结构体(struct)的使用（在数据结构结构体的构造很重要)

到现在为止，我们可以很方便地定义单种我们想要的数据类型（如 `int` 型等），但是如果碰到这样一种情况：实现一个手机通讯录，需要以人为单位，且每个人的内部信息由姓名、年龄、手机号、住址之类的不同类型数据组成。这个时候如果使用单类型的变量进行罗列的话操作起来就不太方便，而这些功能使用结构体(`struct`)却可以很好地实现。结构体在上机考试的题中非常常用，可以将若干个不同的数据类型的变量或数组封装在一起，以储存自定义的数据结构，方便储存一些复合数据。下面我们来介绍结构体的使用。C语言这一点和c++不同，需要在前面增加 `typedef`

结构体的定义

定义一个结构体的基本格式如下：

```
struct name {
    // 一些基本数据
};
```

当我们需要将一些相关的变量放在一起存储时，我们只要依次写出它们的数据类型和变量名称。例如我们需要储存一个学生的学号，性别，姓名和专业，就可以这样定义：

```
struct studentInfo {
    int id;
    char gender; // 'F' or 'M', 性别的意思
    char name[20];
    char major[20];
} Alice, Bob, stu[1000];           //自定义表示变量
```

其中 `studentInfo` 是这个结构体的名字，内部分别定义了 `id`（学号）、`gender`（性别）、`name`（姓名）、`major`（专业），这些就是单个学生的信息。而在大括号外定义了 `studentInfo` 型的 `Alice` 和 `Bob`，代表两个结构体变量；之后的 `stu[1000]` 就是当有很多学生时定义的一个结构体数组（如果不在此处定义变量或数组，则大括号外直接跟上分号）。

定义结构体变量和结构体数组除了可以像上面直接定义外，也可以按照基本数据类型（如 `int` 型）那样定义：

```
studentInfo Alice;
studentInfo stu[1000];
```

需要注意，结构体里面能定义除了自己本身（这样会引起循环定义的问题）之外的任何数据类型。不过虽然不能定义自己本身，但可以定义自身类型的指针变量。例如：

```
struct node {
    node n; // 不能定义node型变量
    node* next; // 可以定义node*型指针变量
};
```

访问结构体内的元素

访问结构体内的元素有两种方法：`.`操作和`->`操作。现在我们把`StudentInfo`类型定义成下面这样：

```
struct studentInfo {
    int id;
    char name[20];
    studentInfo* next;
} stu, *p;                  //之后调用只与定义变量类型有关
```

这样`studentInfo`中多了一个指针`next`用来指向下一个学生的地址，且结构体变量中定义了普通变量`stu`和指针变量`p`。

于是访问`stu`中变量的写法如下：

```
stu.id
stu.name
stu.next
```

而访问指针变量`p`中元素的写法如下：

```
(*p).id
(*p).name
(*p).next
```

可以看到，对结构体变量和结构体指针变量内元素的访问方式其实是一样的，在变量名后面加`.`然后跟上要访问的元素即可。但是同时我们也会发现，对结构体指针变量中元素的访问写法略显复杂，所以 C 语言中又规定了访问结构体指针变量内元素的更简洁的写法：

```
p->id;
p->next;
p->next;
```

正如上面的写法，结构体指针变量内元素的访问只需要使用`->`跟上要访问的元素即可，且使用`*`或`->`访问结构体指针变量内元素的写法是完全等价的。

当然，我们可以给`stu.id`赋值或者把`stu.id`赋值给其他变量：

```
stu.id = 10086;
int getId = stu.id;
```

结构体的初始化

说到初始化，我们自然可以先定义一个 `studentInfo stu` 的结构体变量，然后对其中的元素挨个赋值以达到初始化的目的，就像下面这样：

```
stu.id = 1;
stu.gender = 'M';
```

或者在读入的时候进行赋值：

```
scanf("%d %c", &stu.id, &stu.gender);
```

但是这样做的话，当结构体内变量很多的时候并不方便，此处我们介绍一种使用“构造函数”（在901c++类中，构造函数是重点，到时会详细介绍）的方法来进行初始化，供学弟妹学习。所谓构造函数就是用来初始化结构体的一种函数，它直接定义在结构体中。构造函数的一个特点是它不需要写返回类型，且函数名与结构体名相同。

一般来说，对一个普通定义的结构体，它的内部会生成一个默认的构造函数（但不可见）。例如下面的例子中，`studentInfo(){}` 就是默认生成的构造函数，可以看到这个构造函数的函数名和结构体类型名相同，都是 `studentInfo`；它没有返回类型，所以 `studentInfo` 前面没有写东西；它没有参数，所以小括号内是空的；它也没有函数体，因此花括号内也是空的。由于这个构造函数的存在，我们才可以直接定义 `studentInfo` 类型的变量而不进行初始化（因为它没有让我们提供任何初始化参数）。

```
struct studentInfo {
    int id;
    char gender; // 默认生成的构造函数 studentInfo(){}
};
```

那么，如果想要自己手动提供 `id` 和 `gender` 的初始化参数，应该怎么做呢？很显然，只需要像下面这样提供初始化参数来对结构体内的变量进行赋值即可，其中 `_id` 和 `_gender` 都是变量名。只要不和已有的变量冲突，用 `a`、`b` 或者其他变量名也可以。

```
struct studentInfo {
    int id;
    char gender; // 下面的参数用以对结构体内部变量进行赋值
    studentInfo(int _id, char _gender) { // 赋值
        id = _id;
        gender = _gender;
    }
};   //不要丢掉分号
```

当然，构造函数也可以简化成一行：

```
struct studentInfo {
    int id;
    char gender;
    studentInfo(int _id, char _gender): id(_id), gender(_gender) {}
};
```

这样我们就可以在需要的时候直接对结构体变量进行赋值了：

```
studentInfo stu = studentInfo(10086, 'M');
```

注意，如果自己重新定义了构造函数，我们就不能不经初始化就定义结构体变量，也就是说默认生成的构造函数`studentInfo(){}`此时被覆盖了。为了既能不初始化就定义结构体变量，又能享受初始化带来的便捷，我们可以把`studentInfo(){}`手动加上。这意味着，只要参数个数和类型不完全相同，我们就可以定义任意多个构造函数，以适应不同的初始化场合，例如下面的例子。

```
struct studentInfo {
    int id;
    char gender; //用以不初始化就定义结构体变量
    studentInfo(){}
    //只初始化 gender
    studentInfo(char _gender) {
        gender = _gender;
    }
    //同时初始化 id 和 gender
    studentInfo(int _id, char _gender) {
        id = _id;
        gender = _gender;
    }
};
```

下面是一个应用实例，其中结构体`Point`存放平面点的坐标`x`、`y`。

```
#include <stdio.h>
struct Point {
    int x, y;
    Point(){}   // 用以不经初始化地定义 pt[10]
    Point(int _x, int _y): x(_x), y(_y) {}// 用以提供 x和y的初始化
}pt[10];
int main(){
    int num = 0;
    for(int i = 1; i <= 3; i++) {
        for(int j = 1; j <= 3; j++){
            pt[num++] = Point(i, j);    // 直接使用构造函数
        }
    }
    for(int i = 0; i < num; i++) {
        printf("%d,%d\n", pt[i].x, pt[i].y);
    }
    return 0;
}
```

构造函数在结构体内元素比较多的时候会使代码显得精炼，因为可以不需要临时变量就初始化一个结构体，而且代码更加工整，推荐使用。