



# 第七章 结构体

丁子君



# 目录

- 自定义数据类型的基本概念
- 结构体类型
- 结构体变量的定义及初始化
- 结构体变量的使用
- 结构体数组变量（结构体数组做函数参数）
- 结构体指针变量
- 枚举
- 用typedef声明新类型



## 7.1 自定义数据类型的基本概念

- C++语言有丰富的系统预定义的基本数据类型
  - 基本类型：整型、实型、字符型、指针……



# 衍生问题

学号	姓名	年龄	课程成绩
1	张一	18	80.5
2	李二	18	76
3	王三	19	90.5
4	陈四	17	88

➤ 每列的数据类型不同，如何表示该二维数据？

1. 逐一定义基本变量？



## 1. 逐一定义基本变量:

### • 学生1:

```
int num_1;  
char name_1[20];  
int age_1;  
float score_1;
```

### • 学生2:

```
int num_2;  
char name_2[20];  
int age_2;  
float score_2;  
.....
```

### • 学生n:

```
int num_n;  
char name_n[20];  
int age_n;  
float score_n;
```



1. 工作量巨大, 不现实

2. n个单变量, 修改不便, 无法循环处理



# 衍生问题

学号	姓名	年龄	课程成绩
1	张一	18	80.5
2	李二	18	76
3	王三	19	90.5
4	陈四	17	88

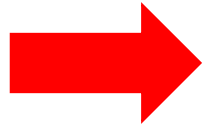
➤ 每列的数据类型不同，如何表示该二维数据？

2. 用数组表示？



2. 用数组表示:

```
int num [4];  
char name [4][20];  
int age [4];  
float score [4];
```



1. 数组表现的数据类型必须相同，无法体现每一个学生（独立个体）的信息
2. 操作（输入输出）时容易信息错位



## 7.1 自定义数据类型的基本概念

- C++语言不仅有丰富的系统预定义的基本数据类型
  - 基本类型：整型、实型、字符型、指针……
- 而且允许用户在现有数据类型的基础上进行数据类型的自定义
  - 自定义数据类型：数组、**结构体 (struct)**、联合体 (union)、**枚举 (enum)**





## 7.2 结构体类型

- 结构体：将不同性质类型但是互相有关联的数据放在一起，组合成一种新的复合型数据类型，称为结构体类型（结构体）

	储存数据	数据类型	格式灵活
数组	多个	相同	
结构体	多个	不同	✓



## 7.3 结构体类型

- 结构体类型声明的一般形式:

关键字

**struct** **结构体类型名**  $\rightarrow$  由用户自定义 (见名知义)

{

类型名 成员名1;

类型名 成员名2;

类型名 成员名3;

.....

} **;** 结束结构体声明

结构体成员列表

举例

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



## 7.3 结构体类型

- 注意:

- 1. 结构体类型是一种构造（自定义）数据类型，它和整型（int）、字符型（char）等系统定义的基本数据类型具有相同地位，不同的是它是由用户自定义的；
- 2. 结构体类型的声明仅仅是声明了一个类型，系统并不为之分配内存，只有当使用这个类型定义了变量时，系统才会为变量分配内存（等同于int类型的声明，不是变量定义）



## 7.4 结构体变量的定义

- 方法一：先声明结构体类型再定义变量名

```
struct  结构体类型名
{
    类型名 成员名1;
    类型名 成员名2;
    类型名 成员名3;
}; //先声明结构体类型
```

举例



```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
struct student st1, st2;
```

```
struct  结构体类型名 变量1, 变量2; //再定义结构体变量
```



## 7.4 结构体变量的定义

- 方法一：先声明结构体类型再定义变量名

```
struct 结构体类型名
{
    类型名 成员名1;
    类型名 成员名2;
    类型名 成员名3;
}; //先定义结构体类型
```

举例

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```

~~struct~~ 结构体类型名 变量1, 变量2; //再定义结构体变量



在C语言中，要在结构体类型前加关键字struct，C++中则可以省略关键字，较为方便



## 7.4 结构体变量的定义

- 方法二：在声明结构体类型的同时定义变量名

```
struct  结构体类型名  
{
```

类型标识符 成员名1;

类型标识符 成员名2;

类型标识符 成员名3;

```
} 变量1, 变量2; //紧接着定义结构体变量
```

举例



```
struct student  
{  
    int num;  
    char name [20];  
    int age;  
    float score;  
} st1, st2;
```

可以继续用方法1定义st3、st4



## 7.4 结构体变量的定义

- 方法三：直接定义结构体变量名（不出现结构体名称）

struct //此处不出现结构体类型名

{

类型标识符 成员名1;

类型标识符 成员名2;

类型标识符 成员名3;

} 变量1, 变量2;

举例



```
struct
{
    int num;
    char name [20];
    int age;
    float score;
} st1, st2;
```

无法继续用方法1定义st3、st4



## 7.4 结构体变量的定义

关于三种方法的使用:

- 第三种方法虽然合法, 但很少使用: 如果只需要 st1、st2 两个变量, 后面不需要再使用结构体名定义其他变量, 可以使用方法三
- 如果程序比较简单, 结构体类型只在本程序中使用的情况下, 可以使用第二种方法
- 第一种方法将声明结构体类型和定义结构体变量分开, 便于不同的函数甚至不同的程序文件都可以使用声明的结构体类型。 (推荐使用第一种方法)





## 7.4 结构体变量的定义

- 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作



## 7.4 结构体变量的定义

- 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存**总和**

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```

st1和st2在内存中各占32个字节 (32=4+20+4+4)

```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
};
int main()
{
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof (st2) << endl;
    return 0;
}
```





```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
};
int main()
{
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof (st2) << endl;
    return 0;
}
```

双编译器验证



Microsoft Vi

```
32
32
```

TDM-GCC 9.2.0 32-bit Debug

D:\hoi

```
32
32
```



```
#include <iostream>
using namespace std;
struct student
{
    short num;
    char name[20];
    char age;
    float score;
};
int main()
{
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof (st2) << endl;
    return 0;
}
```

结果为?





```
#include <iostream>
using namespace std;
struct student
{
    short num;
    char name[20];
    char age;
    float score;
};
int main()
{
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof (st2) << endl;
    return 0;
}
```

双编译器验证

Debug x86

Microsoft Visual Stu

28  
28

TDM-GCC 9.2.0 32-bit Debug

D:\homework\

28  
28

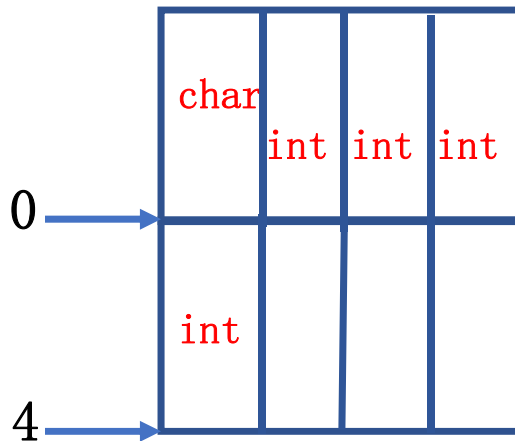


## \*结构体中的内存对齐

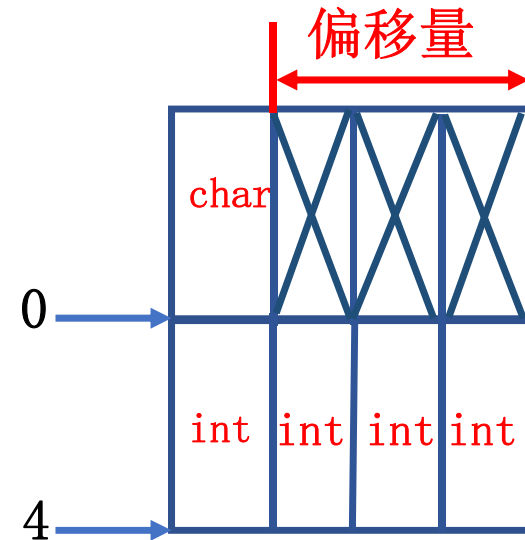
1. 实际上CPU将内存分割成多个模块，每次从内存中读取一个模块，这个模块的大小可能为2、4、8、16 ( $2^n$ ) 字节等

2. 内存对齐的意义：

```
struct student{  
    char num;  
    int age;};  
};
```



1. **无内存对齐**：从0号下标的地址开始访问，int型数据占4字节，需要访问两次（3+1）



2. **有内存对齐**：每次访问一定长度，只要访问一次，以空间换时间，提高访问效率

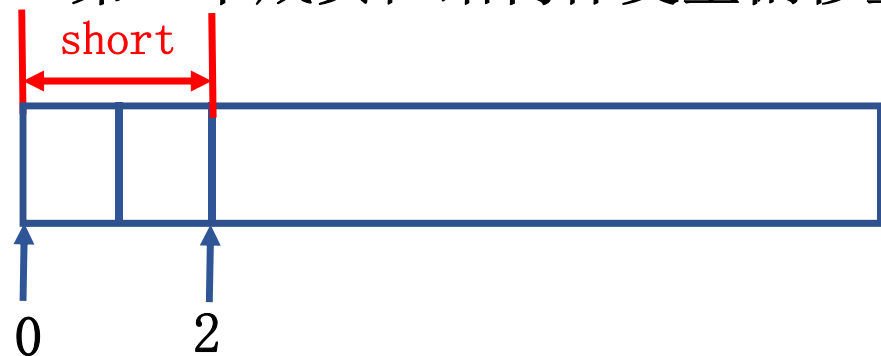


## 结构体的对齐规则:

1. 第一个成员在结构体变量偏移量为 0 的地址处
2. 其他成员变量要对齐到某个数字(对齐数)的整数倍的地址处  
对齐数 = 编译器默认的一个对齐数与该成员大小的较小值, VS中默认的对齐数为8
3. 结构体总大小为最大对齐数 (每个成员变量都有一个对齐数) 的整数倍

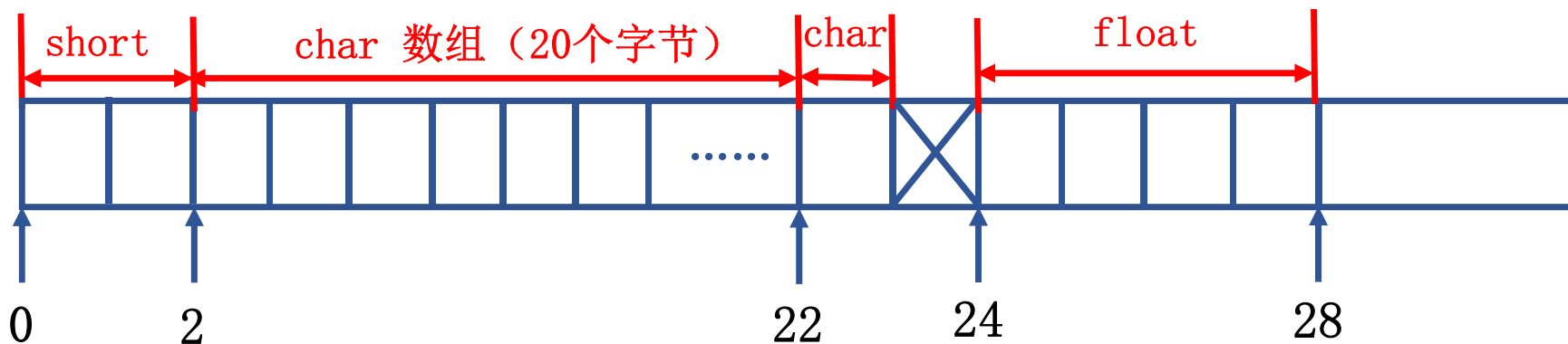


1. 第一个成员在结构体变量偏移量为 0 的地址处



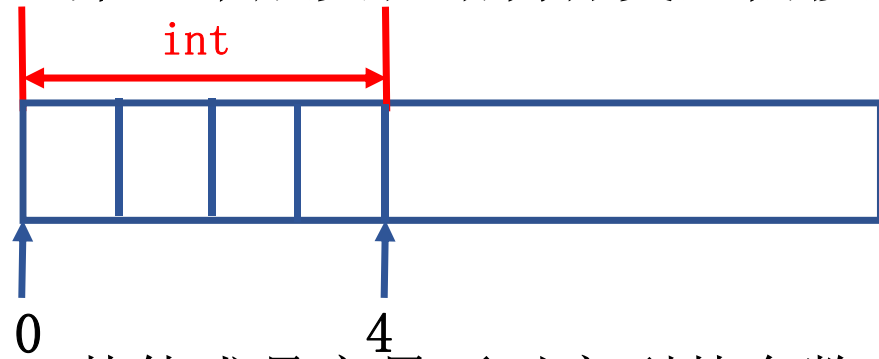
```
struct student{  
    short num;  
    char name[20];  
    char age;  
    float score;  
};
```

2. 其他成员变量要对齐到某个数字(对齐数)的整数倍的地址处  
对齐数 = 默认的一个对齐数与该成员大小的较小值 (默认的对齐数为8)



3. 结构体总大小为最大对齐数 (每个成员变量都有一个对齐数) 的整数倍  
28为4(最大对齐数)的整数倍

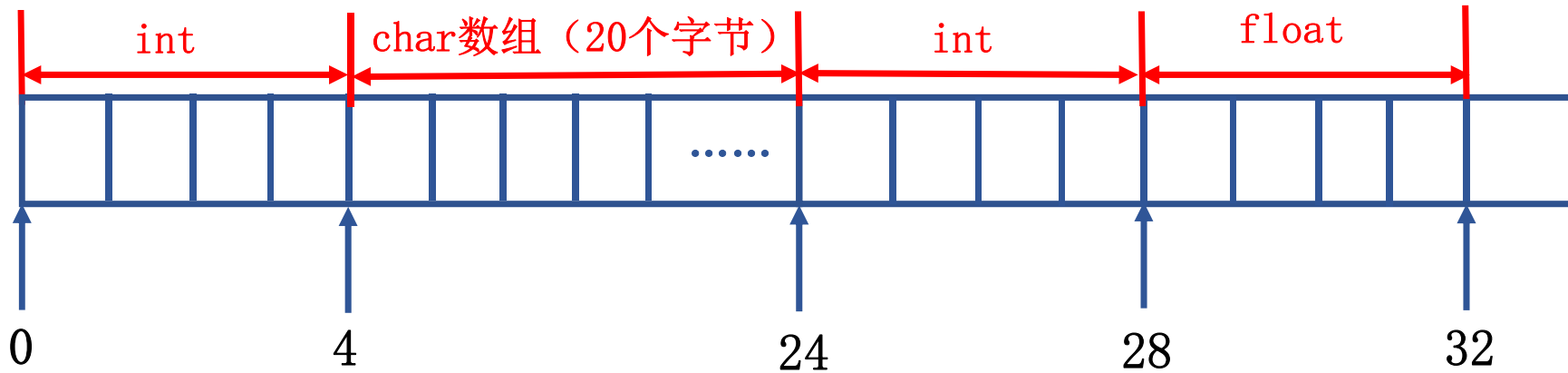
1. 第一个成员在结构体变量偏移量为 0 的地址处



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

2. 其他成员变量要对齐到某个数字(对齐数)的整数倍的地址处

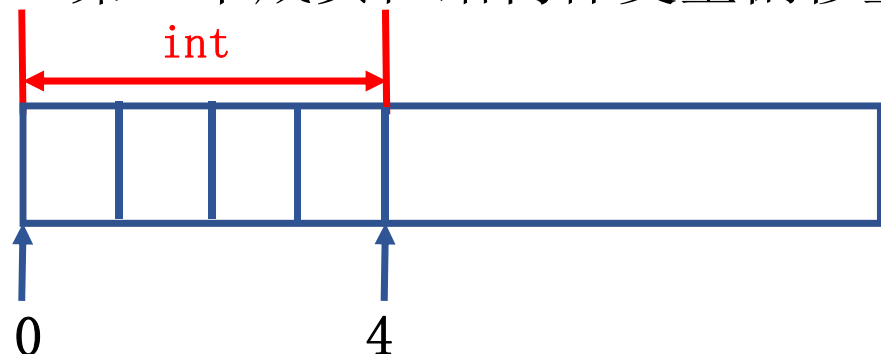
对齐数 = 编译器默认的一个对齐数与该成员大小的较小值, VS中默认的对齐数为8



3. 结构体总大小为最大对齐数 (每个成员变量都有一个对齐数) 的整数倍

32为4(最大对齐数)的整数倍

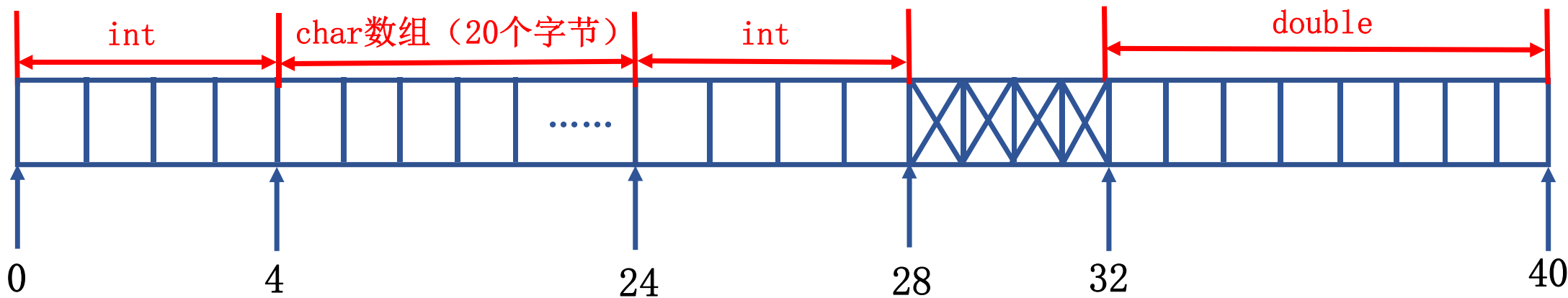
1. 第一个成员在结构体变量偏移量为 0 的地址处



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

2. 其他成员变量要对齐到某个数字(对齐数)的整数倍的地址处

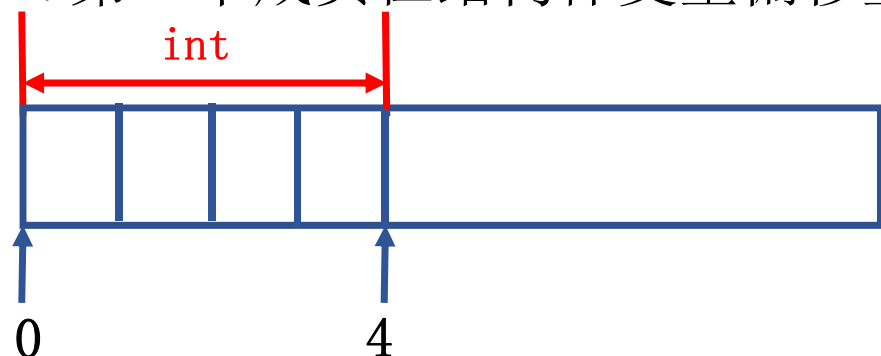
对齐数 = 编译器默认的一个对齐数与该成员大小的较小值, VS中默认的对齐数为8



3. 结构体总大小为最大对齐数 (每个成员变量都有一个对齐数) 的整数倍

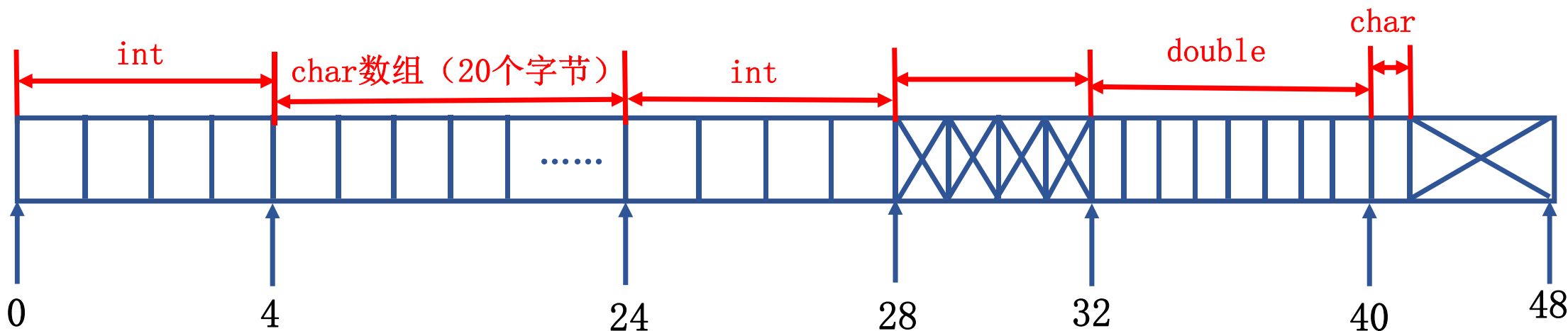
40是8(最大对齐数)的整数倍, 结构体总大小为40

1. 第一个成员在结构体变量偏移量为 0 的地址处



```
struct student{  
    int num;  
    char name[20];  
    int age;  
    double score;  
    char sex;
```

2. 其他成员变量要对齐到某个数字(对齐数)的整数倍的地址处;  
对齐数 = 编译器默认的一个对齐数与该成员大小的较小值, VS中默认的对齐数为8



3. 结构体总大小为最大对齐数 (每个成员变量都有一个对齐数) 的整数倍

41不是8(最大对齐数)的整数倍, 结构体总大小为48



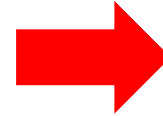
## 7.4 结构体变量的定义

- 注意:

- 结构体类型和结构体变量是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存总和
- 结构体声明的位置很重要



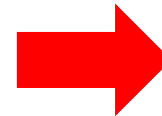
```
#include <iostream>
using namespace std;
int main()
{
    struct student
    {
        int num;
        char name[20];
        int age;
        float score;
    };
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof (st2) << endl;
    return 0;
}
```



1. 内部声明：结构体声明放在函数内部，只能被该声明所属的函数使用



```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
};
int main()
{
    struct student st1, st2;
    cout << sizeof (st1) << endl;
    cout << sizeof(st2) << endl;
    return 0;
}
```



2. 外部声明：结构体声明放在函数外部，可以被其后面所有的函数使用



## 7.4 结构体变量的定义

- 注意:

- 结构体类型和结构体变量是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存总和
- 结构体声明的位置很重要
  1. 常规建议放在函数外部
  2. 如果多源文件，把结构体声明放在头文件中，然后在需要用这个结构体的源文件中包含对应的头文件





## 7.4 结构体变量的定义

- 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存**总和**
- 结构体**声明的位置**很重要
- 结构体成员本身也可以是另一个结构体（**结构体嵌套**）



## 7.4 结构体变量的定义

➤结构体成员本身也可以是另一个结构体（结构体嵌套）

例如：先声明结构体date，再使用结构体date

```
struct date
{
    int year;
    int month;
    int day;
};
struct student
{
    int num;
    char name[20];
    date birthday; //date是结构体类型，birthday是date类型的成员
};
```



## 7.4 结构体变量的定义

➤ 结构体成员本身也可以是另一个结构体（结构体嵌套）

例如：不事先声明，直接嵌套

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



## 7.4 结构体变量的定义

• 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占**内存总和**
- 结构体**声明的位置**很重要
- 结构体成员本身也可以是另一个结构体（**结构体嵌套**）
- **不允许**对结构体本身**递归定义**

例如:

```
struct student
{
    int age;
    struct student stu;
} st1, st2;
```

×



## 7.4 结构体变量的定义

### • 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存**总和**
- 结构体**声明的位置**很重要
- 结构体成员本身也可以是另一个结构体（**结构体嵌套**）
- **不允许**对结构体本身**递归定义**

例如:

```
struct student
{
    int age;
    struct student stu;
}st1, st2;
```



结构体变量是有大小的，不能定义“一个结构体变量的大小等于一个整型变量的大小加上它本身的大小”（无法递归求自身大小！）



## 7.4 结构体变量的定义

### • 注意:

- 结构体**类型**和结构体**变量**是不同的概念，不要混淆，只能对变量进行操作
- 结构体变量所占内存空间是其全体成员所占内存**总和**
- 结构体**声明的位置**很重要
- 结构体成员本身也可以是另一个结构体（**结构体嵌套**）
- **不允许**对结构体本身**递归定义**

### 例如:

```
struct student
{
    int age;
    struct student stu;
}st1, st2;
```



结构体变量是有大小的，不能定义“一个结构体变量的大小等于一个整型变量的大小加上它本身的大小”（无法递归求自身大小！）

- 可以定义成员为**结构体自身的指针** `struct student *next;` 因为指针所占内存大小是确定的（后续课程的动态申请、链表都会用到）



## 7.5 结构体变量的初始化

- 方法一：先声明结构体类型，再定义结构体变量并赋初值

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
}; //先声明结构体类型
student st1={2001156, "AA", 18, 82}; //再定义变量并初始化
```



## 7.5 结构体变量的初始化

- 方法二：在声明结构体类型的同时定义结构体变量并赋初值

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
} st1={2001156, "AA", 18, 82}; //定义结构体变量的同时初始化
```





## 7.5 结构体变量的初始化

```
struct student  
{
```

```
    int num;
```

```
    char name[20];
```

```
    struct date  
    {
```

```
        int year;
```

```
        int month;
```

```
        int day;
```

```
    } birthday;
```

```
} st1;
```

```
st1={2001156, "AA", {2000, 1, 19}}; //嵌套型结构体变量的初始化
```

此处大括号可以不加，  
但是加上可读性更好



```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
}st1 = { 2001156, "AA", 18, 82 };
int main()
{
    cout << st1.num<<' ' <<st1.name<<' ' <<st1.age<<' ' <<st1.score<<endl;
    return 0;
}
```

1. 一一对应赋值
2. 不允许跳跃赋值

```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
} st1 = { 2001156, "AA" };
int main()
{
    cout << st1.num<<' ' << st1.name<<' ' << st1.age<<' ' << st1.score<<endl;
    return 0;
}
```

1. 一一对应赋值
2. 不允许跳跃赋值
3. 可以给成员部分赋值, 其他未赋值成员被设置为0



## 7.5 结构体变量的初始化

➤注意：如果在定义结构体变量的时候没有初始化，那么后面就不能全部一起整体赋值（初始化赋值和变量定义不能分开）

例如：

```
struct student
{
```

```
    int num;
```

```
    char name [20];
```

```
    int age;
```

```
    float score;
```

```
} st1={2001156, "AA", 18, 82};
```

✓

```
struct student
{
```

```
    int num;
```

```
    char name [20];
```

```
    int age;
```

```
    float score;
```

```
} st1;
```

```
st1={2001156, "AA", 18, 82};
```

✗



## 7.5 结构体变量的初始化

➤注意：如果在定义结构体变量的时候没有初始化，那么后面就不能全部一起整体赋值了

例如：

```
struct student
{
```

```
    int num;
    char name [20];
    int age;
    float score;
```

```
} st1={2001156, "AA", 18, 82};
```

✓

```
struct student
{
```

```
    int num;
    char name [20];
    int age;
    float score;
```

```
} st1;
```

对成员进行单个赋值

st1.num=2001156

strcpy(st1.name, "AA") ✓

st1.age=18

st1.score=82



## 7.6 结构体变量的使用

- 定义了结构体变量后，就可以访问其中的每一个成员。结构体中的成员可以像基本变量那样赋值、输入输出、参与表达式计算等操作，这些操作统称为对结构体成员的访问
- 结构体变量成员表现形式：

结构体变量名. 成员名

例：st1.num    st1.age...

直接成员运算符，直接调用结构体中的某个成员，它在所有运算符中的优先级为2

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```



## 7.6 结构体变量的使用

➤注意:

1. 结构体嵌套中, 成员本身是结构体时, 结构体变量必须逐层引用:

```
struct student
{
    int num;
    char name[20];
    struct date
    {
        int year;
        int month;
        int day;
    } birthday;
} st1;
```

逐层找到最低一级成员名进行访问

st1.birthday.year

st1.birthday.month

st1.birthday.day

只能对最内层的成员进行运算, 输入输出操作

## 7.6 结构体变量的使用



➤注意:

2. 不能整体输入输出一个结构体变量，只能对结构体变量中的各个成员分别进行输入输出

```
struct student  
{
```

```
    int num;
```

```
    char name [20];
```

```
    int age;
```

```
    float score;
```

```
};
```

```
student st1 = {2001156, "AA", 18, 82}, st2;
```

`cin>>st1; cout<<st1; ✗`

错误列表

整个解决方案

✗ 错误 2

⚠ 警告 0

i 27消息的 0

✗

生成 + IntelliSense

代码

说明

E0349

没有与这些操作数匹配的 "<<" 运算符

C2679

二进制"<<": 没有找到接受"student"类型的右操作数的运算符(或没有可接受的转换)

`cin>>st1.num; cout<<st1.num; ✓`



## 7.6 结构体变量的使用



➤注意:

3. 相同类型的结构体变量可以相互赋值, 这个变量所有成员的值将全部赋值给另外一个变量的成员 (整体赋值)

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```

`st1=st2` 等价于

`st1.num=st2 . num`  
`strcpy(st1.name, st2.name)`  
`st1.age=st2 . age`  
`st1.score=st2 . score`



## 7.6 结构体变量的使用

➤注意:

4. 结构体变量的成员可以像普通变量一样进行各种操作

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```



<code>int i, *p;</code>	<code>s1.num; int *p;</code>	
<code>i++;</code>	<code>s1.num++;</code>	自增/减
<code>... + i*10 +...;</code>	<code>... + s1.num*10 +...;</code>	各种表达式
<code>if (i&gt;=10)</code>	<code>if (s1.num&gt;=10)</code>	
<code>p = &amp;i;</code>	<code>p = &amp;s1.num;</code>	取地址
<code>scanf("%d", &amp;i);</code>	<code>scanf("%d", &amp;s1.num);</code>	输入
<code>cout &lt;&lt; i;</code>	<code>cout &lt;&lt; s1.num;</code>	输出
<code>fun(i);</code>	<code>fun(s1.num);</code>	函数实参
<code>return i;</code>	<code>return s1.num;</code>	返回值

```
#include <iostream>
using namespace std;
struct student //声明结构体类型student
{
    int num;
    char name[20];
    int age;
    float score;
};
int main()
{
    student st1 = { 2001156, "AA", 18, 82 }, st2; //定义结构体变量st1, st2, 并对st1初始化
    st1.age++; //对结构体变量成员进行自增运算
    st2 = st1; //对结构体变量进行相互赋值 (可以进行整体赋值)
    cout << st2.num << endl << st2.name << endl << st2.age << endl << st2.score << endl;
    //对结构体变量成员分别进行输出 (不能进行整体输出)
    return 0;
}
```

```
2001156
AA
19
82
```



例：书上P89页末例题(此页例题因篇幅较大表现为为左右排版，下同)

```
#include <iostream>
using namespace std;
struct inflatable
{
    char name[20];
    float volume;
    double price;
}; //声明结构体类型

int main()
{
    inflatable guest = { "Glorious Gloria", 1.88, 29.99 };
    //定义结构体变量guest并初始化
    inflatable pal = { "Audacious Arthur", 3.12, 32.99 };
    //定义结构体变量pal并初始化
    cout << "Expand your guest list with " << guest.name;
    //注意访问结构体变量中成员的方式
    cout << " and " << pal.name << "!" << endl;
    cout << "You can have both for $" ;
    cout << guest.price + pal.price << "!" << endl;
    guest = pal; //结构体变量的相互赋值
    cout << guest.name << ' ' << guest.volume << ' ' << guest.price << endl;
    guest.price++;
    pal.price++;
    cout << guest.price + pal.price << endl; //对结构体变量成员的操作
    return 0;
}
```



```
#include <iostream>
using namespace std;
struct inflatable
{
    char name[20];
    float volume;
    double price;
}; //声明结构体类型
```

Microsoft Visual Studio 调试控制台

```
Expand your guest list with Glorious Gloria and Audacious Arthur!
You can have both for $62.98!
Audacious Arthur 3.12 32.99
67.98
```

//定义结构体变量pal并初始化

```
cout << "Expand your guest list with " << guest.name;
```

//注意访问结构体变量中成员的方式

```
cout << " and " << pal.name << "!" << endl;
```

```
cout << "You can have both for $" ;
```

```
cout << guest.price + pal.price << "!" << endl;
```

```
guest = pal;
```

```
cout << guest.name << ' ' << guest.volume << ' ' << guest.price << endl;
```

```
guest.price++;
```

```
pal.price++;
```

```
cout << guest.price + pal.price << endl;
```

```
return 0;
```

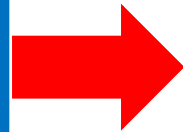
```
}
```



## 7.7 结构体数组变量

- 结构体数组:

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st1, st2;
```



如果有800个学生信息录入，定义800个结构体变量？  
st1, st2.....st800



## 7.7 结构体数组变量

结构体数组：结构体数组中每个元素都是一个结构体类型，它们分别包括各个成员项

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st[100];
```



## 7.7 结构体数组变量

- 结构体数组的定义（参照结构体变量定义方法）

方法1:

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st[n];
```

方法2:

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
} st[n];
```

方法3:

```
struct
{
    int num;
    char name [20];
    int age;
    float score;
} st[n];
```





## 7.7 结构体数组变量

- 结构体数组的初始化

方法1（初始化数组规则）

```
struct student
```

```
{
```

```
    int num;
```

```
    char name [20];
```

```
    int age;
```

```
    float score;
```

```
};
```

```
student st [3] = {{2001156, "AA", 18, 82},  
                  {2001157, "BB", 18, 76},  
                  {2001158, "CC", 19, 80}};
```

初始化写三行，可读性更好

st [0]

st [1]

st [2]

//用逗号分隔每个元素的值，并将这些值用花括号括起来



## 7.7 结构体数组变量

- 结构体数组的初始化

方法2：（初始化结构规则）

```
struct student
```

```
{
```

```
    int num;
```

```
    char name [20];
```

```
    int age;
```

```
    float score;
```

```
};
```

```
student st [3]={2001156, "AA", 18, 82, 2001157, "BB", 18,  
76, 2001158, "CC", 19, 80}; /*用逗号分隔每个成员的值，并  
将这些值用花括号括起来*/
```

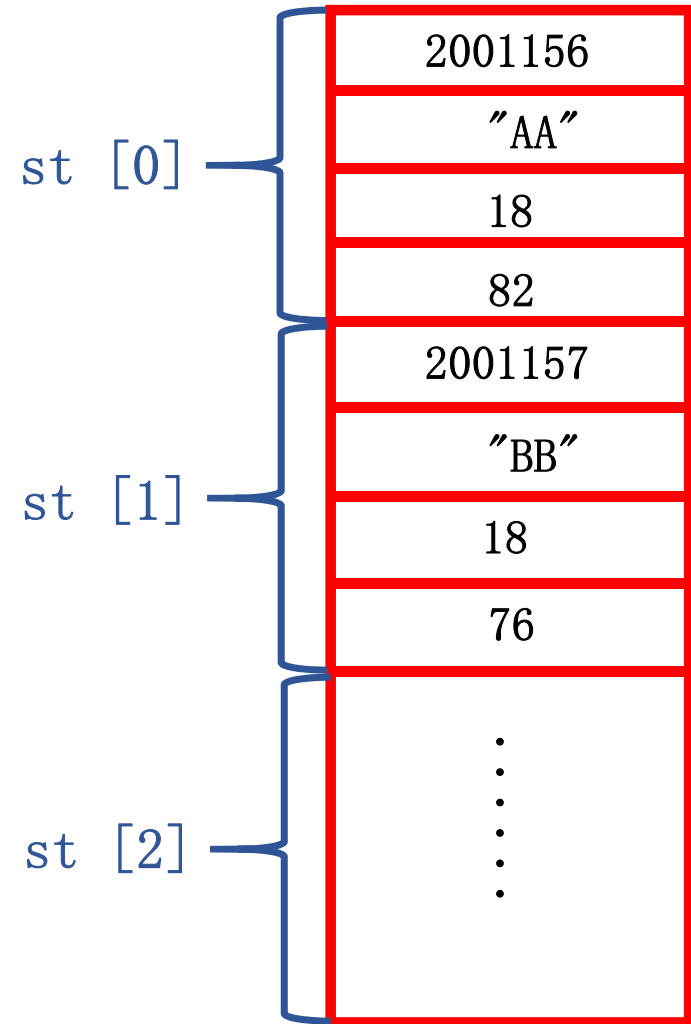
不推荐此用法



## 7.7 结构体数组变量

注意：结构体数组在内存中是连续存放的

```
struct student
{
    int num;
    char name [20];
    int age;
    float score;
};
student st [3] = {{2001156, "AA", 18, 82},
                  {2001157, "BB", 18, 76},
                  {2001158, "CC", 19, 80}};
```





## 7.7 结构体数组变量

- 访问结构体数组成员

```
struct student
{
    int num;
    char name [20];
    int age;
    float score[3];
};
struct student st [3]={
    {2001156, "AA", 18, 82},
    {2001157, "BB", 18, 76},
    {2001158, "CC", 19, 80}};
```

st [0].score -结构体数组第一个元素的score成员

st [1].age -结构体数组第二个元素的age成员

st [2].num-结构体数组第三个元素的num成员

st [0].score[0]-结构体数组第一个元素的第一个成员score (逐个访问)

浮点型

结构体类型

```
#include <iostream>
using namespace std;
struct student //声明结构体类型student
{
    int num;
    char name[20];
    int age;
    float score;
};
int main()
{
    struct student st[3] ;
    for (int i=0; i < 3; i++)
        cin >> st[i].num >> st[i].name >> st[i].age >> st[i].score; //输入所有学生信息
    for (int i=0; i < 3; i++)
        cout << "学号:"<<st[i].num << ' ' << "姓名:"<<st[i].name << ' ' << "年龄:" <<st[i].age << ' '
        << "分数:"<< st[i].score << endl; //输出所有学生信息
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
200156 AA 18 82
200157 BB 18 76
200158 CC 19 80
学号:200156 姓名:AA 年龄:18 分数:82
学号:200157 姓名:BB 年龄:18 分数:76
学号:200158 姓名:CC 年龄:19 分数:80
```



# 结构体数组的访问举例

- 问题描述:

一个小组中有3个学生，每个学生分别有3门成绩，编写程序依次输入学生的学号、姓名、年龄和三门课程的成绩信息，计算学生的平均成绩并输出每个学生的学号、姓名、年龄和三门课程的平均成绩



```
#include <iostream>
using namespace std;
int main()
{
    struct student
    {
        int num;
        char name[20];
        int age;
        float score[3];
        float ave;
    };
    student st[3];
```

```
    int i, j, k;
    for (i = 0; i < 3; i++)
    {
        cout << "请输入学生的学号、姓名和年龄信息"<<endl;
        cin >> st[i].num >> st[i].name >> st[i].age;
        float sum = 0; //sum要清零
        for (j = 0; j < 3; j++)
        {
            cout << "请输入学生的成绩信息" << endl;
            cin >> st[i].score[j];
            sum += st[i].score[j];
        }
        st[i].ave = sum / 3;
    }
    for (k = 0; k < 3; k++)
        cout <<"学号: "<<st[k].num <<' ' << "姓名: "<<
st[k].name <<' ' <<"年龄: "<< st[k].age<<' ' <<"平均分:
"<<st[k].ave << endl;
    return 0;
}
```

```

#include <iostream>
using namespace std;
int main()
{
    struct student
    {
        int num;
        char name[20];
        int age;
        float score[3];
        float ave;
    };
    student st[3];
    int i, j, k;

```

```

        for (i = 0; i < 3; i++)
        {
            cout << "请输入学生的学号、姓名和年龄信息\n";
            for (j = 0; j < 3; j++)
            {
                cout << "请输入学生的成绩信息\n";
                cin >> st[i].score[j];
                sum + st[i].score[j];
            }
            st[i].ave = sum / 3;
            for (k = 0; k < 3; k++)
            {
                cout << "学号: " << st[k].name << " 姓名: " << st[k].name << " 年龄: " << st[k].age << " 平均分: " << st[k].ave << endl;
            }
            return 0;
        }
}

```

Microsoft Visual Studio 调试控制台

请输入学生的学号、姓名和年龄信息

1001 AA 18

请输入学生的成绩信息

90

请输入学生的成绩信息

85

请输入学生的成绩信息

80

请输入学生的学号、姓名和年龄信息

1002 BB 18

请输入学生的成绩信息

90

请输入学生的成绩信息

90

请输入学生的成绩信息

90

请输入学生的学号、姓名和年龄信息

1003 CC 18

请输入学生的成绩信息

95

请输入学生的成绩信息

95

请输入学生的成绩信息

95

学号: 1001 姓名: AA 年龄: 18 平均分: 85

学号: 1002 姓名: BB 年龄: 18 平均分: 90

学号: 1003 姓名: CC 年龄: 18 平均分: 95





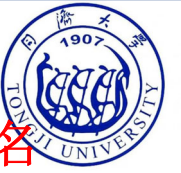
# 结构体数组作函数参数

问题描述:

改写上述程序，将程序中的部分功能定义为函数，实现相同的目的

```
#include <iostream>
using namespace std;
#define N 3
#define M 3
struct student{
int num;
char name[20];
int age;
float score[M];
float ave;
}stu[N] = {{1001, "AA", 18, 90, 85, 80},
           {1002, "BB", 18, 90, 90, 90},
           {1003, "CC", 18, 95, 95, 95}};
void aver(struct student arr[], int n);
//形参为结构体数组
```

```
int main()
{
    aver (stu, N); //实参为结构体数组名
    for (int k = 0; k < N; k++)
        cout << "学号: " << stu[k].num << "
姓名: " << stu[k].name << "年龄: " <<
stu[k].age << "平均分: " << stu[k].ave <<
endl;
    return 0;
}
void aver(student arr[], int n) {
    float sum;
    for (int i = 0; i < N; i++)
    {
        sum = 0;
        for (int j = 0; j < M; j++)
            sum += stu[i].score[j];
        stu[i].ave = sum / 3.0;
    }
}
```



```

#include <iostream>
using namespace std;
#define N 3
#define M 3
struct student{
int num;
char name[20];
int age;
float score[3];
float ave;
}stu[N] = {{1001, "AA", 18, 90, 85, 80},
{1002, "BB", 18, 90, 90, 90},
{1003, "CC", 18, 95, 95, 95}};
void aver(struct student arr[], int n);
//形参为结构体数组

```

Microsoft Visual Studio 调试控制台

```

学号: 1001姓名: AA年龄: 18平均分: 85
学号: 1002姓名: BB年龄: 18平均分: 90
学号: 1003姓名: CC年龄: 18平均分: 95

```

```

stu[i].age = arr[i].age;
stu[i].ave = arr[i].ave;
endl;
return 0;
}
void aver(student arr[], int n) {
float sum;
for (int i = 0; i < N; i++)
{
sum = 0;
for (int j = 0; j < M; j++)
sum += stu[i].score[j];
stu[i].ave = sum / 3.0;
}
}

```



# 结构体数组作函数参数

1. 数组名代表数组首元素地址
2. 用数组名做函数参数，实际上传递的是数组首元素的地址
3. 系统编译时将数组名按指针处理
4. 形参为数组时，从实参数组那里获得起始地址，因此形参数组和实参数组共同占用一段内存单元，调用函数时形参的改变将影响实参的值
5. 形参`struct student arr[]`中方括号的数值并无实际作用（不分配存储单元），只是用`arr[]`这样的形式表明一个一维数组，接受实参的地址，编译时对括号内的数字不做理会，因此形参中的元素个数可以写也可以不写



## 7.8 结构体指针变量

- 定义一个指针变量，用来指向一个结构体变量，该指针的值就是这个结构体变量的起始地址
- 引入结构体指针的目的是为了实现函数之间的数据传递
- 指向结构体变量的指针变量的定义形式：  
    **结构体类型名 \*指针名**
- 注意和指向结构体成员中某个变量的指针进行区分



例如:

```
struct student
{
    int num;
    char name[20];
    int age;
    float score;
} st={2001156, "AA", 18, 82};
```

```
struct student st, *p;
```

```
int *p1;
```

```
float *p2;
```

```
p = &st;
```

//指针指向结构体变量st(++  $\Leftrightarrow$  +32)

```
p1 = &st.age;
```

//指针指向结构体变量的age成员(++  $\Leftrightarrow$  +4)

```
p2 = &st.score;
```

//指针指向结构体变量的score成员(++  $\Leftrightarrow$  +4)



```
struct student
{
    int num;
    char name[20];
    int age;
    float score;
} st={2001156, "AA", 18, 82};
```

```
struct student st, *p;
```

```
p=&st; ✓
```

```
p=&st.num; ✗ (左右地址的基类型不匹配)
```

: error C2440: “=” : 无法从 “int \*” 转换为 “main::student \*”

message : 与指向的类型无关; 强制转换要求 reinterpret\_cast、C 样式强制转换或函数样式强制转换

```
p=(struct student *)&st.num; ✓ 进行强制类型转换后正确
```



## 7.8 结构体指针变量

- 指向结构体变量的指针变量的定义形式:

结构体类型名 \*指针名

- 利用指向结构体变量的指针引用其成员的方式

1. (\*结构体指针). 成员名

例: (\*p). num: 括号不能省略, 括号作用是优先执行取内容操作

(\*p). num  $\xleftrightarrow[\text{结合性}]{\text{优先级}}$  st. num ✓

(\*p). num  $\xleftrightarrow[\text{不加括号?}]{\text{结合性}}$  \*p. num  $\xleftrightarrow{\text{优先级}}$  \* (p. num) ✗





## 7.8 结构体指针变量

- 指向结构体变量的指针变量的定义形式:

结构体类型名 \*指针名

- 利用指向结构体变量的指针引用其成员的方式

1. (\*结构体指针). 成员名

(\*p). num: 括号不能省略, 括号作用是优先执行取内容操作

2. 结构体指针->成员名 (->为指向运算符)

p->num: 指针变量 p 指向结构体变量st中的 num 成员, p->num 最终代表的就是 num 这个成员的内容

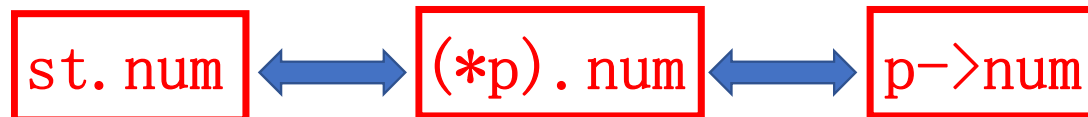


```
struct student
{
    int num;
    char name[20];
    int age;
    float score;
} st={2001156, "AA", 18, 82};
student *p=&st; //定义p为指向student类型数据的指针变量并指向stu
```

利用指向结构体变量的指针引用其成员的方式:

1. `(*p).num`

2. `p->num`



```
#include <iostream>
using namespace std;
struct student
{
    int num;
    char name[20];
    int age;
    float score;
} st = { 2001156, "AA", 18, 82 };
int main()
{
```

```
    student* p = &st; //定义p为指向student类型的指针变量并指向st
```

```
    cout << st.num << ' ' << st.name << ' ' << st.age << ' ' << st.score << endl;
```

```
    cout << (*p).num << ' ' << (*p).name << ' ' << (*p).age << ' ' << (*p).score << endl;
```

```
    cout << p->num << ' ' << p->name << ' ' << p->age << ' ' << p->score << endl;
```

```
//引用结构体变量的成员的三种方式
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 调试控制台

```
2001156 AA 18 82
2001156 AA 18 82
2001156 AA 18 82
```



## 7.8 结构体指针变量

- 利用指针变量引用结构体数组元素成员的方式:

```
struct student st [4];
```

```
struct student *p;
```

```
p=st //p=&st[0]
```

```
p->num; // (*p).num;
```

```
(p+i)->num; // (* (p+i) ).num
```

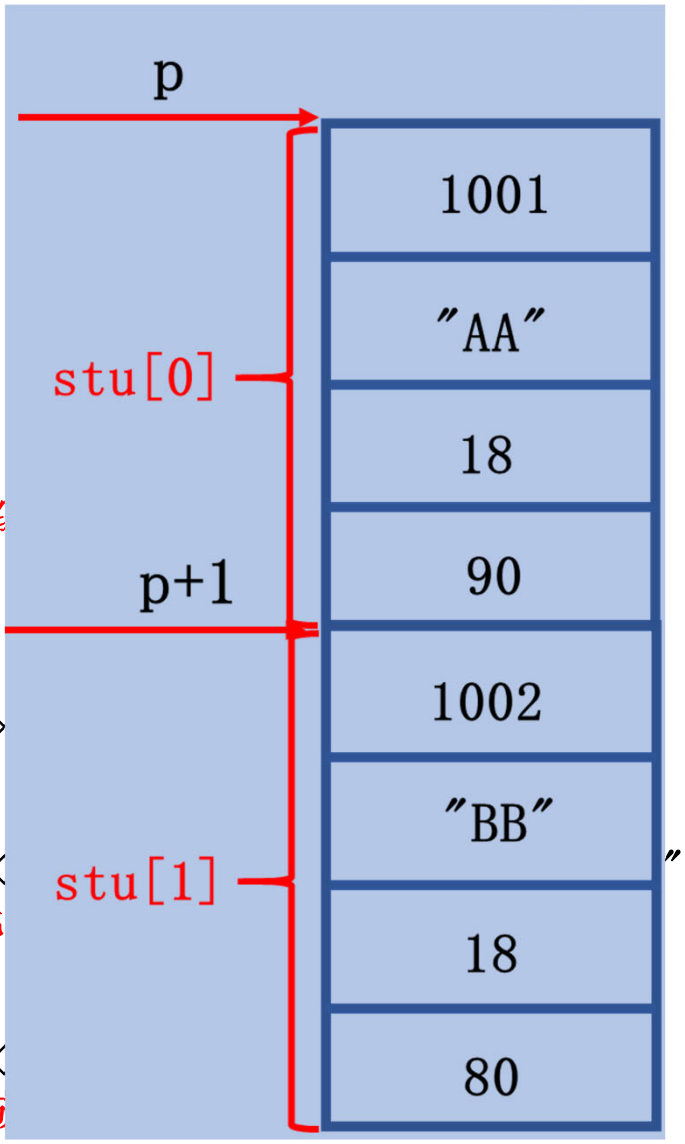


```
st[i].num;
```



```
#include <iostream>
using namespace std;
struct student {
    int num;
    char name[20];
    int age;
    float score;
} stu[2] = { {1001, "AA", 18, 90}, {1002, "BB", 18, 80} }; //定义结构体数组并初始化
int main() {
    struct student* p = stu;
    cout << "学号: " << p->num << "姓名: " << p->name << "年龄: " << p->age << "成绩: " << p->score << endl;
    //利用指针指向结构体数组第一个元素的成员
    cout << "学号: " << (p + 1)->num << "姓名: " << (p + 1)->name << "年龄: " << (p + 1)->age << "
成绩: " << (p + 1)->score << endl; //利用指针指向结构体数组第二个元素的成员
    for(int i=0; i<2; i++)
        cout << "学号: " << (p + i)->num << "姓名: " << (p + i)->name << "年龄: " << (p + i)->age << "
成绩: " << (p + i)->score << endl; //利用指针遍历结构体数组的所有成员
    return 0;
}
```

```
#include <iostream>
using namespace std;
struct student {
    int num;
    char name[20];
    int age;
    float score;
} stu[2] = { {1001, "AA", 18, 90}, {1002, "BB", 18, 80} }; //定义结构体数组
int main() {
    struct student* p = stu;
    cout << "学号: " << p->num << "姓名: " << p->name << "年龄: " << p->
    p->score << endl; //利用指针指向结构体数组第一个元素的成员
    cout << "学号: " << (p + 1)->num << "姓名: " << (p + 1)->name <<
    成绩: " << (p + 1)->score << endl; //利用指针指向结构体数组第二个元
    for(int i=0;i<2;i++)
        cout << "学号: " << (p + i)->num << "姓名: " << (p + i)->name <<
    "成绩: " << (p + i)->score << endl; //利用指针遍历结构体数组的所有成员
    return 0;
}
```



```
#include <iostream>
using namespace std;
struct student {
    int num;
    char name[20];
    int age;
    float score;
```

```
} stu[2] = { {1001, "AA", 18, 90}, {1002, "BB", 18, 80} }; //定义结构体数组并初始化
```

```
int main() {
    struct student* p = stu;
    cout << "学号: " << p->num << "姓名: " << p->name << "年龄: " << p->age << "成绩: " <<
p->score << endl; //利用指针指向结构体数组第一个元素的成员
    cout << "学号: " << (p + 1)->num << "姓名: " << (p + 1)->name << "年龄: " << (p + 1)->age << "
成绩: " << (p + 1)->score << endl; //利用指针指向结构体数组第二个元素的成员
    for(int i=0; i<2; i++)
        cout << "学号: " << (p + i)->num << "姓名: " << (p + i)->name << "年龄: " << (p + i)->age <<
"成绩: " << (p + i)->score << endl; //利用指针遍历结构体数组的所有成员
    return 0;
```

 Microsoft Visual Studio 调试控制台

```
学号: 1001姓名: AA年龄: 18成绩: 90
学号: 1002姓名: BB年龄: 18成绩: 80
学号: 1001姓名: AA年龄: 18成绩: 90
学号: 1002姓名: BB年龄: 18成绩: 80
```



# 结构体类型数据做函数参数

1. 结构体变量名作函数参数
2. 用指向结构体变量的指针作函数参数，将结构体变量的地址传给形参
3. 用结构体数组作函数参数，将结构体数组的首地址传给形参





```
#include<iostream>
using namespace std;
struct stud
{long int num;
float score;};
```

```
void funvr(struct stud t)
{t.num = 2000101; //注意结构体变量成员的引用形式
t.score = 71.0;}
```

```
void funar(struct stud t[])
{t[0].num = 3000101;
t[0].score = 81.0; //注意结构体数组元素的成员的引用形式
t[1].num = 3000102;
t[1].score = 82.0;}
```

```
void funpr(struct stud* t)
{t->num = 4000101; //注意通过结构体指针变量引用成员的具体形式
(*t).score = 92.0;}
```



```
void main()
{
    struct stud a[2] = {{1000101, 61.0}, {1000102, 62.0}};
    struct stud b = a[0];

    cout<<"old b:"<< b.num<<' ' <<b.score<<endl; //输出结构体变量b的成员的原有值
    funvr(b); //将结构体变量作为函数的实参
    cout<<"call funvr() new b: "<< b.num << ' ' << b.score << endl;

    funpr(&b); //将结构体变量的地址作为函数的实参
    cout<<"call funpr() new b: "<< b.num << ' ' <<b.score << endl;

    cout<<"old a[0]:"<< a[0].num<<' ' <<a[0].score << endl; /*输出结构体数组a元素的原来的成员值*/
    cout<<"old a[1]:"<< a[1].num << ' ' << a[1].score << endl;
    funar(a); //将结构体数组a作为函数的实参，然后再输出其元素的成员的值
    cout<<" new a[0]:"<< a[0].num << ' ' << a[0].score << endl;
    cout<<"new a[1]:"<< a[1].num << ' ' << a[1].score << endl;
}
```

```

void main()
{
    struct stud a[2] = {{1000101, 61.0},
    struct stud b = a[0];
    cout<<"old b:"<< b.num<<' ' <<b.score<<endl;
    funvr(b); //将结构体变量作为函数的实参
    cout<<"call funvr() new b: " << b.num<<' ' <<b.score<<endl;
    funpr(&b); //将结构体变量的指针对作函数的实参
    cout<<"call funpr() new b: " << b.num<<' ' <<b.score<<endl;
    /*值被修改了*/
    cout<<"old a[0]:"<< a[0].num<<' ' <<a[0].score << endl; /*输出结构体数组a元素的原来的成员值*/
    cout<<"old a[1]:"<< a[1].num << ' ' << a[1].score << endl;
    funar(a); //将结构体数组名a作为函数的实参，然后再输出其元素的成员的值，已经被修改了
    cout<<" new a[0]:"<< a[0].num << ' ' << a[0].score << endl;
    cout<<"new a[1]:"<< a[1].num << ' ' << a[1].score << endl;
}

```

Microsoft Visual Studio 调试控制台

```

old b:1000101 61
call funvr() new b: 1000101 61
call funpr() new b: 4000101 92
old a[0]:1000101 61
old a[1]:1000102 62
new a[0]:3000101 81
new a[1]:3000102 82

```

# 结构体参数调用归纳



1) 结构体变量作为函数参数时，由于实参中全部内容是通过值传递的方式一一传给形参，形参结构体变量成员值的改变不影响对应的实参结构体变量成员值的改变(单向传值)，程序简单易懂，但是如果结构体变量占的内存空间很大，在实参形参传递过程中空间的开销较大，效率不高。

实参：结构体变量b

```
b. num=1000101  
b. score=61.0
```

单向传值

形参：结构体变量t

```
t. num= 1000101  
t. score=61.0
```

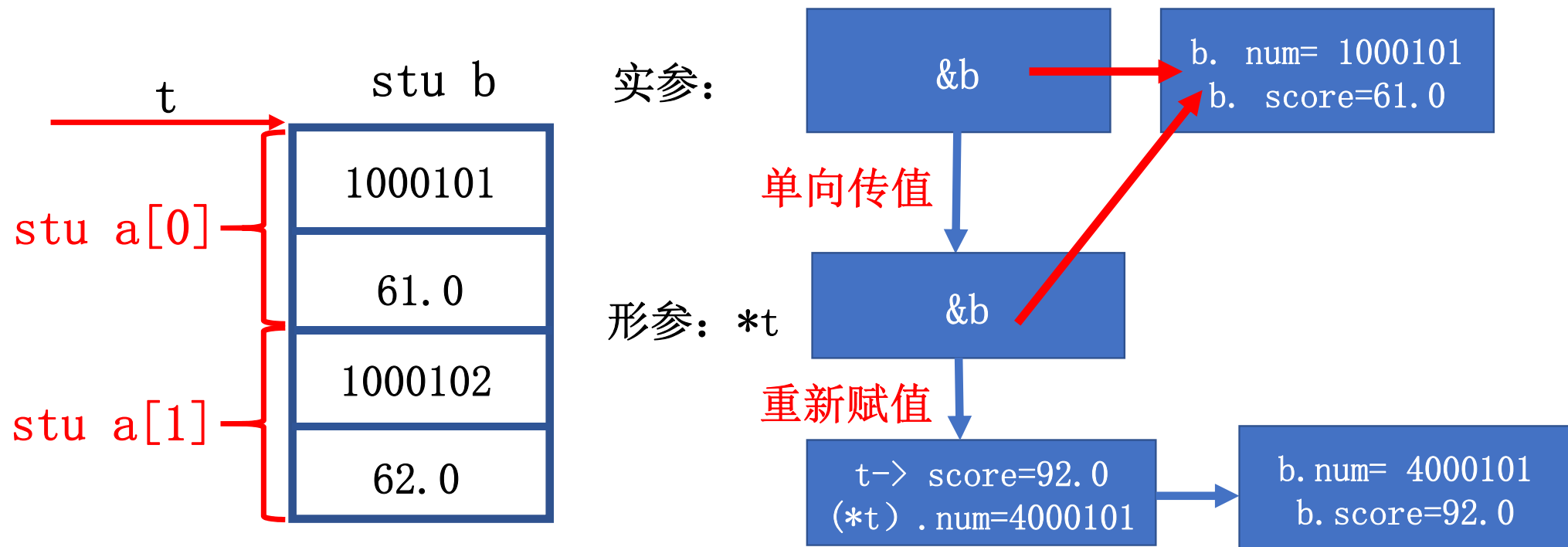
重新赋值

```
t. num= 2000101  
t. score=71.0
```



# 结构体参数调用归纳

2) 结构体指针变量或结构体数组作为函数参数时，形参结构体数组元素成员值的改变会影响对应的实参结构体数组成员值的改变。在调用函数时实参将结构体数组的首地址/变量的地址传给形参，而不是一一传值，因此空间开销较小，效率较高。





## 7.9 枚举

- 定义：如果一个变量的取值有限且离散，那么将这个变量所有可能的取值一一列举出来，并限定在某个集合内（将变量的值一一列举）

- 声明形式：

enum 枚举类型名 {枚举（元素）常量取值表};

↑  
关键字

↑  
以标识符形式表示的整型量，  
表示枚举类型的取值集合



## 7.9 枚举

• 例如:

1. 定义一周内的天数

```
enum weekday {sun, mon, tue, wed, thu, fri, sat};
```

关键字

枚举类型名

枚举常量（元素）列表（可能取值）

2. 定义光谱的分类

```
enum spectrum {red, orange, yellow, green, blue, indigo, violet,  
ultraviolet};
```

## 7.9 枚举



### ➤注意:

1. 声明枚举类型时，系统给每一个枚举元素一个指定的整数值（序号），默认从0开始，依次加1

```
enum weekday {sun, mon, tue, wed, thu, fri, sat};  
              0,    1,    2,    3,    4,    5,    6
```

2. 可以在声明时另行指定枚举元素的序号值（部分指定）

```
enum weekday {sun=7, mon=1, tue, wed, thu, fri, sat};  
              7,      1,    2,    3,    4,    5,    6
```

```
enum weekday {sun, mon=4, tue, wed, thu, fri, sat};  
              0,      4,    5,    6,    7,    8,    9
```





## 7.9 枚举

➤注意:

3. 如果定义的常量重复，编译器不报错，但后续使用中会有问题

```
enum weekday {sun=3, mon=1, tue, wed, thu, fri, sat};  
              3,      1,      2,      3,      4,      5,      6
```

4. 声明枚举类型时系统已初始化，结束声明后不允许再对枚举元素重新赋值

```
int sun; 错误 error C2365: “main::sun”: 重定义; 以前的定义是“枚举数”
```

```
sun=10; 错误 error C2440: “=”: 无法从“int”转换为“main::weekday”
```



## 7.9 枚举

• 枚举变量定义形式:

1. `enum 枚举类型名 {枚举 (元素) 常量取值表} 变量; //声明枚举类型的同时定义变量`

2. `enum 枚举类型名 {枚举 (元素) 常量取值表}; //声明枚举类型`

`enum 枚举类型名 变量; (该变量的值只能在枚举常量中取值) //用已声明的枚举类型定义变量`

• 例如:

1. 定义一周内的天数

```
enum weekday {sun, mon, tue, wed, thu, fri, sat};
```

↑      ↑  
关键字 枚举类型名

↑  
枚举常量列表 (可能取值)

```
enum weekday w1, w2;
```

定义枚举类型变量w1和w2, 且枚举变量的值只能是sun到sat中间之一

## 7.9 枚举



5. 枚举变量只能参与赋值、比较和输入输出操作，参与运算时用其本身的整数值

0      1      2      3      4      5      6

```
enum weekday {sun, mon, tue, wed, thu, fri, sat}w1,w2;
```

(1) 赋值:

```
w1 = mon; //将枚举常量值1赋给枚举变量
```

```
w2 = fri; //将枚举常量值5赋给枚举变量
```

```
w2 = w1; //将枚举变量w1的值1赋值给同类型枚举变量w2
```

(2) 比较:

```
w1 == mon //将枚举变量w1的值与枚举常量mon比较，返回true/false
```

```
w2 > wed //将枚举变量w2的值与枚举常量sat比较，返回true/false
```

```
w2 >= w1 //比较枚举变量w1和w2的值，返回true/false（按各自的值比较）
```



## 7.9 枚举

5. 枚举变量只能参与赋值、比较和输入输出操作，参与运算时用其本身的整数值

0      1      2      3      4      5      6

```
enum weekday {sun, mon, tue, wed, thu, fri, sat}w1,w2;
```

(3) 输出:

直接输出: (按整型输出)

```
w1 = wed;
```

```
cout << w1 << endl;      //输出的是变量w1的整数值，即wed的序号值3
```

```
printf("w1=%d\n",w1);      //w1=3
```

间接输出:

```
switch(w1) //括号内表达式为枚举类型
```

```
{
```

```
    case wed:      //wed是整型常量3，不加双引号
```

```
        cout << "wed"; //判断变量w1的值，输出对应字符串
```

```
        break;
```

```
    ... //其它case
```

```
}
```



## 7.9 枚举

5. 枚举变量只能参与赋值、比较和输入输出操作，参与运算时用其本身的整数值

0      1      2      3      4      5      6

```
enum weekday {sun, mon, tue, wed, thu, fri, sat}w1,w2;
```

(4) 输入:

直接输入:

```
weekday w1;
```

```
scanf("%d", &w1); //键盘读入对应的数值（3），赋值为相应的枚举常量（wed）（输入9也可以，后续会错）
```

```
cin >> w1; //cin如果想读入枚举，需要对>>进行运算符重载（荣誉课）
```

间接输入:

```
char s[80];
```

```
cin >> s; //键盘输入sun
```

```
if (!strcmp(s, "sun"))
```

```
    w1=sun;
```

```
    //w1被赋值为sun常量
```

```
else if (...)
```

```
int w;
```

```
cin >> w;
```

```
w1=(weekday)w; //无法直接赋值，需要强制类型转换
```

```
//如果w的值不在0-6之间，后续会错
```




## 7.10 用typedef声明类型

- 定义：为一种数据类型声明一个新名字。这里的数据类型包括基本数据类型（int, char等）和自定义的数据类型（struct等），使得代码可读性更高，意义更明确

- 形式：

typedef 原有类型名 新类型名（大写）



通常将typedef声明的类型名用大写字母表示，以便于系统提供的标准类型标识符区别



## 7.10 用typedef声明类型

- 形式:

typedef 原有类型名 新类型名 (大写)

例如:

typedef int COUNTER; //用指定标识符COUNTER代表(非替代) int类型

COUNTER i, j;  int i, j; //相当于定义i, j为COUNTER型, 即整型



例如:

```
struct Point
{
    float x;
    float y;
}; // 声明结构体类型

typedef struct Point POINT;

POINT pt; //新类型名定义结构体变量
pt.x = 10.0f;
pt.y = 20.0f;
```

```
//在struct前加了typedef, 表明是声明新类型名
typedef struct Point
{
    float x;
    float y;
} POINT; //Point是新类型名, 不是变量名

POINT pt; // 用新类型名定义结构体变量
pt.x = 10.0f;
pt.y = 20.0f;
```

- 上述两种方式都是为struct Point结构体类型声明了一个新名字POINT
- 所以POINT pt 相当于 struct Point pt //pt为结构体类型变量
- POINT \*p 相当于 struct Point \*p //p为指向此结构体类型的指针





## 7.10 用typedef声明类型

• 例如:

```
int a[10], b[10], c[10], d[10]; //四个一维数组, 大小相同
```

```
typedef int ARR[10]; //将大小为10的一维数组声明为新类型ARR
```

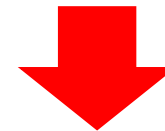
```
ARR a, b, c, d; //用ARR定义数组变量
```

```
1. ARR a; ⇔ int a[10];
```

```
2. ARR b[5] ⇔ int b[5][10];
```



相当于a, b, c, d都被定义为一维整型数组, 含有10个元素



相当于数组b中每一个元素被定义为一个含有10个元素的一维整型数组, 数组b即二维数组



## 7.10 用typedef声明类型

- 声明类型的方法步骤:

1. 先按定义变量的方法写出定义语句 (`int i;`)
2. 将变量名换成新类型名 (`int INTEGER;`)
3. 在最前面加typedef(`typedef int INTEGER;`)
4. 用新类型名去定义变量(`INTEGER i, j;`)



有了上面的typedef声明, `INTEGER i, j;` 相当于定义了  
两个INTEGER类型变量*i*, *j*, 即整型



## 7.10 用typedef声明类型

1. 先按定义变量的方法写出定义语句 (`int a[10];`)
2. 将变量名换成新类型名 ( `int ARR[10];`)
3. 在最前面加typedef(`typedef int ARR[10];`)
4. 用新类型名去定义变量(`ARR a, b[5];`)



有了上面的typedef声明:

1. 相当于a被定义为一维整型数组, 含有10个元素
2. 相当于b被定义为一个二维整型数组

## 7.10 用typedef声明新类型



- 注意:

1. typedef 没有创造新类型，只是将原有数据类型声明新名字（“大名和昵称”）
2. typedef是声明类型，不是定义变量
3. typedef和#define的差异

#define则是宏定义（用一个标识符来表示一个常量），发生在预处理阶段，也就是编译之前，它只进行简单机械的字符串替换，不进行任何检查



## 7.10 用typedef声明新类型

- 注意:

1. typedef 没有创造新类型，只是将原有数据类型声明新名字
2. typedef是声明类型，不是定义变量
3. typedef和#define的差异

例如:

```
typedef (char*) PIONT; //声明PIONT为字符型指针类型
```

PIONT a, b的效果同char\* a; char\* b;表示定义了两个字符型指针变量

```
#define PIONT char*//预处理指令，仅将char*替换为标识符PIONT，替换完毕再编译
```

PIONT a, b的效果同char\* a, b;表示定义了一个字符型指针变量a和字符型变量b



```
#include <iostream>
using namespace std;
typedef (char*) pint; //注意：两者有行末分号的区别
#define PINT char *
pint s1, s2;
PINT s3, s4;
int main()
{
    cout << sizeof(s1) << sizeof(s2) << sizeof(s3) << sizeof(s4);
    return 0;
}
```



Microsoft Visual Studio

4441

```
#include <iostream>
using namespace std;
typedef (char*) pint; //注意：两者有行末分号的区别
#define PINT char *
pint s1, s2;
PINT s3, s4;
int main()
{
    cout << sizeof(s1) << sizeof(s2) << sizeof(s3) << sizeof(s4);
    return 0;
}
```

字符型指针变量

字符型变量



# 总结

- 自定义数据类型（数组、结构体、枚举）
- 结构体类型声明
- 结构体变量的定义及初始化（三种形式）
- 利用结构体变量访问成员
- 结构体数组变量（访问结构体数组成员、结构体数组做函数参数）
- 结构体指针变量（结构体和函数、数组、指针内容的融合）
- 枚举
- 用typedef声明类型（用新类型名定义变量的意义）