

§ 7. 用户自定义数据类型

7.1. 自定义数据类型的引入

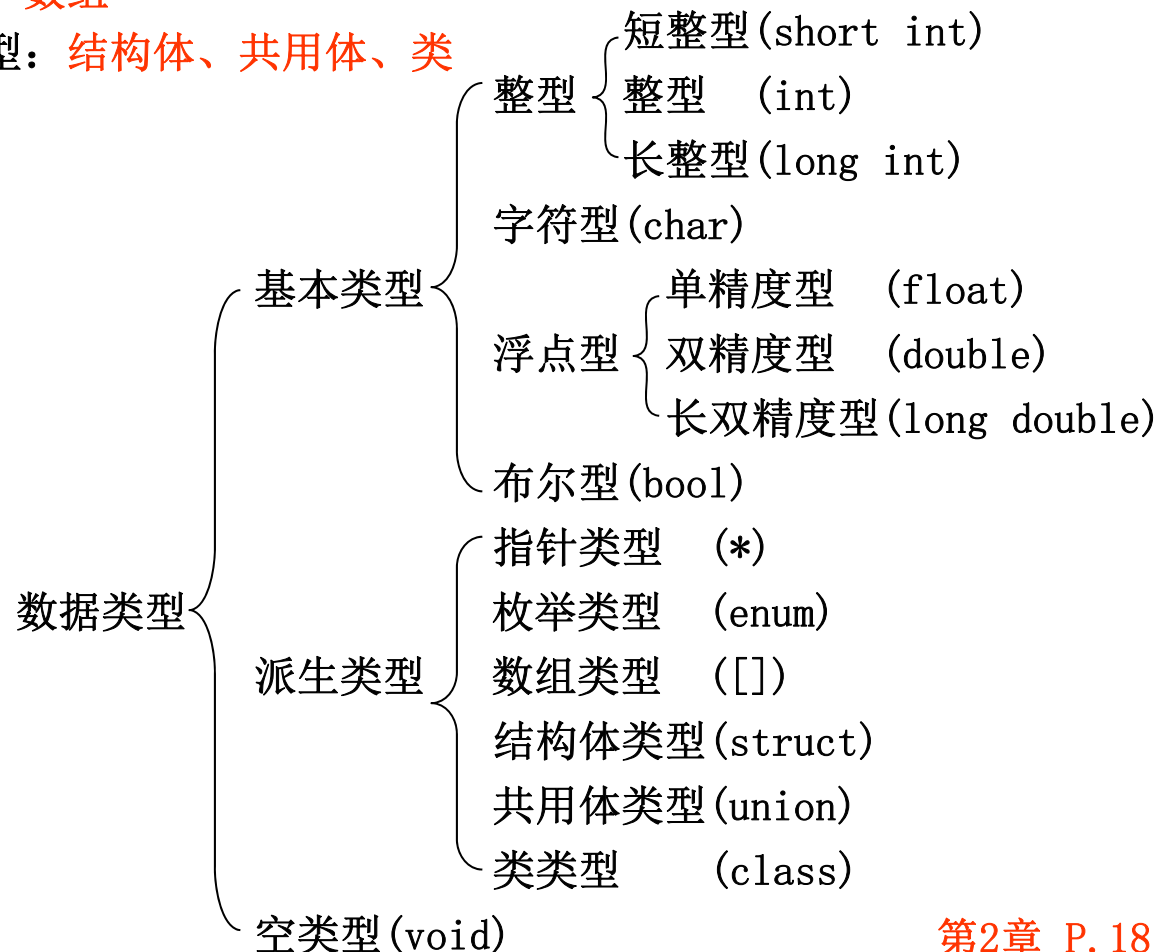
7.1.1. 自定义数据类型的含义

用**基本数据类型**以及**已存在的自定义数据类型**组合而成的新数据类型

7.1.2. 自定义数据类型的分类

元素同类型的自定义数据类型：**数组**

元素不同类型的自定义数据类型：**结构体、共用体、类**



§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.1. 引入

例1: 键盘输入学生的学号、姓名、性别、年龄、成绩和家庭住址, 再依次输出

```
int main()
{
    int num, age;
    char sex, name[20], addr[30];
    float score;
    cin >> num ... ;
    ...
    cout << sex ... ;
    return 0;
}
```

1个学生的6方面信息:
用6个彼此完全独立的
不同类型的变量来表达

缺点: 访问时无整体性

例2: 键盘输入100个学生的学号、姓名、性别、年龄、成绩和家庭住址, 再依次输出

```
const int N=100;
int main()
{
    int num[N], age[N], i;
    char sex[N], name[N][20], addr[N][30];
    float score[N];
    for(i=0; i<N; i++) {
        cin >> num[i] ... ;
        ...
        cout << sex[i] ... ;
    }
}
```

100个学生的6方面信息:
用6个彼此完全独立的不同类型的
数组变量来表达

缺点: 1. 访问时无整体性
2. 访问同一个人时, 不同数组
的下标必须对应

例3: 键盘输入学生的学号、姓名、性别、年龄、成绩和家庭住址, 再依次输出, 要求以指针方式操作

```
int main()
{
    int num, age, *p_num=&num, *p_age=&age;
    char sex, name[20], addr[30];
    char *p_sex=&sex, *p_name=name, *p_addr=addr;
    float score, *p_score=&score;
    cin >> *p_num ... ;
    ...
    cout << *p_sex ... ;
    return 0;
}
```

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.1. 引入

将描述一个事物的各方面特征的数据组合成一个有机的整体，说明数据之间的内在关系

7.2.2. 结构体类型的定义

struct 结构体名 {	struct student {
结构体成员1 (类型名 成员名)	int num;
...	char name[20];
结构体成员n (类型名 成员名)	char sex;
}; (带分号)	int age;
	float score;
	char addr[30];
	};

★ 结构体名, 成员名命名规则同变量

★ 同一结构体的成员名不能同名, 但可与其它名称 (其它结构体的成员名, 其它变量名等) 相同

struct x1 {	struct x2 {	struct x3 {
int num;	int num;	float num;
...
};	};	};
main()	fun()	void num()
{	{	{
long num;	int num[10];	...
}	}	}

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.2. 结构体类型的定义

- ★ 结构体名, 成员名命名规则同变量
- ★ 同一结构体的成员名不能同名, 但可与其它名称 (其它结构体的成员名, 其它变量名等) 相同
- ★ 每个成员的类型可以相同, 也可以不同
- ★ 每个成员的类型既可以是基本数据类型, 也可以是已存在的自定义数据类型

```
struct date {  
    int year;  
    int month;  
    int day;  
};  
struct student {  
    int num;  
    char name[20];  
    char sex;  
    struct date birthday;  
    float score;  
    char addr[30];  
};
```

struct date必须在struct student的前面定义, 否则无法知道birthday占多少字节

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

★ 每个成员的类型不允许是自身的结构体类型

无法判断 monitor 占多少个字节

- ★ 每个成员的类型不允许是自身的结构体类型
- ★ 结构体类型的定义既可以放在函数外部, 也可以放在函数内部 (具体定义及访问规则见 7.2.8)
- ★ 结构体类型的大小为所有成员的大小的总和, 可用sizeof(struct 结构体名) 计算, 但不占用具体的内存空间 (结构体变量占用一段连续的内存空间)

int i; sizeof(int)得4
但int型不占空间, i占4字节

§ 7. 用户自定义数据类型

7.2. 结构体类型

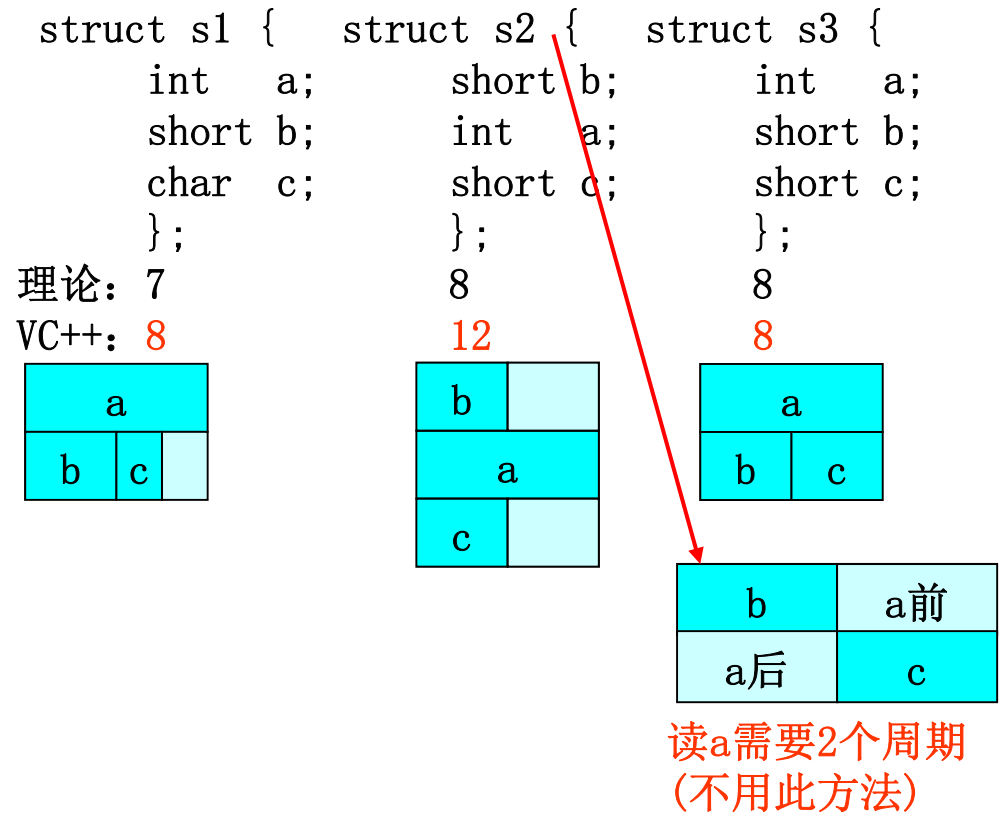
7.2.2. 结构体类型的定义

★ 在不同的编译系统中, 有时为了加快程序运行速度, 采用按**数据总线宽度**对齐的方法来计算结构体类型的大小, 可能出现填充字节

=>此概念需了解, 本书不继续讨论, 仍按无填充计算

P.196 注释① sizeof(student)结果为 64 => 68

```
#include <iostream>
using namespace std;
struct s1 {
    int a;
    short b;
    char c;
};
struct s2 {
    short b;
    int a;
    short c;
};
struct s3 {
    int a;
    short b;
    short c;
};
void main()
{
    cout << sizeof(s1) << endl; 8
    cout << sizeof(s2) << endl; 12
    cout << sizeof(s3) << endl; 8
}
```



§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.2. 结构体类型的定义

- ★ 结构体名, 成员名命名规则同变量
- ★ 同一结构体的成员名不能同名, 但可与其它名称 (其它结构体的成员名, 其它变量名等) 相同
- ★ 每个成员的类型可以相同, 也可以不同
- ★ 每个成员的类型既可以是基本数据类型, 也可以是已存在的自定义数据类型
- ★ 每个成员的类型不允许是自身的结构体类型
- ★ 结构体类型的定义既可以放在函数外部, 也可以放在函数内部 (具体定义及访问规则见 7.2.10)
- ★ 结构体类型的大小为所有成员的大小的总和, 可用sizeof(struct 结构体名) 计算, 但不占用具体的内存空间 (结构体变量占用一段连续的内存空间)
- ★ 在不同的编译系统中, 有时为了加快程序运行速度, 采用按数据总线宽度对齐的方法来计算结构体类型的大小, 可能出现填充字节 (需了解, 本书不讨论)
- ★ C的结构体只能包含数据成员, C++还可以包含函数

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.3. 结构体变量的定义及初始化

7.2.3.1. 先定义结构体类型，再定义变量

```
struct student {  
    ...  
};  
struct student s1;  
struct student s2[10];  
struct student *s3;
```

★ struct在C中不能省，在C++中可省略

★ 结构体变量占用实际的内存空间，根据变量的不同类型(静态/动态/全局/局部)在不同区域进行分配

7.2.3.2. 在定义结构体类型的同时定义变量

```
struct student {  
    ...  
} s1, s2[10], *s3;  
struct student s4;
```

★ 可以再次用7.2.3.1的方法定义新的变量(struct在C++中可省)

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.3. 结构体变量的定义及初始化

7.2.3.3. 直接定义结构体类型的变量 (结构体无名)

```
struct {  
    ...  
} s1, s2[10], *s3;
```

- ★ 因为结构体无名，因此无法再用7.2.3.1的方法进行新的变量定义
(适用于仅需要一次性定义的地方)

7.2.3.4. 结构体变量定义时初始化

```
student s1={1, "张三", 'M', 20, 78.5, "上海"};
```

- ★ 按各成员依次列出

- ★ 若嵌套使用，要列出最低级成员

```
student s1={1, "张三", 'M', {1982, 5, 9}, 78.5};
```

- ★ 可用一个同类型变量初始化另一个变量

```
student s1={1, "张三", 'M', {1982, 5, 9}, 78.5};  
student s2=s1;
```

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

内{}可省
但不建议

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

```
struct student {  
    int    num;  
    char  name[20];  
    char  sex;  
    struct date birthday;  
    float score;  
};
```

内{}可省
但不建议

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.4. 结构体变量的使用

★ P. 198 7.1.3 结构体变量的引用（与“引用”无关）

7.2.4.1. 形式

变量名. 成员名

★ P. 199 说明(5)中：“由于. 运算符的优先级最高” => C中最高，C++中次高

```
struct student {  
    int    num;          s1.num = 1;  
    char   name[20];     strcpy(s1.name, "张三");  
    char   sex;          s1.sex = 'M';  
    int    age;          s1.age = 20;  
    float  score;        s1.score = 76.5;  
    char   addr[30];     strcpy(s1.addr, "上海");  
} s1;
```

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.4. 结构体变量的使用

7.2.4.1. 形式

7.2.4.2. 使用

★ 结构体变量允许进行整体赋值操作

```
student s1={...}, s2;
```

```
s2=s1; //赋值语句
```

用一个同类型变量初始化另一个变量:

```
student s1={...}, s2=s1; //定义时初始化
```

★ 在所有基本类型变量出现的地方，均可以使用该基本类型的结构体变量的成员

<code>int i, *p;</code>	<code>student s1; 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>=10)</code>	<code>if (s1.num>=10)</code>	
<code>p = &i;</code>	<code>p = &s1.num;</code>	取地址
<code>scanf("%d", &i);</code>	<code>scanf("%d", &s1.num);</code>	输入
<code>cout << i;</code>	<code>cout << 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

§ 7. 用户自定义数据类型

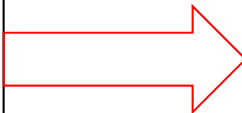
7.2. 结构体类型

7.2.4. 结构体变量的使用

7.2.4.2. 使用

- ★ 结构体变量允许进行整体赋值操作
- ★ 在所有基本类型变量出现的地方，均可以使用该基本类型的结构体变量的成员
- ★ 若嵌套使用，只能对最低级成员操作

```
struct date {  
    int year;  
    int month;  
    int day;  
};  
struct student {  
    int num;  
    char name[9];  
    char sex;  
    struct date birthday ;  
    float score;  
};
```



```
s1.birthday.year=1980;  
cin >> s1.birthday.month;  
cout << s1.birthday.day;
```

- ★ 结构体变量不能进行整体的输入和输出操作

```
student s1={...};  
cin >> s1;    ✖  
cout << s1;   ✖
```

§ 7. 用户自定义数据类型

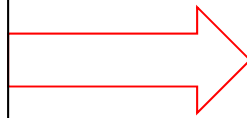
7.2. 结构体类型

7.2.4. 结构体变量的使用

7.2.4.2. 使用

例1：键盘输入学生的学号、姓名、性别、年龄、成绩和家庭住址，再依次输出 (前面例子的对比)

```
int main()
{
    int num;
    int age;
    char sex;
    char name[20];
    char addr[30];
    float score;
    cin >> num ... ;
    ...
    cout << sex ... ;
    return 0;
}
```



```
struct student {
    ...;
};
int main()
{
    struct student s1;

    cin >> s1.num ... ;
    ...
    cout << s1.sex ... ;
    return 0;
}
```

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.5. 结构体变量数组

7.2.5.1. 含义

一个数组，数组中的元素是结构体类型

7.2.5.2. 定义

~~struct~~ 结构体名 数组名[正整常量表达式] 包括整型常量、整型符号常量和整型只读变量

~~struct~~ 结构体名 数组名[正整常量1][正整常量2]

~~struct~~ student s2[10];

~~struct~~ student s4[10][20];

7.2.5.3. 定义时初始化

```
struct student s2[10] = { {1, "张三", 'M', 20, 78.5, "上海"},  
                          {2, "李四", 'F', 19, 82, "北京"},  
                          {..}, {..}, {..}, {..}, {..}, {..}, {..}, {..} };
```

内{}可省
但不建议

★ 其它同基本数据类型数组的初始化

(占用空间、存放、下标范围、初始化时省略大小)

7.2.5.4. 使用

数组名[下标]. 成员名

s2[0].num=1;

cin >> s2[0].age >> s2[0].name;

cout << s2[1].age << s2[1].name;

s2[2].name[0] = 'A'; //注意两个[]的位置

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.5. 结构体变量数组

7.2.5.1. 含义

7.2.5.2. 定义

7.2.5.3. 定义时初始化

7.2.5.4. 使用

例2：键盘输入100个学生的学号、姓名、性别、年龄、成绩和家庭住址，再依次输出(前面例子对比)

```
const int N=100;

int main()
{   int num[N], age[N], i;
    char sex[N];
    char name[N][20];
    char addr[N][30];
    float score[N];
    for(i=0; i<N; i++) {
        cin >> num[i] ... ;
        ...
        cout << sex[i] ...;
    }
}
```



```
const int N=100;
struct student {
    ...;
};
int main()
{   int i;
    struct student s2[N];
    for(i=0; i<N; i++) {
        cin >> s2[i].num ... ;
        ...
        cout << s2[i].sex ...;
    }
    return 0;
}
```

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.5. 结构体变量数组

7.2.5.4. 使用

P. 202 例7.2

```
struct Person {
    char name[20];
    int count;
};

int main()
{
    Person leader[3]={...};
    int i, j;
    char leader_name[20];
    for(i=0; i<10; i++) {
        cin >> leader_name;
        for(j=0; j<3; j++)
            if (!strcmp(leader_name, leader[j].name))
                leader[j].count++;
    }
    cout << endl;
    for(i=0; i<3; i++)
        cout << leader[i].name << ":" <<
            leader[i].count << endl;
    return 0;
}
```

if(!strcmp(leader_name, leader[j].name)){
leader[j].count++;
break;
} 运行效率高, 避免不必要的比较

P. 203 例7.2的变化

```
struct Person {
    string name;
    int count;
};

int main()
{
    Person leader[3]={...};
    int i, j;
    string leader_name;
    for(i=0; i<10; i++) {
        cin >> leader_name;
        for(j=0; j<3; j++)
            if (leader_name == leader[j].name)
                leader[j].count++;
    }
    cout << endl;
    for(i=0; i<3; i++)
        cout << leader[i].name << ":" <<
            leader[i].count << endl;
    return 0;
}
```

if (leader_name == leader[j].name) {
leader[j].count++;
break;
} 运行效率高, 避免不必要的比较

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.6. 指向结构体变量的指针

含义：存放结构体变量的地址

7.2.6.1. 结构体变量的地址与结构体变量中成员地址

student s1;

&s1 : 结构体变量的地址

(基类型是结构体变量, +1表示一个结构体)

&s1.age : 结构体变量中某个成员地址

(基类型是该成员的类型, +1表示一个成员)

7.2.6.2. 结构体指针变量的定义

struct 结构体名 *指针变量名

struct student s1, *s3;

int *p;

s3=&s1; 结构体变量的指针

s3的值为2000, ++s3后值为2063

p=&s1.age; 结构体变量成员的指针

p的值为2025, ++p后值为2029

```
struct student s1;  
&s1  
&s1.age
```

s1	2000 2003	num
	2004 ... 2023	name
	2024	sex
	2025 2028	age
	2029 2032	score
	2033 ... 2062	addr

```
#include <iostream>  
#include <iomanip>  
using namespace std;  
struct student {  
    int    num;  
    char  name[20];  
    char  sex;  
    int    age;  
    float score;  
    char  addr[30];  
};  
void main()  
{ struct student s1;  
  cout << &s1 << endl;  
  cout << &s1+1 << endl;  
  
  cout << &s1.num << endl;  
  cout << &s1.num+1 << endl;  
  
  cout << hex <<int(&s1.sex) << endl;  
  cout << hex <<int(&s1.sex+1)<<endl;  
  
  cout << &s1.age << endl;  
  cout << &s1.age+1 << endl;  
}
```

地址X
地址X + 68

地址X
地址X + 4

地址Y (X + 24)
地址Y + 1

地址Z (Y+1 + 3)
地址Z + 4

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.6. 指向结构体变量的指针

7.2.6.1. 结构体变量的地址与结构体变量中成员地址

7.2.6.2. 结构体指针变量的定义

7.2.6.3. 使用

(*指针变量名). 成员名

指针变量名->成员名 ⇔ (*指针变量名). 成员名

```
struct student s1, *s3=&s1;  
cout << s1.num    << s1.name    << s1.sex;  
cout << (*s3).num << (*s3).name << (*s3).sex;  
cout << s3->num    << s3->name    << s3->sex;
```

s3->age++; 值后缀++

++s3->age; 值前缀++

§ 7. 用户自定义数据类型

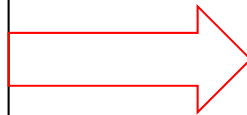
7.2. 结构体类型

7.2.6. 指向结构体变量的指针

7.2.6.3. 使用

例3：键盘输入学生的学号、姓名、性别、年龄、成绩和家庭住址，再依次输出，要求以指针方式操作 (前例)

```
int main()
{  int num, age;
   char sex, name[20], addr[30];
   float score;
   int *p_num=&num;
   int *p_age=&age;
   char *p_sex=&sex;
   char *p_name=name;
   char *p_addr=addr;
   float *p_score=&score;
   cin >> *p_num ... ;
   ...
   cout << *p_sex ...;
   return 0;
}
```



```
struct student {
    ...;
};

int main()
{  struct student s1;
   struct student *s3;
   s3 = &s1;
   cin >> s3->num ... ;
   ...
   cout << s3->sex ...;
   return 0;
}
```

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.6. 指向结构体变量的指针

7.2.6.4. 指向结构体数组的指针

```
struct student s2[10], *p;
```

```
p=s2; ✓
```

```
p=&s2[0]; ✓
```

```
p=&s2[0].num; ✗ 类型不匹配
```

```
p=(struct student *)&s2[0].num; ✓ 强制类型转换
```

与第6章中

```
int a[3][4], *p;
```

```
p=a; //编译报错
```

原因相同

各种表示形式:

`(*p).num` : 取p所指元素中成员num的**值**

`p->num` : ...

`p[0].num` : ...

`p+1` : 取p指元素的下一个元素的**地址**

`(*p+1).num`: 取p指向的元素的下一个元素的num**值**

`(p+1)->num` : ...

`p[1].num` : ...

`(p++)->num` : 先取p所指元素的成员num的值, p再指向下一个元素

`(++p)->num` : p先指向下一个元素, 再取p所指元素的成员num的值

`p->num++` : 取p所指元素中成员num的值, 值++

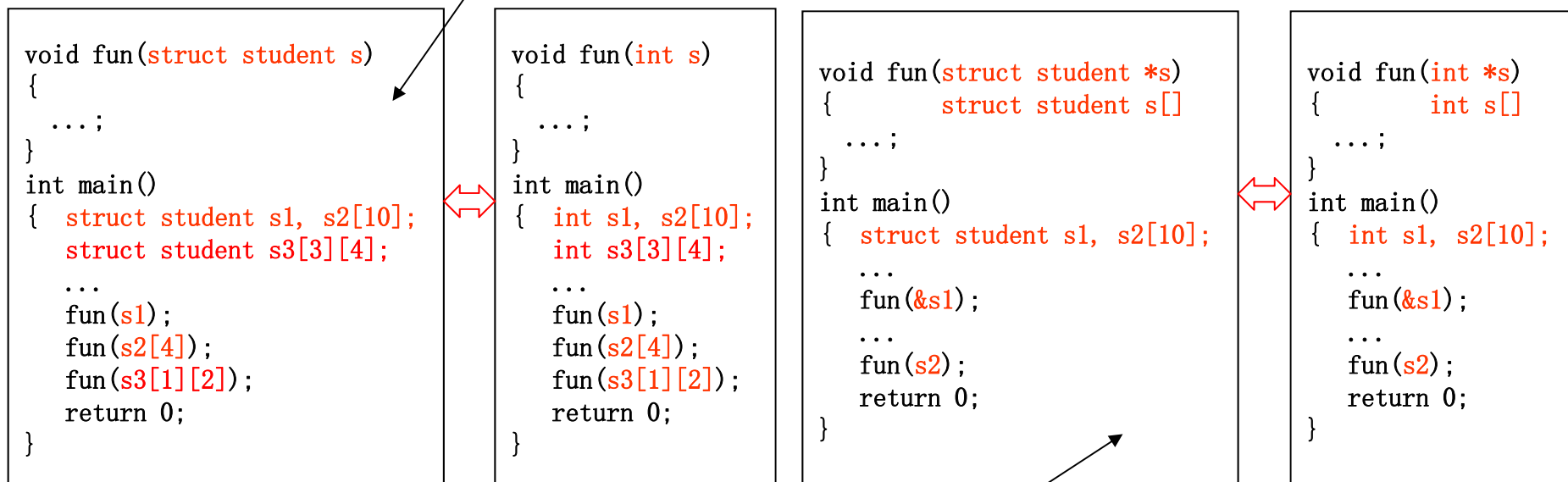
§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.7. 结构体数据类型作为函数参数

7.2.7.1. 形参为结构体简单变量

★ 对应实参为结构体简单变量/数组元素



7.2.7.2. 形参为结构体变量的指针

★ 对应实参为结构体简单变量的地址/一维数组名

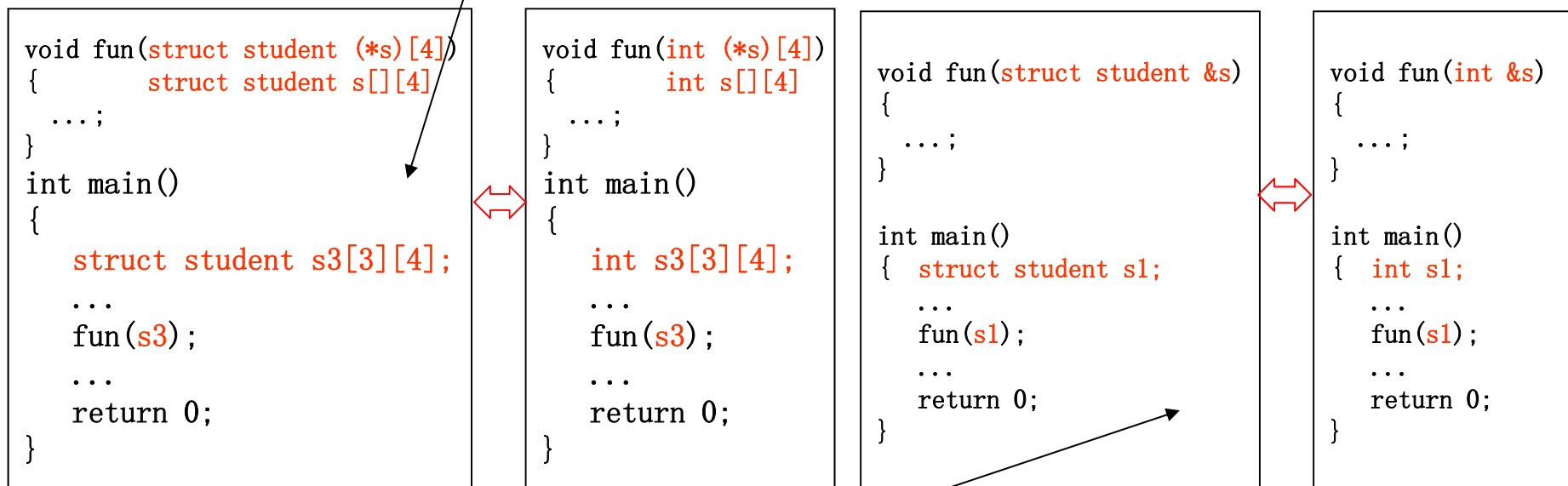
§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.7. 结构体数据类型作为函数参数

7.2.7.3. 形参为结构体数组的指针

★ 对应实参为结构体二维数组名



7.2.7.4. 形参为结构体的引用声明

★ 对应实参为结构体简单变量

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部, 也可以放在函数内部 (具体定义及访问规则见 7.2.8)

- ★ 函数内部:
 - 可定义结构体类型的各种变量/成员级访问
- ★ 函数外部:
 - 从定义点到本源程序文件的结束前:
 - 可定义结构体类型的各种变量/成员级访问
 - 其它位置 (本源程序定义点前/其它源程序):
 - 有该结构体的提前声明:
 - 仅可定义指针及引用/整体访问
 - 有该结构体的重复定义:
 - 可定义结构体类型的各种变量/成员级访问

类似外部全局变量概念, 但不完全相同

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况一: 定义在函数内部

```
#include <iostream>
using namespace std;

void fun(void)
{ struct student {
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};

    struct student s1, s2[10], *s3;
    s1.num = 10;
    s2[4].age = 15;
    s3 = &s1;
    s3->score = 75;
    s3 = s2;
    (s3+3)->age = 15;
}
```

正确

```
int main()
{
    struct student s;
    s.age = 15;

    return 0;
}
```

不正确

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况二: 定义在函数外部,从定义点到本源程序结束前

<pre>#include <iostream> using namespace std; struct student { int num; char name[20]; char sex; int age; float score; }; void f1(void) { struct student s1, s2[10], *s3; s1.num = 10; s2[4].age = 15; s3 = &s1; s3->score = 75; s3 = s2; (s3+3)->age = 15; }</pre>	<div>都正确</div> <pre>void f2(struct student *s) { s->age = 15; } struct student f3(void) { struct student s; ... return s; } int main() { struct student s1, s2; f1(); f2(&s1); s2 = f3(); return 0; }</pre>
---	--

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况三: ex1.cpp和ex2.cpp构成一个程序, 无提前声明

<pre>/* ex1.cpp */ #include <iostream> using namespace std; void f1() { 不可定义/使用student型各种变量 ✕ } struct student { ...; }; int fun() { 可定义student型各种变量, 访问成员 ✓ } int main() { 可定义student型各种变量, 访问成员 ✓ }</pre>	<pre>/* ex2.cpp */ #include <iostream> using namespace std; int f2() { 不可定义/使用student型各种变量 ✕ }</pre>
---	--

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况四: ex1.cpp和ex2.cpp构成一个程序, 有提前声明

```
/* ex1.cpp */
#include <iostream>
using namespace std;
struct student; //结构体声明
void f1(struct student *s1)
{
    s1->age;
}
void f2(struct student &s2)
{
    s2.score;
}
struct student {
    ...;
};
int main()
{
    可定义student型各种变量, 访问成员
}
```

允许

不允许

```
/* ex2.cpp */
#include <iostream>
using namespace std;
struct student; //结构体声明

void f2()
{
    struct student *s1;
    struct student s3, &s2=s3;
    s1.age = 15;
}
```

允许

不允许

虽可定义指针/引用,但不能进行成员级访问,无意义

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况四(变化1): ex1.cpp和ex2.cpp构成一个程序, 有提前声明

```
/* ex1.cpp */
#include <iostream>
using namespace std;
struct student; //结构体声明
void f1(struct student *s1)
{
    s1->age;
}
void f2(struct student &s2)
{
    s2.score;
}
struct student {
    ...;
};
int main()
{
    可定义student型各种变量, 访问成员
}
```

允许

不允许

```
/* ex2.cpp */
#include <iostream>
using namespace std;

void f2()
{
    struct student *s1;
}

void f3()
{
    struct student; //结构体声明

    struct student *s1;
    s1->age = 15;
}
```

不允许

允许

不允许

虽可定义指针/引用,但不能进行成员级访问,无意义

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

情况五: ex1.cpp和ex2.cpp构成一个程序, 有重复定义

```
/* ex1.cpp */
#include <iostream>
using namespace std;

struct student { //结构体定义
    ...;
};

int fun()
{
    可定义/使用student型各种变量 ✓
}

int main()
{
    可定义/使用student型各种变量 ✓
}
```

```
/* ex2.cpp */
#include <iostream>
using namespace std;

struct student { //结构体定义
    ...;
};

int f2()
{
    可定义/使用student型各种变量 ✓
}
```

本质上是两个不同的结构体
struct student, 因此即使
不完全相同也能正确, 这样
会带来理解上的偏差

§ 7. 用户自定义数据类型

7.2. 结构体类型

7.2.8. 结构体类型在不同位置定义时的使用 (补7.2.2)

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部(具体定义及访问规则见 7.2.8)

问题: 如何在其它位置访问定义和使用结构体?

```
/* ex.h */  
struct student { //结构体定义  
    ...;  
};
```

```
/* ex1.cpp */  
#include <iostream>  
#include "ex.h" ←  
using namespace std;  
  
int fun()  
{  
    可定义/使用student型各种变量 ✓  
}  
  
int main()  
{  
    可定义/使用student型各种变量 ✓  
}
```

```
/* ex2.cpp */  
#include <iostream>  
#include "ex.h" ←  
using namespace std;  
  
int f2()  
{  
    可定义/使用student型各种变量 ✓  
}
```

解决方法: 在头文件中定义