1、结构体类型

不能在定义结构体类型的时候给成员赋值。

```
struct studentT {
    char *name = new char[20]; // 错误！
    int English = 60; // 错误！
    int Math = 60; // 错误！
};
```

【注意 **struct** 的拼写，别拼错了】

成员不能只有类型而没有变量名。【注意区分结构体类型名和变量名】

```
struct studentT {
    char name [20];
    DateT birthday; // DateT 是类型名，birthday 是变量名
};
```

2、结构体变量

一个结构体变量的各个成员在内存中是连续存储的。
结构体变量的初始化：studentT student1 = {“Jack”, 82, 75}; // 类似于一维数组

同类型的结构体变量之间可以相互赋值，例如 student1 = student2;
本质上是把 student2 在内存中的每个字节按顺序复制到 student1 的内存空间里。

3、结构体指针

对于 student *p = &student1; 可以用(*p).name 或 p->name 来访问 student1.name。
【如果用(*p).name，不能漏掉小括号】
注意 p->name 的意思不是指针 p 指向 name。指针 p 是指向 student1 的。
【->是所有运算符中优先级最高的】

4、结构体数组

结构体数组：studentT students[50]; // students[0].name 表示第 0 个元素的 name

5、结构体与函数

如果将结构体变量作为函数的参数，那么在调用时执行值传递（实参被整体复制给形参）。
结构体常用 const 限定的引用传递：节省内存、调用速度快、实参不会被函数修改。

函数可以返回一个结构体的变量、指针或引用。
当返回的是结构体类型的指针或引用时，指向的结构体变量**不能**是该函数的局部变量。

6、单链表

链表中的每个节点都是一个结构体变量，其成员包含指向该结构体类型的指针。
表示链表结点的结构体变量通常通过 new 来动态分配内存空间。

创建单链表时，先创建头结点，然后依次创建后续的各个结点。
单链表的头结点一般不存储数据，仅用来为第一个存储数据的结点提供前驱结点。
与数组相比，单链表查询元素的效率较低，但增加、删除元素的效率更高，且不会填满。