4.4 结构体 (python中的 class 对应到C++中的 struct)

结构体的使用场景:

- 需要一种数据结构, 可以同时将多种不同类型的数据组合在一起
  - 数组不能完成这个任务, 虽然数组能存储多个元素, 但是数组中只能存储同一类型的数据
- 结构体是一种用户自定义的数据类型, 定义好之后, 我们直接把它当做 `int` 那样使用就好了, 很方便.

书中结构体的定义例子:

- 结构体花括号内部的每一行都是一条语句, 语句的结尾记得使用分号`;`.
- 结构体花括号外面也不要漏掉分号`;`.

例如，假设 Bloataire 公司要创建一种类型来描述其生产线上充气产品的成员。具体地说，这种类型应存储产品名称、容量（单位为立方英尺）和售价。下面的结构描述能够满足这些要求：

```
struct inflatable   // structure declaration
{
    char name[20];
    float volume;
    double price;
};
```

关键字 struct 表明，这些代码定义的是一个结构的布局。标识符 inflatable 是这种数据格式的名称，因此新类型的名称为 inflatable。这样，便可以像创建 char 或 int 类型的变量那样创建 inflatable 类型的变量了。接下来的大括号中包含的是结构存储的数据类型的列表，其中每个列表项都是一条声明语句。这个例子使用了一个适合用于存储字符串的 char 数组、一个 float 和一个 double。列表中的每一项都被称为结构成员，因此 infatable 结构有 3 个成员（参见图 4.6）。总之，结构定义指出了新类型（这里是 inflatable）的特征。

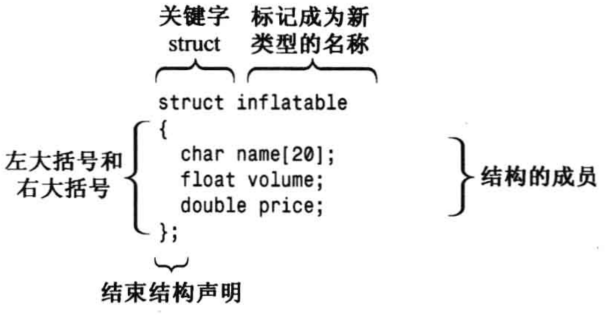

图 4.6 结构描述的组成部分

C语言的结构体与C++结构体在使用时的区别:

- C语言的结构体在使用时, 需要在结构体名字前面加上`struct`关键字, 但是C++中不需要.
  - 如: `struct Student stu; // C语言写法` 与 `Student stu; // C++ 写法` 效果是一样的.

访问结构体的成员, 使用成员运算符 . (点号)

- 如: `stu.age = 20;`

代码例子1 --> 使用结构体

```
#include <iostream>

// 结构体定义, 通常写在 main 函数外面(称为外部声明), 好处是整个文件都能使用这个结构体
struct inflatable
{
    char name[20];  // 这是一个语句, 别忘了分号
    float volume;
    double price;

};                  // 这是一个语句, 别忘了分号

int main(void)
{
    using namespace std;

    //使用定义好的结构体
    struct inflatable guest =                   // 使用C语言的写法, 前面加上 struct
    {
        "Glorious Gloria",  // 给结构体成员赋值的时候, 不再是一条条语句, 所以这里用的是逗号 `,`
        1.88,
        29.99               // 最后已经没有值了, 所以最后的分号可以省略 (把这个想象成花括号赋值的语法)
    };              // 这是一个语句, 别忘了分号

    inflatable pal =                            // 使用 C++ 的写法, 省略struct
    {
        "Audacious Arthur",
        3.12,
        32.99
    };              // 这是一个语句, 别忘了分号

    cout << "Expand your guest list with " << guest.name << " and " << pal.name << "!\n";
    cout << "You can have both for $" << guest.price + pal.price << "!\n";

    return 0;
}
```

代码例子2 --> 结构体中包含string类的成员(注意 std 命名空间语句)

```
#include <iostream>
#include <string>

using namespace std;      // 命名空间如果写在这里, 之后的代码都能使用这个命名空间, 结构体中也就可以省略 std:: 这部分


// 结构体定义, 通常写在 main 函数外面(称为外部声明), 好处是整个文件都能使用这个结构体
struct inflatable
{
    // char name[20];  // 别忘了分号
    // std::string name;   // 使用 string 类型, 需要包含头文件 <string>, 如果std命名空间没有写在结构体定
```

```
义之前，就需要加上 std:: 这部分
    string name;
    float volume;
    double price;

};                  // 别忘了分号

int main(void)
{
    // using namespace std;

    //使用定义好的结构体
    struct inflatable guest =                   // 使用C语言的写法，前面加上 struct
    {
        "Glorious Gloria",  // 给结构体成员赋值的时候，不再是一条条语句，所以这里用的是逗号 `,`
        1.88,
        29.99               // 最后已经没有值了，所以最后的分号可以省略（把这个想象成花括号赋值的语法）
    };              // 别忘了分号

    inflatable pal =                            // 使用 C++ 的写法，省略struct
    {
        "Audacious Arthur",
        3.12,
        32.99
    };              // 别忘了分号

    cout << "Expand your guest list with " << guest.name << " and " << pal.name << "!\n";
    cout << "You can have both for $" << guest.price + pal.price << "!\n";

    return 0;
}
```

代码例子3 --> 结构体对象之间的直接赋值

- 结构体中虽然包含了字符串数组，但是结构体对象之间支持直接赋值，赋值之后，结构体里的字符串数组也会被赋值.

```
#include <iostream>
#include <string>

using namespace std;

struct inflatable
{
    char name[20];    // 注意这里是字符串数组，数组不能直接给数组赋值
    float volume;
    double price;

};

int main(void)
{
    inflatable bouquet =
    {
        "sunflowers",
        0.20,
        12.49
```

```
    };

    inflatable choice;      // 只进行了声明，没有进行初始化

    cout << "bouquet: " << bouquet.name << " for $" << bouquet.price << endl;

    // 当结构体一样时，可以直接将结构体赋值给结构体.（结构体对象之间支持直接赋值）
    choice = bouquet;     // 将 bouquet 的值直接赋值给 choice，  注意，结构体内包含了字符串数组，但是此时也
可以直接进行赋值
    cout << "choice: " << choice.name << " for $" << choice.price << endl;

    return 0;
}
```

代码例子4 --> 结构体定义与结构体对象的声明可以直接写到一起

```
#include <iostream>
int main(void)
{
  struct perks
  {
    int key_number;
    char car[12];
  } mr_smith, ms_jones;  // 结构体定义与结构体对象的声明可以直接写到一起

  return 0;
}
```

甚至可以把结构体的定义, 结构体对象的声明, 结构体对象的初始化都写到一起 (很丑...不推荐)

```
#include <iostream>
int main(void)
{
  struct perks
  {
    int key_number;
    char car[12];
  } mr_glitz = {7, "Packard"};  // 结构体定义，结构体对象的声明 以及该对象的初始化 可以直接写到一起
}
```

代码例子5 --> 结构体声明时, 可以省略结构体名, 直接声明结构体对象

- 此时, 结构体对象可以正常使用, 但是我们再也无法创建相同的结构体对象了, 因为没有结构体名

```
#include <iostream>
int main(void)
{
  struct
  {
    int key_number;
    char car[12];
```

```
    } mr_glitz = {7, "Packard"};  // 结构体声明时，可以省略结构体名，直接声明结构体对象
}
```

4.4.5 结构体数组

- 结构体数组是一个数组，每个元素都是一个结构体对象
    - 如：`inflatable guests[2];` --> `guests` 是一个数组，包含2个元素，数组中的每个元素都是一个结构体对象
- 结构体数组的初始化
    - 数组的赋值需要使用一个花括号，把数组的的元素值给括起来
        - 由于结构体数组的每一个元素都是结构体对象，所以每个元素的初始化也需要使用一个花括号，把结构体对象的成员值给括起来
        - 如：`inflatable guests[2] = {{"Bambi", 0.5, 21.99}, {"Godzilla", 2000, 565.99}};`

代码例子1 --> 结构体数组的简单操作

- 如果结构体数组中有元素没有初始化的话，里面的数值成员会变成0，字符串的话会变成空字符串.

```
#include <iostream>

using namespace std;
struct inflatable
{
    char name[20];
    float volume;
    double price;
};

int main(void)
{
    inflatable guest[3] =
    {
        {"Glorious Gloria", 1.88, 29.99},
        {"Godzilla", 2000, 565.99}            // 实验结果表明，如果结构体数组中有元素没有初始化的话，里面的数值成员会变成0，字符串的话，会变成"空字符串"
    };
    cout << "The guests " << guest[0].name << " and " << guest[1].name << " have a combined volume of "
        << guest[0].volume + guest[1].volume << " cubic feet.\n\n";

    // 下面测试一下结构体数组中没有初始化的元素的值
    cout << "The elements of the Third guest are: \n"
        << "Name: " << guest[2].name << "\n"
        << "volume: " << guest[2].volume << "\n"
        << "price: " << guest[2].price << ".\n";

    return 0;
}
```

4.4.6 结构体的位字段 (用在低级编程中)

位字段通常在与某些硬件设备上的寄存器对应的数据结构上, 用于表示寄存器中的各个位的含义.

- 通常用冒号:来表示位字段的长度

4.4.6 结构中的位字段

与 C 语言一样，C++也允许指定占用特定位数的结构成员，这使得创建与某个硬件设备上的寄存器对应的数据结构非常方便。字段的类型应为整型或枚举（稍后将介绍），接下来是冒号，冒号后面是一个数字，它指定了使用的位数。可以使用没有名称的字段来提供间距。每个成员都被称为位字段（bit field）。下面是一个例子：

```
struct torgle_register
{
    unsigned int SN : 4;    // 4 bits for SN value
    unsigned int : 4;       // 4 bits unused
    bool goodIn : 1;        // valid input (1 bit)
    bool goodTorgle : 1;    // successful torgling
};
```

可以像通常那样初始化这些字段，还可以使用标准的结构表示法来访问位字段：

```
torgle_register tr = { 14, true, false };
...
if (tr.goodIn)    // if statement covered in Chapter 6
...
```

位字段通常用在低级编程中。一般来说，可以使用整型和附录 E 介绍的按位运算符来代替这种方式。