Ch-07

自定义数据类型

主要内容

- 结构体
- 共用体
- 枚举类型
- 用typedef声明新类型名

7.1 结构体

7.1.1 结构体的定义

结构体是由相互关联的不同数据类型的数据组成的有机整体，是用户根据自己的需要自定义的一种构造类型数据。

构成结构体的各个数据项称为结构体成员。

7.1.2 结构体类型的声明

```
struct [结构体名]
{
    类型标识符 成员名;
    类型标识符 成员名;
    ...
};
```

结构体名用户定义的合法标识符。可省略，即无名结构体。

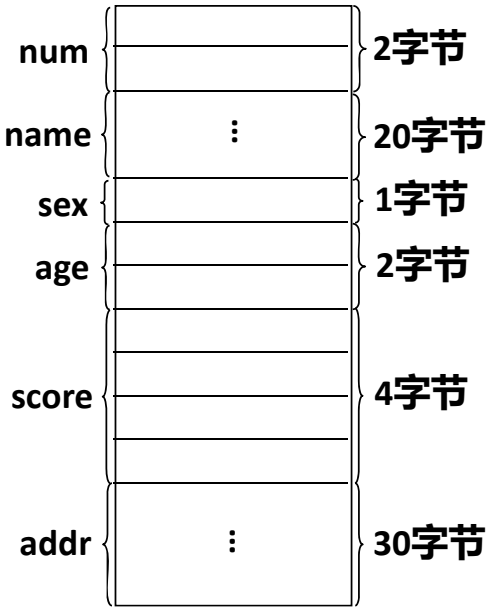
{ }中是组成该结构体的成员列表。成员的数据类型可以是基本型或构造型。

struct是关键字，不能省略。

末尾分号不能省略。

例7.1 用结构体类型定义学生学籍的数据结构

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



说明：

- **结构体类型声明只是定义了一种新的数据类型，类似int等。**
- **它是对结构组织形式的描述，编译系统还没分配实际的内存空间。只有定义结构体类型的变量，才分配内存空间。**

7.1.3 结构体变量的定义

声明结构体类型后，就可以定义结构体变量，共有三种变量定义形式：

□ 先声明结构体类型，再定义变量

```
struct 结构体名
{
    类型标识符 成员名;
    类型标识符 成员名;
    ...
};
struct 结构体名 变量名表列;
```

例如：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};  //结构体类型的声明

struct student stu_1, stu_2;  //结构体变量的定义
```

□ 声明结构体类型的同时定义结构体变量

```
struct 结构体名
{
    类型标识符 成员名;
    类型标识符 成员名;
    ...
}变量名表列；
```

例如：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu_1, stu_2;
```

□ 直接定义结构体变量（即省略结构体名）

```
struct
{
    类型标识符 成员名;
    类型标识符 成员名;
    ...
}变量名表列；
```

例如：

```
struct
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu_1, stu_2;
```

说明：用无名结构体直接定义变量只能一次。

说明：

- **结构体类型与结构体变量不同；**

数据类型定义/声明	**不分配内存**	**不能赋值、存取、运算**
变量定义	**分配内存**	**可以赋值、存取、运算**

- **结构体变量中的成员可单独使用，方法如普通变量；**
- **结构体可嵌套；**
- **结构体变量需要的存储空间大小等于其所有成员所占存储空间之和；**
- **结构体成员名与程序中的变量名可以相同，但两者不代表同一个对象。**

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

int num;
```

7.1.4 结构体变量的初始化和引用

7.1.4.1 结构体变量的初始化

- 形式1

```
struct 结构体名
{
    类型标识符  成员名;
    类型标识符  成员名;
    ...
};
struct 结构体名 结构体变量 = {初值表列};
```

例如：

```
struct student
{
    int num;
    char  name[20];
    char sex;
    int age;
    char addr[30];
};
struct student stu_1 = {100102, "WangLin", 'M', 20, "Beijing"};
```

□ 形式2

```
struct 结构体名
{
    类型标识符  成员名；
    类型标识符  成员名；
    …
}结构体变量 = {初值表列}；
```

例如：

```
struct student
{
    int num;
    char  name[20];
    char sex;
    int age;
    char addr[30];
} stu_1 = {100102, "WangLin", 'M', 20, "Beijing"};
```

□ 形式3

```
struct
{
    类型标识符  成员名；
    类型标识符  成员名；
    …
}结构体变量 = {初值表列}；
```

例如：

```
struct
{
    int num;
    char  name[20];
    char sex;
    int age;
    char addr[30];
} stu_1 = {100102, "WangLin", 'M', 20, "Beijing"};
```

7.1.4.2 结构体变量的引用

□ 引用规则

一般情况下结构体变量不能整体引用，只能引用结构体变量的成员。

□ 结构体变量成员的引用形式

结构体变量名.成员名

其中，“.”是成员（分量）运算符，优先级: 2，结合性：从左向右。

例如：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu_1, stu_2;

stu_1.num = 10;
stu_1.score += stu_2.score;
stu_1.age++;
```

- **说明：**
  - **不能将一个结构体变量作为一个整体进行输入和输出，只能对各个成员分别输入输出。**
  - **具有相同结构体类型的结构体变量之间可以进行整体赋值。**

□ 结构体类型的嵌套

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

或

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

struct student
{
    int num;
    char name[20];
    struct date birthday;
} stu;
```

说明：结构体嵌套时要逐级引用成员，只能对最低级的成员进行赋值、存取和运算。

例如 **stu.birthday.month = 3;**

7.1.5 结构体数组

□ 定义

由若干个具有相同结构体类型的结构体变量构成，每个数组元素都具有全部的成员项。

□ 定义形式

形式1：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
struct student stu[3];
```

形式2：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu[3];
```

形式3：

```
struct
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu[3];
```

□ 结构体数组的初始化

形式1：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
};
struct student stu[3] = {{100,“ Wang Lin” ,'M',20},
                         {101,"Li Gang",'M',19},
                         {110,"Liu Yan",'F',19}};
```

形式2：

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
} stu[3] = {{100,“ Wang Lin” ,'M',20},
            {101,"Li Gang",'M',19},
            {110,"Liu Yan",'F',19}};
```

- **说明：**

  1、全部元素初始化时，数组长度可省略；

  2、按元素顺序初始化时，内层括号可以省略。

7.1.6 结构体与指针

□ 指向结构体变量的指针

指向结构体变量存储空间的起始地址。

- **定义形式**

struct 结构体名 *指针名;

例如 struct student *p;

- **成员引用形式**
  - **(*结构体指针名).成员名**
  - **结构体指针名->成员名**
  - **结构体变量名.成员名**

说明：“->”指向运算符，优先级: 2级，结合方向：从左向右。

□ 用结构体变量和指向结构体的指针作函数参数

- **用结构体变量的成员作函数实参——值传递**
  - **注意形、实的类型要一致。**
- **用指向结构体变量（数组）的指针作实参——地址传递**
  - **传递的是结构体变量的首地址。**
- **用结构体变量作参数——（多）值传递，效率低**
  - **将结构体变量所有成员的值按照顺序传递给形参，要求形参与实参个数、类型相同。**

7.2 共用体

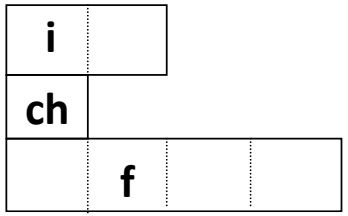
有时需要将几个不同时出现的变量共享一个内存单元，如：将一个整型变量、实型变量、字符型变量共同放入同一个地址空间（当然这几个变量不能同时用），怎么办？

C++提供了一种构造类型——共用体（联合体）类型支持。

7.2.1 共用体类型

```
union [共用体名]
{
    类型标识符 成员名;
    类型标识符 成员名;
     …
 } ;
```

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



7.2.2 共用体变量的定义

形式1:

```
union data
{
    int i;
    char ch;
    float f;
};
union data a, *p, d[3];
```

形式2:

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

形式3:

```
union
{
    int i;
    char ch;
    float f;
} a, b;
```

口 共用体变量的特点

共用体变量的几个成员共用一段内存空间。

- 共用体变量所占内存空间大小是多少？

使用最大内存空间的成员所占的内存空间字节数

- 共用体变量的几个成员能同时存在吗？

不能，一个时刻只能有一个成员存在。

- 共用体变量成员不能同时存在，那当前起作用的是哪个成员？

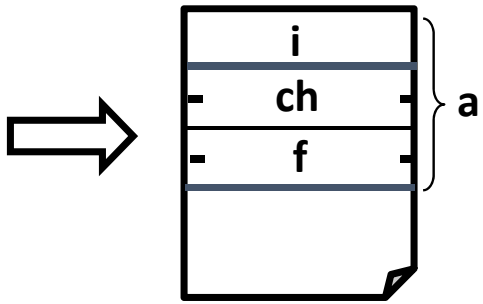
起作用的成员是最后一次存储的成员。

- 共用体变量和其成员是否具有相同的地址？

共用体变量和它的各成员的地址是相同的。

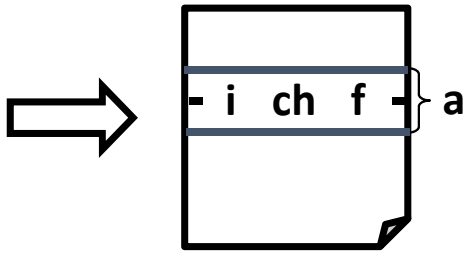
□ 共用体变量和结构体变量的比较

```
struct data
{
      int i;
      char ch;
      float f;
} a;
```



“结构体”变量a：
各成员同时存在，
占2+1+4=7个字节。

```
union data
{
      int i;
      char ch;
      float f;
} a;
```



“共用体”变量a：
任一时刻只有一个成员存在，
占4个字节（float型占4个字节）。

7.2.3 共用体变量的引用

口 三种等价的引用方式

- 共用体变量名.成员名
- 共用体指针名->成员名
- (*共用体指针名).成员名

```
union data
{
    int i;
    char ch;
    float f;
};
union data a, b, c, *p, d[3];
```

a.i a.ch a.f

p->i p->ch p->f

(*p).i (*p).ch (*p).f

d[0].i d[0].ch d[0].f

引用规则

- 不能引用共用体变量，只能引用其成员

```
union data
{
    int i;
    char ch;
    float f;
} a;
printf("%d", a);  ×
printf("%d", a.i);  √
```

- 不能对共用体变量赋值，也不能在定义共用体变量时初始化，但可以用一个共用体变量为另一个共用体变量赋值

```
union data
{
    int i;
    char ch;
    float f;
} a;
a = 1;  ×
```

```
union data
{
    int i;
    char ch;
    float f;
} a = {1, 'a', 1.5);  ×
```

```
float  x;
union
{
    int i;
    char ch;
    float f;
} a, b;
a.i = 1; a.ch = 'a'; a.f = 1.5;

b = a;
x = a.f;   √
```

- 不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针(与结构体变量的这种用法相仿)。
- 共用体类型可出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可出现在共用体类型定义中，数组也可作为共用体的成员。

7.3 枚举类型

□ 定义

将变量的值一一列举出来，变量的值只限于列举出来的值的范围之内。

□ 枚举类型的定义

```
enum [枚举名] {sun, mon, tue, wed, thu, fri, sat};
```

其中 sun, mon, … , sat称为枚举元素或枚举常量，它们是用户定义的标识符。

□ 枚举变量的定义

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

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

□ 关于枚举类型的说明

- **在编译中，对枚举元素按照常量处理。**

```
enum weekday {sun, mon, tue, wed, thu, fri, sat};
sum = 0; mon = 1; //对sun、mon的赋值是错误的，因为枚举元素是常量，不能在定义之外的其他时候被赋值
```

- **枚举元素是有值的，按定义时的顺序使它们的值依次为0，1，2，…。**
  也可以在定义时由程序员指定枚举元素的值。

```
enum weekday {sun, mon, tue, wed, thu, fri, sat}; //sun ~ sat的值依次为: 0, 1, 2, 3, 4, 5, 6

enum weekday {sun, mon = 8, tue, wed, thu = 100, fri, sat};
//sun ~ sat 依次为: 0, 8, 9, 10, 100, 101, 102
```

- **枚举值可以用来作判断比较，例如 if(workday == mon) …；**

- **枚举变量要用枚举元素进行赋值，若使用整数进行赋值，则需要对整数进行强制类型转换。**

```
enum weekday i, j;
i = mon;
j = (enum weekday)2; //相当于将顺序号为2的枚举元素赋给j
```

7.4 用typedef声明新类型名

□ typedef语句的一般格式

```
typedef  已有类型名  新类型名;
```

- 作用：用新类型名代替已有类型名。
- 说明：新类型名是用户自定义的标识符，而已有类型名可以是：基本数据类型、指针、结构体、共用体、枚举类型。
- 例如：type float REAL; 此后可以使用REAL代替float进行变量定义。
  即 REAL a, b; 等价于 float a, b;

- **说明:**

1、typedef 没有创造新数据类型；

2、typedef 是声明类型，不能定义变量；

3、typedef 与 define 不同；

- **define 预编译时处理，是简单字符置换；**
- **typedef 编译时处理，是已有类型的重命名 。**

4、使用 typedef 有利于程序的通用与移植。

例如：VC中int占4个字节；而TC中int占2个字节，long占4个字节。

若VC下的程序要移植到TC下，对于原来int类型的变量要改成long类型。多处修改时不方便，此时可自己定义一个INTEGER来表示4个字节：

在VC下 typedef int INTEGER；

在TC下 typedef long INTEGER。

□ 用typedef声明类型的步骤

① 按照定义变量的方法写出变量的定义，如：int i;

② 将变量名换成新类型名，如：int INTEGER;

③ 最前面加上typedef，如：typedef int INTEGER;

④ 用新类型名定义变量，如：INTEGER i, j;

声明数组类型

① int a[10];

② int ARRAY[10];

③ typedef int ARRAY[10];

④ ARRAY a, b;

⇔ int a[10], b[10];

声明指针类型

① char *str;

② char *STRING;

③ typedef char *STRING;

④ STRING p, s[10];

⇔ char *p, *s[10];

声明函数指针类型

① int (*p)();

② int (*POWER)();

③ typedef int (*POWER)();

④ POWER p1, p2;

⇔ int (*p1)(), (*p2)();

声明结构体类型

① struct date
{ int month;
int day;
int year;
} d;

② struct date
{ int month;
int day;
int year;
} DATE;

③ typedef struct date
{ int month;
int day;
int year;
} DATE;

④ DATE birthday, *p;

⇔ struct date
{ int month;
int day;
int year;
} birthday, *p;

- typedef 声明类型可以嵌套

```
例  typedef  struct club
    {
        char  name[20];
        int  size;
        int  year;
    } GROUP;
    typedef   GROUP  *PG;
    PG    pclub;
```

GROUP为结构体类型
PG为指向GROUP的指针类型

⇔ GROUP *pclub;
⇔ struct club *pclub;

CH07 End