教学目标

了解结构体的意义及写法；

理解结构体的操作方式；

感受构造函数的便利性，学会用构造函数给结构体赋值；

理解结构体数组；

感受结构体数组排序的魅力，并能够编程进行结构体排序；

〇、教学引入

生活中很多事物并不只有一个信息，比如班上同学，每个人的基本信息包括：学号、姓名、性别、出生日期、身高、体重等。很多时候我们希望把事物的一组信息放在一起，这样信息更加完整，也更便于管理。

之前我们学过数组，对于上述问题，我们当然也可以开6个数组来存储，但是，这样也会给后续操作带来一些麻烦。如果要按出生日期对班上同学进行排序，那么每次需要调整数组元素的时候，我们同时要操作6个数组——这就很繁琐了；如果每个人的信息更多，比如有20项，那这项工作简直要让你抓狂——繁琐、易错、难检查、代码冗长！

为解决这个问题，C++为我们提供了一种基本数据结构——**结构体，**其作用是可以把一组信息封装起来。

一、结构体的基本写法

对于上面问题，我们通过以下写法就可以把这些信息放在一起管理。

```
struct node
{
    int id;
    string name;
    bool gender;
    string date;
    int height, weight;
};
```

现在我们拥有了一个自己定义的新的类型——`node`（当然，你也可以换其他类型名，比如`student`），这个类型包含了6个信息，使用方式如下：

```
node a, b, c; // 声明3个类型为node的变量
a.name = "Wang Anshi"; // 给a的name成员变量赋值
cout << a.name; // 输出a的成员变量
b = a; // 把结构体a整体赋值给b
```

你可以结合字符串 `string` 类型来理解这里 `.` 的含义，在C++中，我们可以通过 `.` 运算符来操作对象的成员。`string` 的 `.size()` 就是一个字符串类型的成员函数。同样，如果有需要，我们也可以给结构体定义成员函数，其操作的数据仅限于结构体本身。

二、给结构体整体赋值

当前我有这样一条信息，需要使用结构体存储：

> 学号：13；
>
> 姓名:Zhang Wei；
>
> 性别：女；
>
> 出生日期：2003-07-12；
>
> 身高：160cm；
>
> 体重:45kg

我们如何存入结构体中呢？

按照上一节介绍的写法，我们当然可以这样来写：

```
node a;
a.id = 13;
a.name = "Zhang Wei";
a.gender = 0; // 0女，1男
a.date = "2003-07-12";
a.height = 160;
a.weight = 45;
```

但是，这样仍显得有些繁琐，如果类似的赋值操作在一个程序中多次出现，代码也会变得冗长。

方法1. 使用构造函数

接下来介绍一种写法，通过构造函数给结构体成员赋值：

```
struct node
{
    int id;
    string name;
    bool gender;
    string date;
    int height, weight;
    node(){} //保留的构造函数，以保证在没有参数传入的时候不出错
    node(int _id, string _name, bool _gender, string _date, int _height, int _weight)
    {
        id = _id;
        name = _name;
        gender = _gender;
        date = _date;
```

```
        height = _height;
        weight = _weight;
    }
};
```

注意：构造函数没有返回类型，函数名与结构体名字一致。

可能有同学会说：这不还是要写很长吗？

我的回答是，如果你看了下面的写法，你就知道构造函数有什么用了，

```
node a = node(13, “Zhang Wei”, 0, “2003-7-12”, 160, 45);
```

一行搞定！当这样的整体赋值操作在程序中多次出现，构造函数就明显地起到了精简代码的作用。

方法2. 使用大括号构造数据

此外，还有没有更简单的整体赋值操作呢？

答案是有的：参考如下

```
struct node
{
    int id;
    string name;
    bool gender;
    string date;
    int height, weight;
};

int main()
{
    node a = (node){13, "Zhang Wei", 0, "2003-7-12", 160,122};
    cout << a.height;
}
```

大家注意到通过 `{}` 构造的数据需要强制转化为你定义的结构体类型，形如 `(node){...}` ，否则，有些评测系统可能会识别出错。

相比方法1而言，这种方法是不是简洁更多？

你是不是要问，那为什么还要构造函数呢？事实上，使用大括号构造结构体数据的时候，C++会去调用默认的构造函数来实现。了解程序原理，有助于我们日后的程序设计以及对算法的优化。

注意：结构体整体赋值，以上不管使用哪一种方式，参数序列一定要与结构体成员一一对应，否则会出错！

三、结构体数组排序

结构体作为自己定义的数据类型，也可以使用它来开数组，写法参照基本数据类型的数组写法。

现在，我们将使用上一节中引出的问题：对班上同学按照出生日期排序。

大家可以运用我们以前学过的几个排序算法来做这个事情，比较的时候只涉及出生日期，而交换或者移动数据的时候，则是操作这个结构体元素，这样就可以保证排序能够正确执行。

.

现在介绍一个C++内部提供的排序函数： `sort()`

调用格式：

写法1： `sort(第一个有效地址，第一个无效地址);`

写法2： `sort(第一个有效地址，第一个无效地址,比较函数);`

对于基本数据来说，可以使用前者进行升序排序（从小到大）；但是降序排序以及结构体排序，则需要使用后者，因为排序中元素两两比较的时候，需要知道两者**是否正序（不逆序）**！对于基本数据类型来说，默认前小后大为正序；但是， `sort()` 函数在遇到两个我们定义的结构体类型数据的时候，并不能识别出它俩是正序还是逆序，除非——**我们告诉它**！

方法如下：

写一个比较函数

```
bool cmp(node x, node y)
{
    return x.date < y.date;
}
```

然后传入给 `sort()` 函数，供其调用

```
sort(a, a + n, cmp);
```

这样就可以完成结构体的排序了（升序排序）

> 思考1分钟：
>
> 对于降序排序，cmp函数应该如何写？

补充说明1：

有些问题要求进行**稳定排序**，这时可以使用 `stable_sort()` 函数替代 `sort()` 函数，两者的使用规则完全一样。

所谓稳定排序，即关键字相同的元素，排序前后，不会改变元素间前后关系。

> 举个例子：A班同学上体育课时排成一列，现在体育老师要求大家按身高从小到大重新排列；小王和小军身高相同，小王原本排在小军前面；进行稳定排序后，小王也应该仍然排在小军前面。

补充说明2：

为了让排序进行得更快些，在比较函数中，我们通常会使用引用传参，而不是传值，如下：

```
bool cmp(node &x, node &y)
{
    return x.date < y.date;
}
```

更安全的写法如下：

```
bool cmp(const node &x, const node &y)
{
    return x.date < y.date;
}
```

> 思考1分钟，并回答：
>
> 以上写法有何不同？
>
> 后者写法会更加安全，你可以猜一下原因...