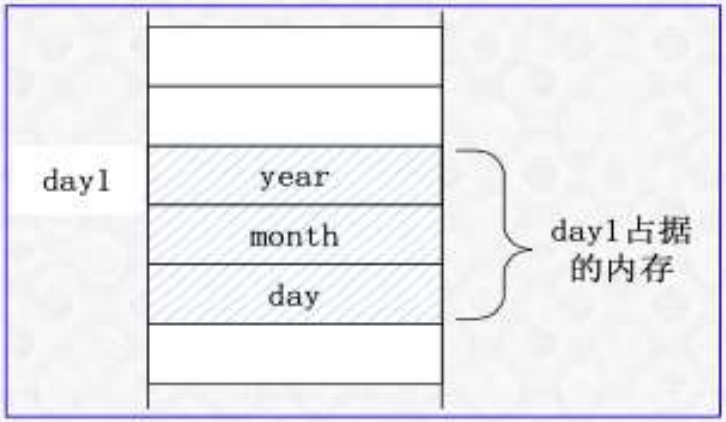
如　　struct Date　day1;

或者 Date　day1; (别名)

结构体变量

Date day1占据的内存

- 一个结构体变量占据的内存空间至少是该结构体所有成员占据内存的总和，且可能会占据更大的空间。



具体占据多少字节内存空间：

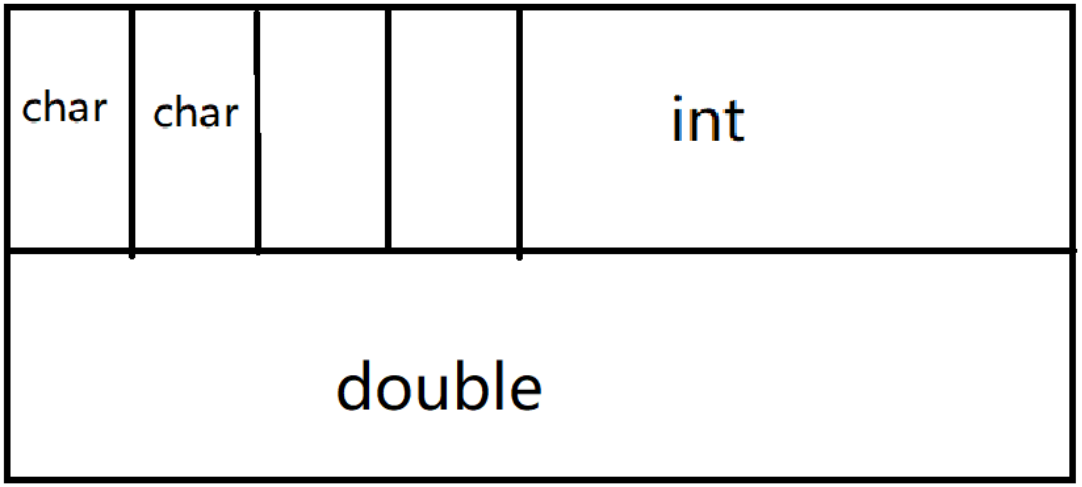
操作系统　　编译器

用sizeof来获得结构体的大小：
sizeof (struct Date)
或 sizeof (Date)

结构体变量


```
void main()
{   struct st
    {
        char a;
        char b;
        int c;
        double d;
    } p;
    printf("%d\n", sizeof(struct st));

}
```



结构体变量

- 还可在定义结构体类型的同时定义该类型的变量。语法形式：

```
struct 结构类型名
{
    类型1    成员1;
    类型2    成员2;
    ……
    类型n    成员n;
}变量名;
```

结构体变量

- 例如

```
struct Date
{
    int year;          /*年*/
    int month;         /*月*/
    int day;           /*日*/
}day2;
```

其中结构体类型名称“Date”也可以省略，从而直接定义结构体类型的变量。

结构体变量的赋值

- (1)定义的时候直接初始化，如：

```
Date day1 = { 2014, 11, 30};
Date day2 = { 2015, 1};
    /* 相当于Date day2 = { 2015, 1, 0}; */
Date day3 = day1;
   /* 用day1给day3初始化，day1已被赋值*/
```

- (2)定义后赋值

```
Date day4;
day4 = day2;
```

结构体变量的赋值

- (3) 给每个成员依次赋值，访问结构体变量中的成员时需使用运算符“.”

  结构变量名.成员名

  例如，给变量day4赋值：

  day4.year = 2014;

  day4.month = 12;

  day4.day = 1;

  存在结构体嵌套时，可以使用多个“.”。

例10-1

- 例10-1代码见课本。
- 说明
  - 存取、读写结构体变量时，都只能按成员进行依次操作，不能进行整体操作。

```
scanf( "%d%s%d%d%d%c%lf", &s1 );
/* 错误：不能整体输入 */
s4 = { 1004, "Liu", { 1992, 7, 5 }, 'F', 80 };
/* 错误：除初始化外，不能整体赋值 */
printf( "%d %s %d.%d.%d %c %lf\n", s1);
/* 错误：不能整体输出 */
```

例10-1

- 说明
  - 结构体Student中的成员birthday也是结构体类型，因此对birthday进行访问时使用了两层“.”运算符，如s1.birthday.year、s1.birthday.month等。
  - 从键盘读入数据时，s2.ID、s2.birthday.year、s2.sex、s2.score等前面均需加上“&”运算符，表示取这些成员的地址，s2.name是字符数组，本身就代表数组首地址，因此无需加上。

小结

- 结构体变量的定义
- 结构体变量的赋值的三种方式
- 访问结构体成员的“.”运算符

设置

1、对于一个结构体变量，系统分配的存储空间至少是_____。

A 第一个成员所需的存储空间

B 最后一个成员所需的存储空间

C 占用空间最大的成员所需的存储空间

D 所有成员存储空间的总和

提交

设置

2、已有以下定义，则正确的表达式是_____。

```
struct AA
{    int m,n;
}aa;
```

A AA.m = 10;

B AA bb = { 10, 5 };

C AA.aa.m = 10 ;

D aa.m = 10;

提交

设置

3、以下定义不正确的是_____。

A struct AA {int m,n;};

B struct AA {int m,n;} aa;

C struct {int m,n;};

D struct {int m,n;} aa;

提交

结构体指针

- 结构体指针的定义
- 结构体指针的访问等

结构体指针

- 结构体指针就是指基类型为结构体的指针，其定义方式如下：

结构体类型名 * 指针变量名;

Date *p;

- 通过结构体指针访问结构体成员的方式为：

结构指针 -> 结构成员

p-> year = 2018;

当然，从语法上来说，“(*结构指针). 结构成员”的形式也是可以的，但不推荐。

例10-2主要代码


```
。。。。。。                                    /*结构体定义*/
int main( )
{
    Student s1, *p;
    p = &s1;                                   /* 直接赋值 */
    s1.ID = 2001;                              /* 通过指针赋值 */
    strcpy( p->name, "Liang" );
    p->birthday.year = 1978;
     。。。。。。
    p->sex = 'M';
    p->score = 100;
    printf( "%d %s %d.%d.%d %c %.2f\n", p->ID, p->name, p->birthd
    ay.year,  p->birthday.month, p->birthday.day, (*p).sex, (*p).score );
    return 0;
}
```

输出结果为：2001 Liang 1978.4.20 M 100.00

例10-2主要代码

思考题：

结构体中，“.”运算符和“->”运算符分别在什么情况下使用呢？

```
Student s1, *p;  p = &s1;
s1.ID = 2001;          /* 结构体变量用点运算符“.”访问成员 */
p->ID = 1978;          /* 结构体变量指针“->”访问成员*/
```

结构体数组

- 结构体数组的定义
- 结构体数组的两种访问方法等

结构体数组

- 结构体数组：元素类型为结构体的数组

结构体类型名　数组名[常量表达式];

Student　sa[30];

- 访问数组某个元素的成员：

(1) 下标法：结构体数组名[下标].结构成员

表示数组中的元素，相当于变量，通过变量访问成员用“.”运算符。

sa[i].ID = 1001;

结构体数组

- 结构体数组：元素类型为结构体的数组

结构体类型名　数组名[常量表达式];

Student　sa[30];

- 访问数组某个元素的成员：

(2) 指针法：(结构体数组名+下标)->结构成员

数组名是数组的首地址，是常指针，加上i，相当于指向下标为i的元素的地址，再通过指针的“->”运算符来访问成员。

(sa+i)->ID = 1001;

(*(结构数组名+下标)).结构成员

例10-3主要代码


```
。。。。。。                                    /*结构体定义*/
int main( )
{   Student st[3] = {。。。。。。 };
    int i;  /* 第一种方式访问结构体成员 */
    st[2].ID = 1003; strcpy( st[2].name, "Li" ); st[2].birthday.year = 1993;
    st[2].birthday.month = 7; st[2].birthday.day = 22; st[2].sex = 'M'; st[2].score = 65;
    /* 第二种方式访问结构体成员 */
    for ( i=0 ; i<3 ; i++ )
    {
    printf( "%d %s %d.%d.%d %c %f\n", (st+i)->ID, (st+i)->name, (st+i)->birthday.year, (st+i)->birthday.month, (st+i)->birthday.day, (st+i)->sex, (st+i)->score );
    }
    return 0;
}
```

例10-3

- 说明
  - 本例中定义了一个含有3个元素的数组st，但只初始化了前2个元素的值。因此，第3个元素的值被全部置为0。
- 输出结果为：

```
1001 Zhang 1992.5.21 F 83.000000
1002 Wang 1993.6.18 M 66.000000
1003 Li 1993.7.22 M 65.000000
```

例10-3

思考题：

若在例10.3中，定义一个结构体指针p，即：Student * p;
如何通过p来访问数组st呢？

4、已有以下定义，则不正确的是_____。

```
struct AA
{   int  m ;
    char *n;
} aa={10, "abc"},*p=&aa;
```

A *p->n

B p->n “abc”的地址

C *p.n 左操作数指向“struct”，使用“->”

D *aa.n

提交

5、struct person
{ int ID;
char name [20];
struct { int year, month, day; } birthday;
} Jerry;
将Jerry中的month赋值为10的语句为_____。

A Jerry.month=10;

B Jerry.birthday.month=10;

C person.Jerry.birthday.month=10;

D person.Jerry.month=10;

提交

设置

6、已有定义“struct { int m, n; } arr[2] = { { 11, 22 }, { 33, 44 } }, *ptr = arr;”，则表达式++ptr->m的值为_____，（++ptr）->m的值为_____。

A 12,33

B 11,33

C 11,34

D 12,34

成员选择（指针） -> 优先级高于 自增自减运算符

提交

访问结构体数组元素的方法

- 结构体数组就是元素类型为结构体的数组，其定义方式如下：

  结构体类型名 数组名[常量表达式];

- 通过数组来访问某个元素的成员时，可使用如下方法：

  方式一： 结构数组名[下标].结构成员

- 下列两种方式也是可以的，但是由于可读性较差，所以很少使用，尤其是最后一种。

结构体指针

- 方式二：（结构数组名+下标）-> 结构成员
- 方式三：（*（结构数组名+下标））.结构成员

向函数传递结构体

- **传递结构体成员**

如果只需传递结构体中的部分成员，则直接将该成员作为函数实参；函数的形参与实参中结构体成员同类型；属于简单的传值调用。

- **传递结构体变量**

可将结构体变量作为函数的实参，向函数传递完整的结构体内容；函数形参是同类型的结构体变量；属于传值调用。

- **传递结构体地址**

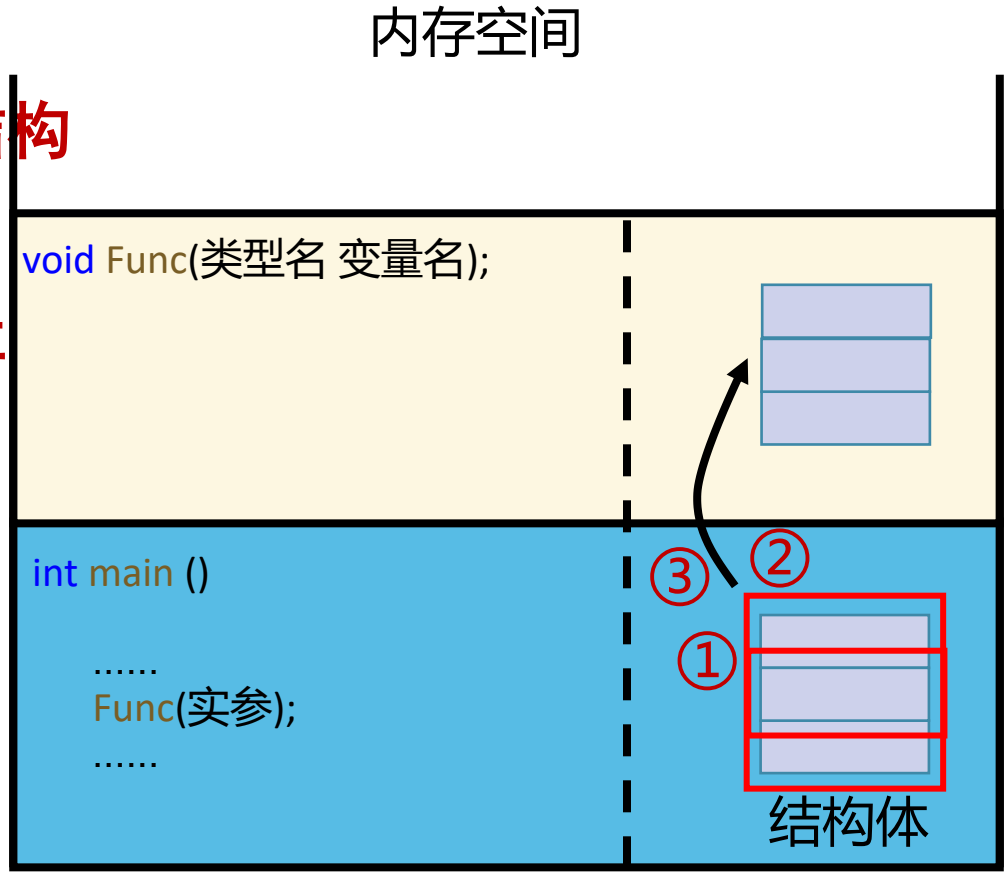
可将结构体指针或结构体数组的首地址作为函数实参，向函数传递结构体地址；函数形参为同类型的结构体指针或数组；属于传地址调用。

向函数传递结构体

① 向函数传递结构体的**单个成员**
- 复制单个成员的内容

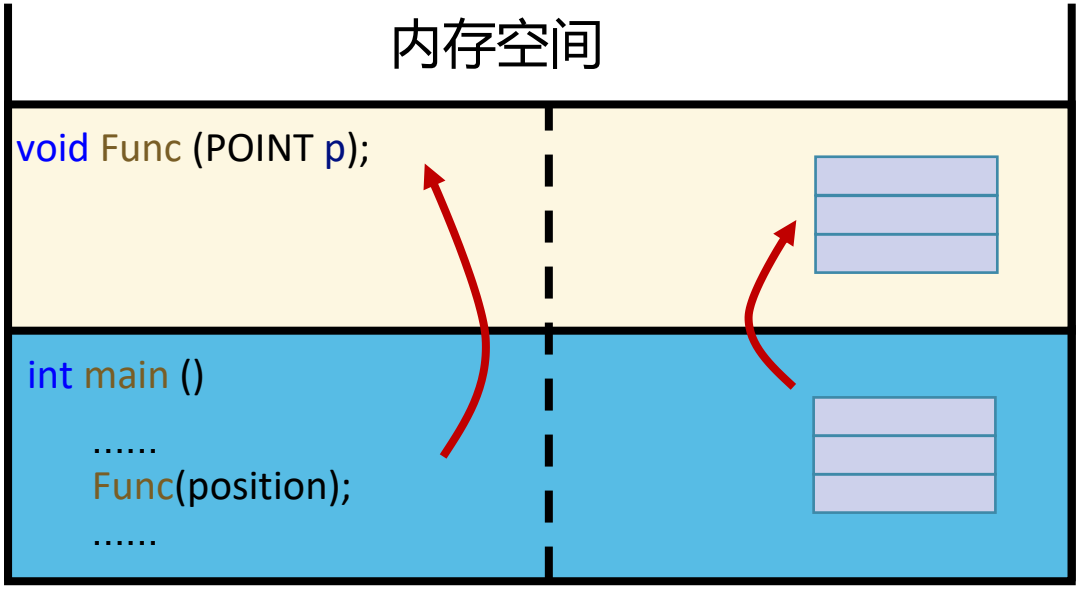
② 向函数传递结构体的**完整结构**
- 复制结构体的所有成员

③ 向函数传递结构体的**首地址**
- 仅复制一个地址值



结构体变量作函数参数


```
typedef struct point
{
    int x;
    int y;
    int z;
}POINT;
void Func(POINT p)
{
   p.x = 1;
   p.y = 1;
   p.z = 1;
}
```



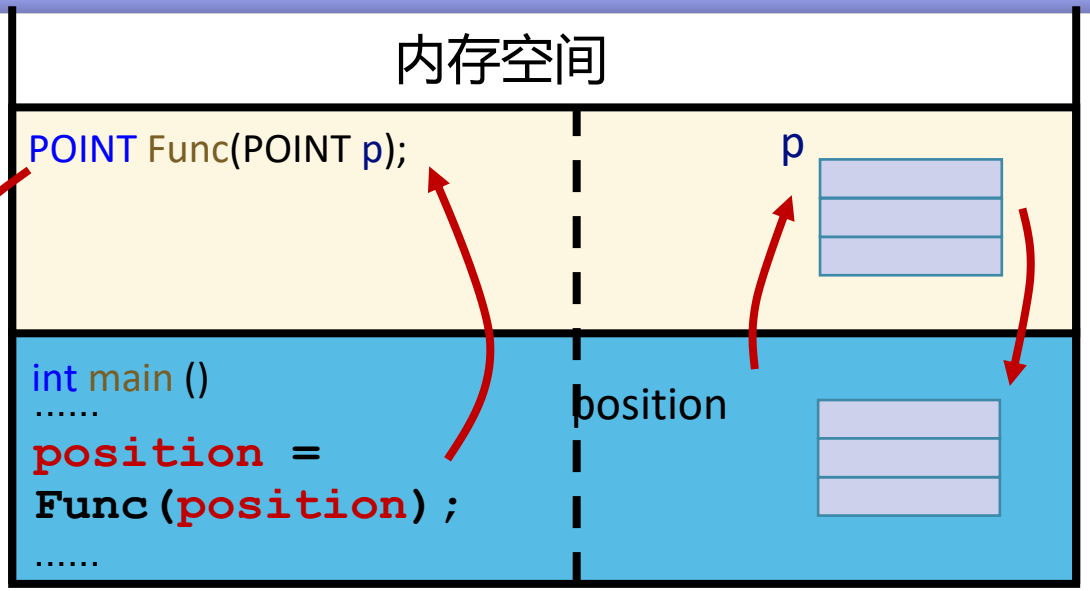

```
int main()
{
  POINT position = {0, 0, 0};
  printf("Before:%d,%d,%d\n",
          position.x, position.y, position.z);
  Func(position);
  printf("After:%d,%d,%d\n",
          position.x, position.y, position.z);
  return 0;
}
```

```
Before:0,0,0
After:0,0,0
```

复制结构体的所有成员给函数
函数对结构体内容的修改不影响原结构体

结构体变量做函数返回值


```
typedef struct point
{
    int x;
    int y;
    int z;
}POINT;
POINT Func(POINT p)
{
   p.x = 1;
   p.y = 1;
   p.z = 1;
   return p;
}
```



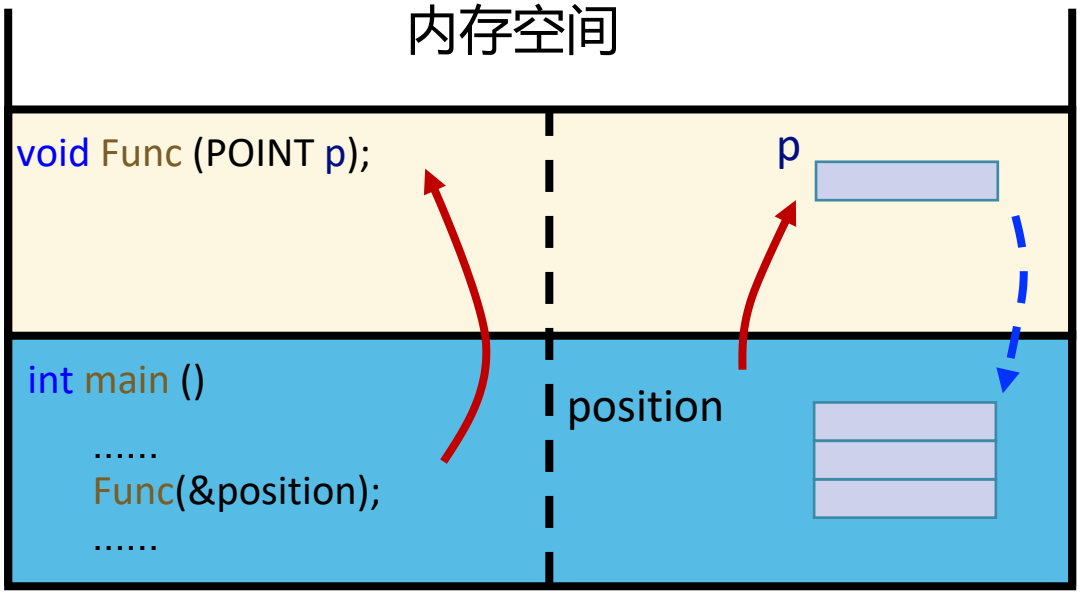

```
int main()
{
  POINT position = {0, 0, 0};
  printf("Before:%d,%d,%d\n",
          position.x, position.y, position.z);
  position = Func(position);
  printf("After:%d,%d,%d\n",
          position.x, position.y, position.z);
  return 0;
}
```

```
Before:0,0,0
After:1,1,1
```

返回结构体变量可得到修改后的结构体内容，但效率低

结构体指针作函数参数


```
typedef struct point
{
    int x;
    int y;
    int z;
}POINT;
void Func(POINT *pt)
{
   pt->x = 1;
   pt->y = 1;
   pt->z = 1;
}
```



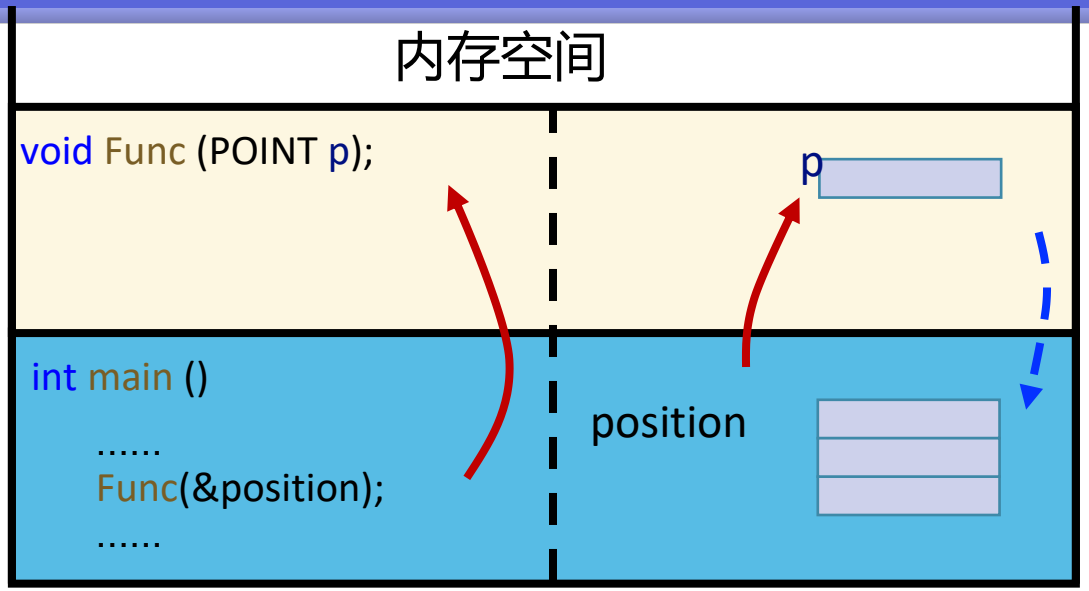

```
int main()
{
  POINT position = {0, 0, 0};
  printf("Before:%d,%d,%d\n",
         position.x, position.y, position.z);
  Func(&position);
  printf("After:%d,%d,%d\n",
         position.x, position.y, position.z);
  return 0;
}
```

向函数传递结构体变量的地址
函数对结构体的修改影响原结构体

```
Before:0,0,0
After:1,1,1
```

用const保护结构体指针指向的结构体


```
typedef struct point
{
    int x;
    int y;
    int z;
}POINT;
void Func(const POINT *pt)
{
   pt->x = 1;
   pt->y = 1;
   pt->z = 1;
}
```




```
int main()
{
  POINT position = {0, 0, 0};
  printf("Before:%d,%d,%d\n",
         position.x, position.y, position.z);
  Func(&position);
  printf("After:%d,%d,%d\n",
         position.x, position.y, position.z);
  return 0;
}
```

```
error: assignment of member 'x' in read-only object
error: assignment of member 'y' in read-only object
error: assignment of member 'z' in read-only object
```

例10-4主要代码


```
。。。。。。                                    /*结构体定义*/
renew_value1(st1);          /*传值调用方式*/
renew_value2(&st2);          /*传地址调用方式*/
。。。。。。

 void renew_value1( Student st )          /* 传值调用方式*/
{
   st.ID=1003;
   strcpy(st.name, "Jean");
   st.score=98;
}
void renew_value2( Student *pt )                    /* 传地址调用方式*/
{
   pt->ID=1004;
   strcpy(pt->name, "Dell");
   pt->score=98;
}
```

例10-4

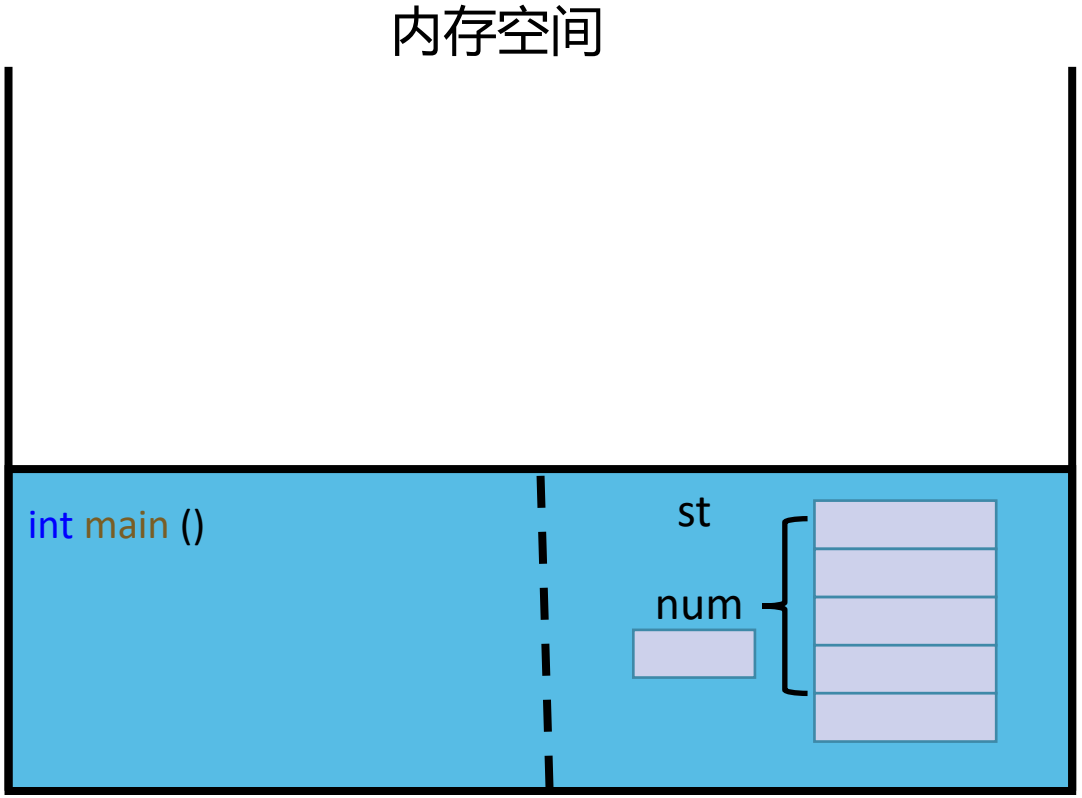
- 说明
  - 函数renew_value1形参为结构体类型的变量，实参为结构体类型的变量st1；传递方式为传值调用，因此，在函数内修改结构体成员的值，st1中成员值并没有发生改变；
  - 函数renew_value2的形参为结构体类型的指针，接收实参中结构体变量的地址&st2；传递方式为传地址调用，在函数内可利用指针修改st2中成员的值。

结构体应用

- 例10-5 以一个班级成绩排名为例，说明结构体在处理批量数据记录时的应用。

```
typedef struct Student
{
        int ID;
        char name[20];
        double score;
}STUDENT;
```

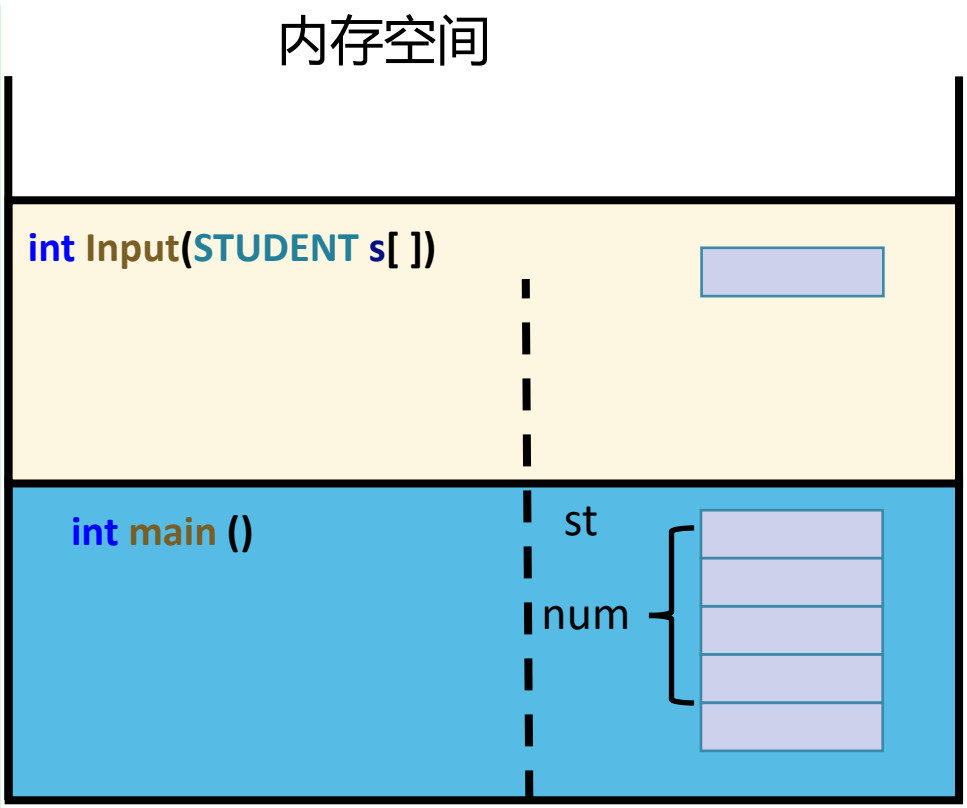
```
int main( )
{
        STUDENT st[10];
        int num;
        num = Input(st);
        Sort(st, num);
        Output(st, num); return 0;
}
```



结构体应用

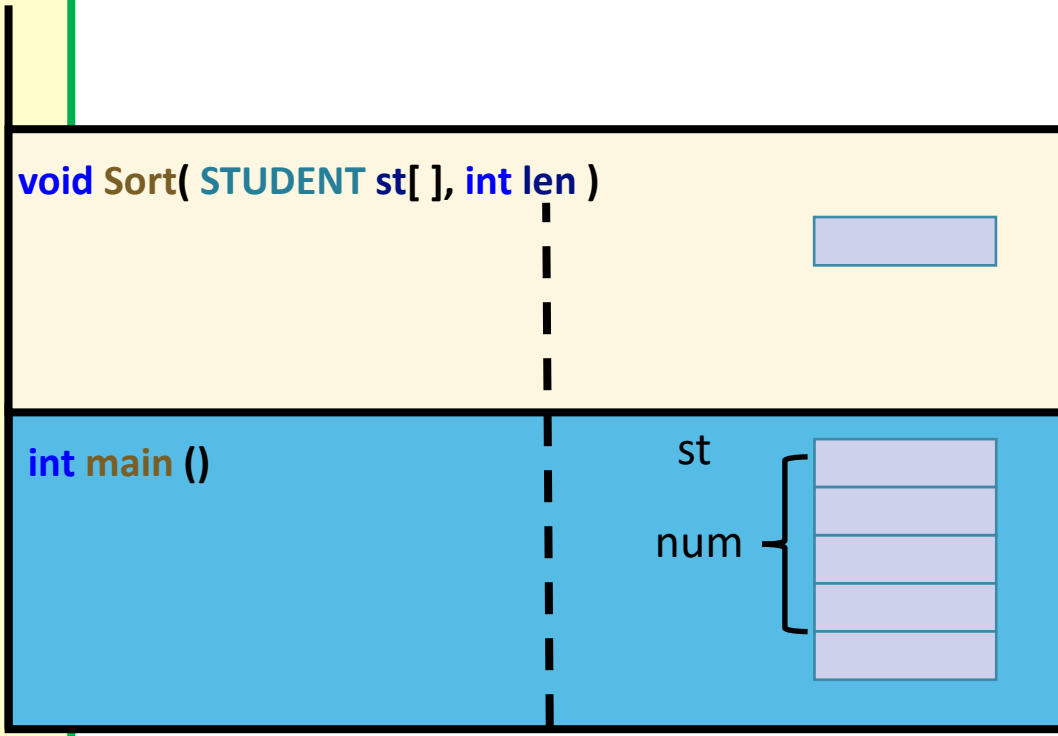
- 例10-5 以一个班级成绩排名为例，说明结构体在处理批量数据记录时的应用。

```
int Input( STUDENT s[ ] )
{
        int i, n;
        do
        {
                printf("Enter the sum of students: \n");
                scanf("%d", &n);
        } while ( n<=0 || n>10 );
        for ( i=0 ; i<n ; i++ )
        {
                printf("Enter %d-th student : ", i+1);
                scanf("%d%s%lf", &s[i].ID, s[i].name,
                                        &s[i].score);
        }
        return n;
}
```

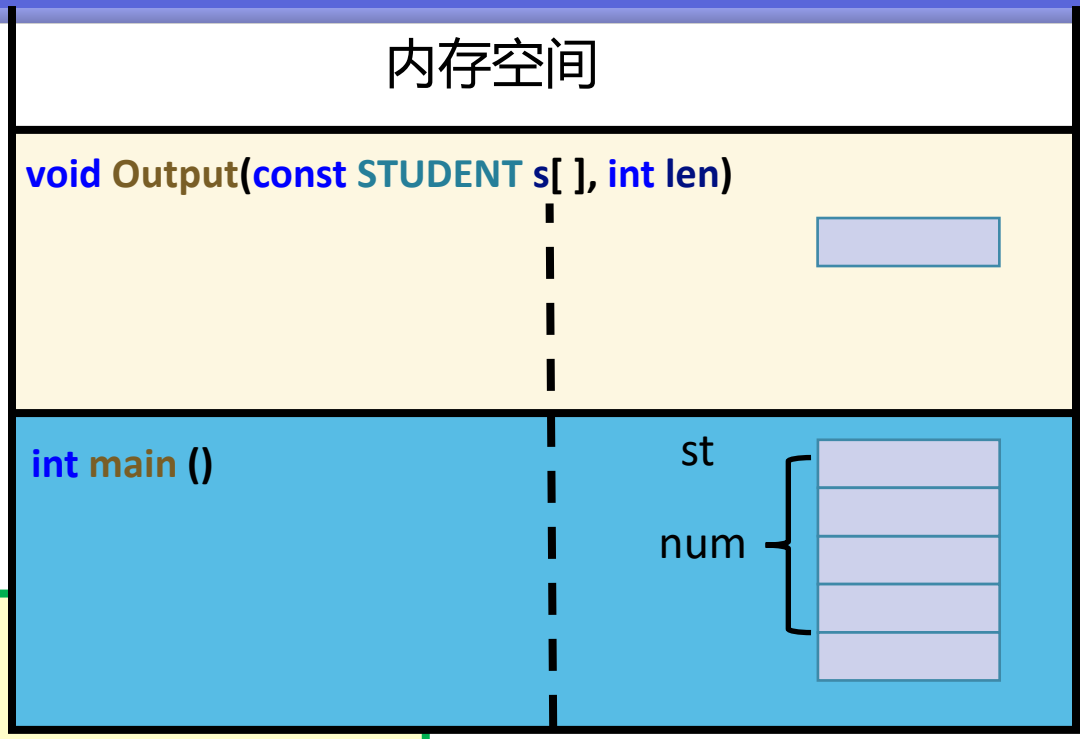


结构体应用


```
void Sort( STUDENT st[ ], int len )
{
        int i, k, index;
        STUDENT temp;
        for ( k=0 ; k < len-1 ; k++ )
        {
                index = k;
                for( i=k+1 ; i<len ; i++ )
                        if ( st[i].score > st[index].score )
                        index = i;
                if ( index != k )
                {
                        temp = st[index];
                        st[index] = st[k];
                        st[k] = temp;
                }
        }
}
```



结构体应用

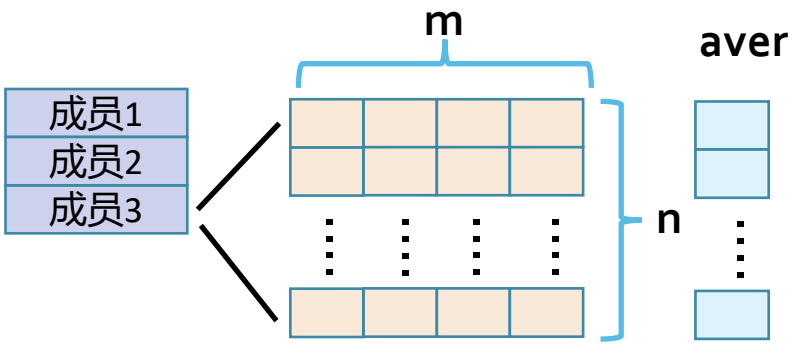



```
void Output(const STUDENT s[ ], int len )
{
        int i;
        printf("学号 姓名 成绩\n");
        for ( i=0 ; i<len ; i++ )
        {
                printf("%4d %-8s %.0f\n",
                                s[i].ID, s[i].name, s[i].score);
        }
}
```

结构体应用

- 每个学生录入的成绩增加两门，计算每个学生所有课程的平均分

```
typedef struct Student
{
        int ID;
        char name[20];
        int score[3];
}STUDENT;
```

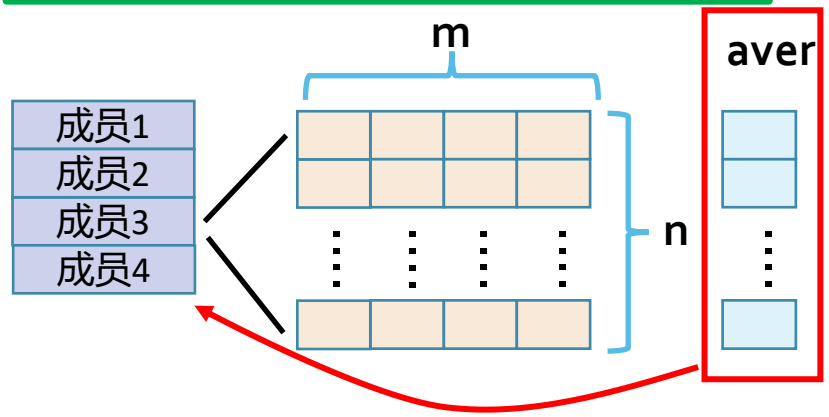



```
void AverScore(STUDENT s[], float aver[], int n, int m)
{
        int i, j, sum[10];
        for (i=0; i<n; i++)
        {
                sum[i] = 0;
                for (j=0; j<m; j++)
                {
                        sum[i] = sum[i] + s[i].score[j];
                }
                aver[i] = (float)sum[i]/m;
        }
}
```

结构体应用

- 改进版

```
typedef struct Student
{
        int ID;
        char name[20];
        double score[3];
        float aver;
}STUDENT;
```




```
void AverScore(STUDENT s[], int n, int m)
{
        int i, j, sum[10];
        for (i=0; i<n; i++)
        {
                sum[i] = 0;
                for (j=0; j<m; j++)
                {
                        sum[i] = sum[i] + s[i].score[j];
                }
                s[i].aver = (float)sum[i]/m;
        }
}
```

结构体应用

- 用结构体类型封装函数参数的好处是什么?
  - 精简参数个数
  - 使函数接口更简洁
  - 可扩展性好

小结

- 结构体数组
- 结构体指针
- 向函数传递结构体

利用指针和数组实现函数之间批量数据的共享。

- 结构体应用——学生成绩排名

输入理想的程序

输出快乐的人生

联合

- 联合类型定义的方式与结构体类型类似

```
union 联合名
{
    类型1    成员1;
    类型2    成员2;
    ……
    类型n    成员n;
};
```

其中，“union”是定义联合的关键字，“联合名”是用户给新类型命名的名称。各成员的类型可以相同，也可以不同。

联合

- 定义一种类型，用于课程成绩的存放。一门课程的成绩可能是整型、实型或者等级制，但对一门具体的课程而言，其分数类型应该是确定的，只有一种。因此，我们可以把成绩定义为联合类型。
- union Score

```
{
    int i;
    double d;
    char c;
};
```

联合变量

- 定义一个联合变量的语法如下：

  联合类型名 变量名;

  例如，union Score sc;

  其中，“union Score”为类型名，sc为新定义的变量。

- 与结构体类似，我们也可以为联合类型定义一个别名，以进行简化。例如：

  typedef union Score Score;

  Score sc;

联合变量

- sc的存储示意如图所示。注意，联合变量所有成员则共享同一段内存空间，其占有内存是所需内存最大的那个成员的空间。联合变量这种占有内存的方式，决定了它每次只能有一个成员起作用，也就是最后赋值的那个成员。联合类型的这些特征可以概括为：空间共享，后者有效。

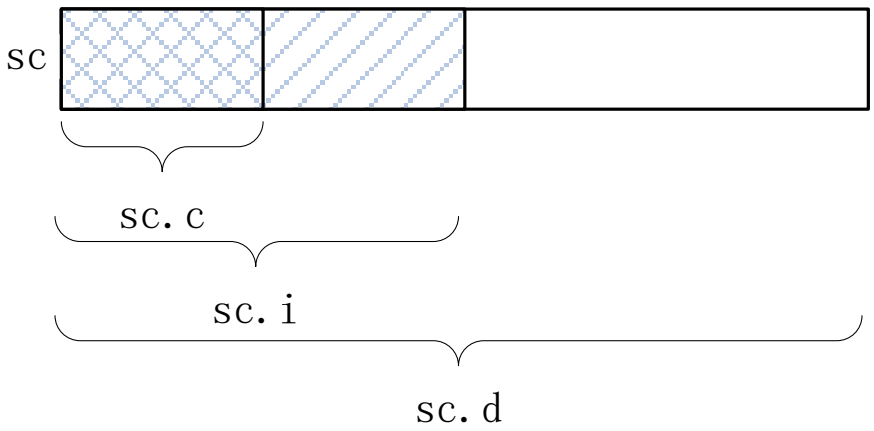


图10.8 sc存储示意图

联合变量

- 访问联合变量中的成员时，也使用 “.” 运算符：

  类型变量名.成员名

- 由于各成员的空间共享，联合变量的使用有一些特殊，它不能整体赋值和输出，在初始化时也只能初始化第一个成员。例：

```
sc.i = 88;                          /* 合法 */
sc.d = 78.5;                        /* 合法 */
sc = { 88,78.5,'A' };               /* 不合法 */
Score sc = {88} ;                   /* 合法 */
Score sc = { 88,78.5,'A' };         /* 不合法 */
```

例10-6

- 例10-6 代码见课本
- 说明
  - 从本例可以看出，sc的三个成员各占4 字节、8字节和1字节，但是sc总共的字节数只有8字节，为double型成员d的字节数。
  - 当对其中一个成员赋值后，其它两个成员也可以访问，但是输出的数据并不准确。

联合

- 联合还可以与结构体联合使用。
  - 例如在描述学生信息时，学生分为普通学生和转专业学生，每个学生只能处于一种状态。除姓名、成绩等信息外，如果是普通学生，只需了解学号信息，而转专业学生则需要原学号和原专业信息。

<table>
<tr><td colspan="5">学生信息（struct Student）</td></tr>
<tr><td colspan="3">转专业状态（union Transfer_state）</td><td rowspan="3">姓名</td><td rowspan="3">成绩</td><td rowspan="3">是否转专业标记</td></tr>
<tr><td>否</td><td colspan="2">是（struct Transfer）</td></tr>
<tr><td>学号</td><td>原学号</td><td>原专业</td></tr>
</table>

联合


```
struct Transfer{                    /*转专业学生信息结构体*/
        int ID_transfer;                /*学号*/
        char *major_orginal;            /*原专业*/
};
union Transfer_state{               /*学生状态信息联合*/
        int ID_normal;                  /*普通学生学号*/
        struct Transfer tansfer_inf;        /*转专业学生学号和原专业*/
};
struct Student{
        union Transfer_state ID;
        char name[10];
        int score;
        int if_transfer;                /*是否为转专业标记*/
};
```

枚举

- 枚举（enumeration）类型也是一种用户自定义类型。所谓“枚举”是一一列举之意，即它允许用户自定义一种数据类型，该类型具有有限的取值范围，可以逐一列举出来。

枚举

- 枚举类型定义的语法为：

  enum 枚举类型名 {枚举常量1，枚举常量2，…，枚举常量n};

  其中，“enum”是定义枚举的关键字，“枚举类型名”是用户给新类型命名的名称，枚举常量1、枚举常量2……枚举常量n是n个**常量**，称为**枚举元素**或**枚举常量**，它们表示该类型可取值的范围。

枚举

- 季节类型的定义：

  enum Seasons { Spring, Summer, Autumn, Winter };

  本例中，Seasons是一个新创建的枚举类型，该类型的取值范围只有4个，即Spring, Summer, Autumn, Winter 。

- C语言中，系统会为每个枚举元素对应一个默认整数值，通常从“0”开始，并顺次加1。

  例如，上例中， Spring, Summer, Autumn, Winter分别对应0、1、2、3。

枚举

如果要改变这种默认值，可以在定义时进行指定，如：

```
enum Season { Spring=4, Summer=1, Autumn, Winter };
```

这样，这4个枚举元素对应的整数就变为4、1、2、3。

枚举变量

- 定义一个枚举变量的语法如下：

  枚举类型名 变量名;

  例如，

  enum Season day;

  或者

  typedef enum Season Season;

  Season day;

- 在使用枚举变量时，通常它无法直接输入，赋值时可以赋以枚举类型数据和整型数据，输出时可以以整型方式输出，无法直接输出枚举常量。

例10-7

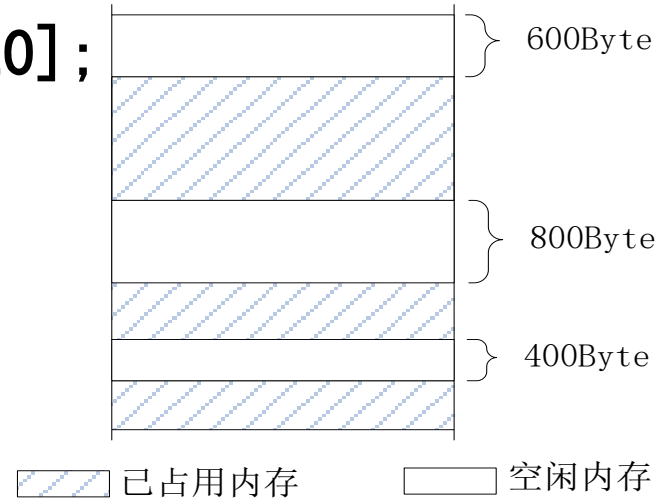
- 例10_7代码见课本。
- 说明
  - 本例中只对day数组第0个元素进行了初始化，后面三个元素都被初始化为0。
  - 对枚举变量赋值时，可以赋以枚举类型数据和整型数据。另外，从本例也可以看出，虽然枚举类型取值范围对应的整数只有1-4，但赋值超出该范围时（“day[2] = 6;”），系统并不会进行检查。
  - 枚举类型变量可以以%d形式输出，其它方式需要编写代码进行转换。

链表

- 对大批量的同类型数据进行处理时，常借助于数组来进行存储和相应的操作，比如添加、删除、查找、排序等等
- 数组优点：直观、方便
- 数组缺点：需要提前明确数组长度；

  需占有连续的内存空间，如int a[220]；

  插入和删除需要移动大量元素。



链表的概念

- **解决方案**：链表。
- **问题**：使用链表来存储数据时，这些数据就有可能散布在内存的各处。那么程序在处理时，如何找到下一个数据？又如何能确认所有数据都已处理完？

链表的概念

- 链表的基本构成单位是结点。一个结点又由两部分组成：数据部分和指针部分。

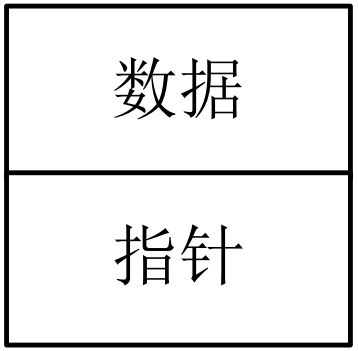


图10_5 结点示意图

链表的概念

- 一个链表就是由这样一系列的结点所“链接”而成

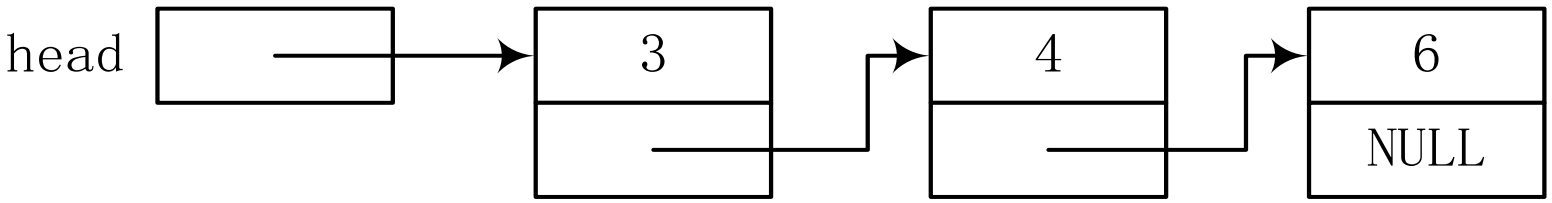


图10_6 链表示意图

链表的概念

- 一个结点至少包含两个内容：数据和指针。因此可以定义如下。

```
struct Node                    /* 结点的结构体类型定义 */
{
    int data;                  /* 结点的数据部分 */
    struct Node *next;         /* 结点的指针部分 */
};
```

- data成员用于存放数据，这里假定存放的是int型数据（可根据实际需要修改）
- next成员用于存放结点的地址，所以其类型是struct Node *。

例10-8

- 例10-8代码见课本。编程环境下演示。

例10-8说明

- 本例主要展示了两个功能：建立链表和打印链表
- 本例是链表建立的一个简单演示，只包含三个结点。在实际中，有可能要处理的是大批量的数据，并且数据的数量也可能在运行时动态发生变化，因此不可能采取本例中的方式，即事先为每个数据定义一个结点，再将它们链接起来。而是在需要存储数据的时候向系统动态申请内存。每增加一个数据，程序就申请一个结点大小的内存空间，将数据存放进去，并将其添至链表中

例10-8说明

- 在打印链表中的数据时，首先从head指针开始找到链表的第一个结点，打印其中的数据（“printf("%d ", p->data);”），并从这个结点中找到下一个结点的地址（“p = p->next;”），再重复这个过程，直至最后一个结点（p值变成'\0'）。在后续学习中，我们也将发现，存储链表首地址的head指针极为重要，它是整个链表的入口，我们对链表进行的绝大部分操作，都是从head指针开始入手的，并且在操作的过程中，也要时刻注意head指针的维护。

链表的基本操作

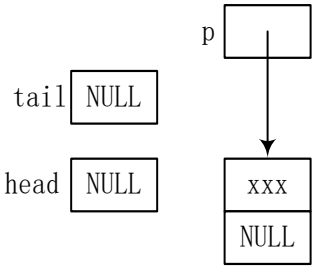
- 链表的基本操作
  - 建立（批量存入数据）
  - 打印（输出所有数据）
  - 删除（在批量数据中删除指定数据）
  - 插入（在批量数据中添加一个数据）

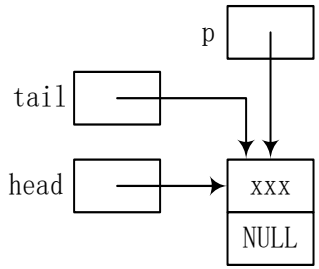
例10-9

- 例10-9代码见课本。
- 说明
  - 本例除main函数外，还包括Create、Print、Release这3个函数，功能分别是创建链表、打印链表和释放链表。
  - 在建立链表（Create函数）时，本例使用了“尾插法”，即向链表中添加新结点时，新结点总是添加在链表的末尾。

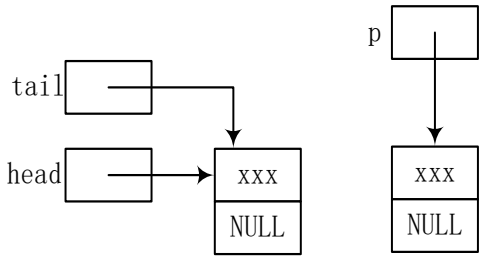
例10-9

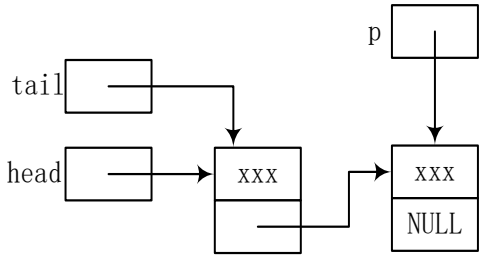
(a) 链表初始化

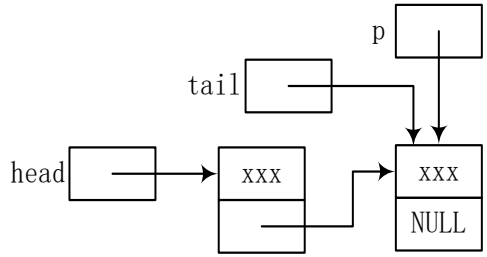




(b) 往链表中添加第一个结点







(C) 往链表中添加第二个结点

图10-7 链表建立过程

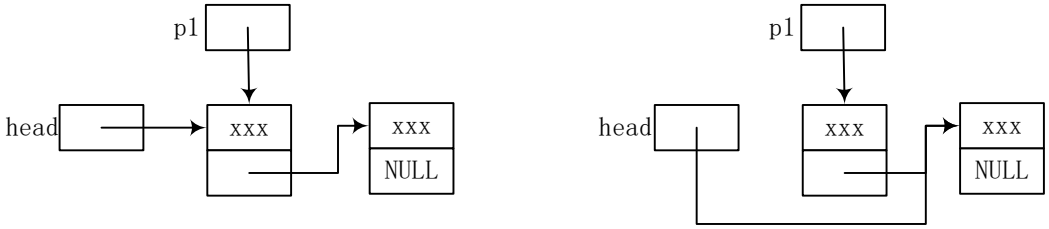
例10-9

- 说明
  - 链表打印（Print函数）的基本思想与例10-8相同，主要是增加了链表是否为空的判断。
  - 链表释放（Release函数）的作用是，当程序运行结束时，释放建立链表时所申请的内存空间。它的基本思想是：从第一个结点开始，首先保存该结点下一个结点的地址，再将该结点的内存空间释放。重复上述过程，直至链表结尾。
- 例10-9的思考题：修改Create函数，要求建立链表时，总是把新结点添加在链表的最前面。

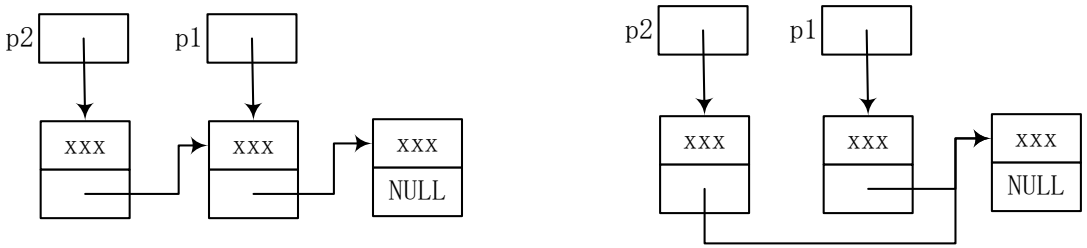
例10-10

- 在例10-9的基础上，增加一个函数Delete，实现数据的删除，代码见课本。编程环境下演示

例10-10



（a） 待删除结点为链表首结点



（b） 待删除结点非链表首结点

图10-7 结点删除过程

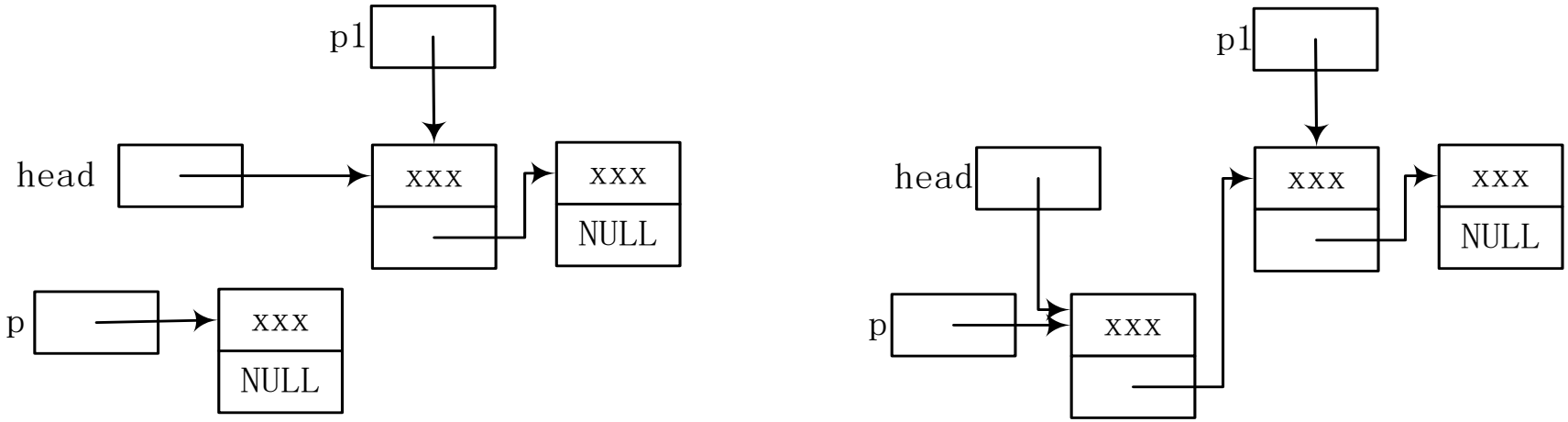
例10-11

- 假定链表中的数据是从小到大存放的，现要求在例10-9的基础上，增加一个函数Insert，实现向链表中插入数据的功能，并保持从小到大的次序不变。 代码见课本。

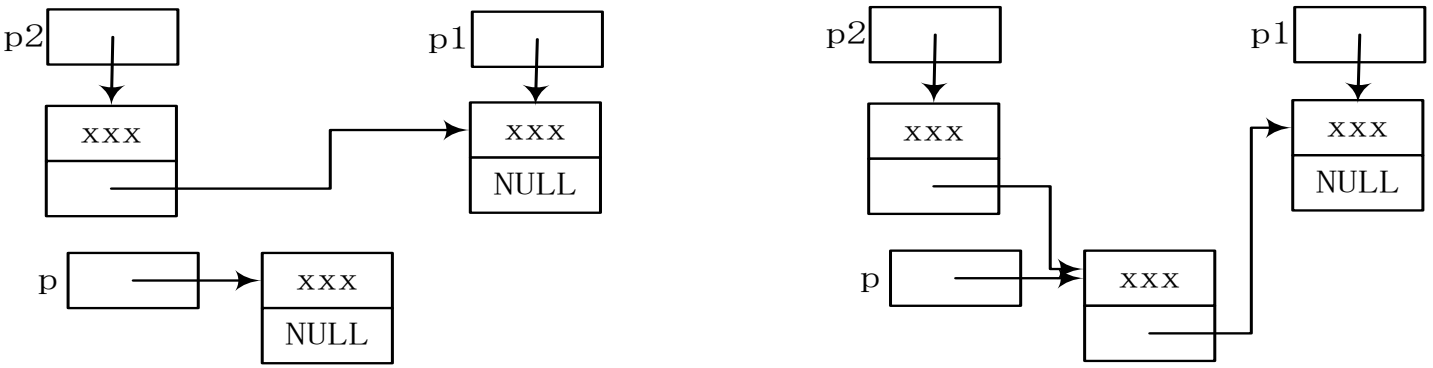
例10-11

- 说明：往链表中插入数据，一般需经三个步骤：生成结点、确定位置、插入结点。
  - 生成结点：为待插入数据申请一块结点大小的存储空间p，将数据存入结点的数据域，结点的指针域则赋以初值NULL。
  - 确定位置：确定新结点在链表中的插入位置，在函数中通过while循环实现。
  - 插入结点：将新结点插入到合适的位置，在函数中通过if~else语句及后续的“p->next = p1;”共同完成。

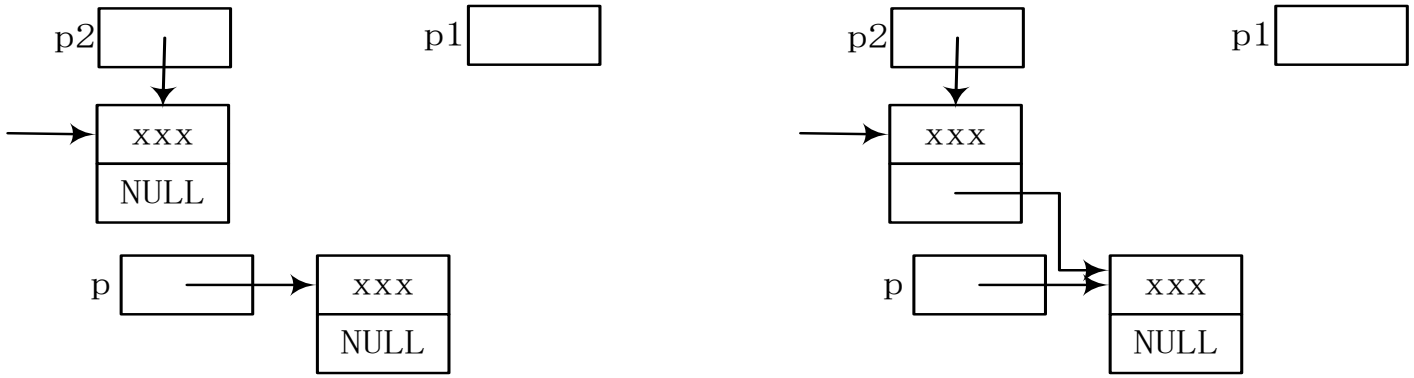
例10-11



(a) 插入位置在第一个结点之前



(b) 插入位置在两个结点之间



(c) 插入位置在末尾结点之后

图10-11 结点插入过程示意

本章小结

- 几种构造类型的定义：结构、联合、枚举
- 结构体变量的定义，以及结合指针、数组和函数的相关操作
- 用typedef为类型定义别名
- 了解链表的递归定义，以及一些相关的基本操作，如链表的建立、插入、删除、查找、遍历等

输入理想的程序

输出快乐的人生