结构体是一种自定义数据结构。

结构体的主要作用有两个：

1、用一个结构体描述一个事物。

2、用一个结构体对相关变量进行打包。

结构体定义：结构体的关键字是struct，示例：

```
struct Test{
    int a;
    char b;
};
```

上例中，struct Test就是自定义类型的名字。可以用struct Test作为类型直接定义变量。如：struct Test t，则t就是定义出的结构体变量。t可以通过大括号初始化：`t = {5, 'a'};`

匿名结构体：

声明结构体的时候可以不给结构体起名：

```
struct{
    int x;
    int y;
}point;
```

这种情况下会如上面的代码一样直接定义变量，这类结构体往往只用于定义单个结构体变量。

typedef：用于给变量类型起别名。方法是——先用打算起别名的变量类型定义一个变量，再在这个定义的前面加上typedef，则刚刚起的变量名，就成了这个变量类型的别名。

通过typedef起别名：

```
typedef signed int s32;
typedef int arr_10[10];
typedef int* pint;

//结构体
typedef struct{
    int x;
    int y;
}Point;
```

typedef和#define的不同：typedef是直接给变量类型起别名，#define是查找替换，例如下面场景就会产生差异：

```
#define PINT int *
typedef int* Pint;

PINT a, b;
Pint c, d;
```

上述定义中，b不是`int *`类型，而其他都是。这是由于#define是单纯的查找替换而带来的理解歧义。

结构体取元素运算符（点运算符）：运算符“.”作用于结构体变量，后面跟成员名，取出该结构体的对应成员。

结构体指针取元素运算符（箭头运算符）：运算符“->”作用于结构体指针变量，后面跟成员名，取出该指针指向的结构体的对应成员。

※箭头运算符是原生的，点运算符是语法糖，类似*(p+1)和p[1]的关系。

结构体大小：是所有成员的总和，但是长度要向最长的成员对齐。如果连续几个成员加起来也没有对齐标准的成员长，则共用一个成员的空间。

※安排结构体成员时，要从小到大或者从大到小排列，避免无意义的浪费。

柔性数组：结构体的最后一个元素可以是一个大小为0的数组：

```
struct data{
    int n;
    int arr[0];
};
```

这样的写法是方便整个结构体用下面的方式一次性申请释放：

```
struct data *sd = (struct data*)malloc(sizeof(struct data) + 100 * sizeof(int));
......
free(sd);
```

如果arr是指针，则必须申请两次空间，这种申请方式即避免了申请两次空间，还能节省出一个指针的空间。

※在C89标准中，不允许大小为0的数组出现，在部分编译环境下，这种0长度的数组无法通过编译，这种情况下，将0改为1即可。

位段：结构体里允许一个成员使用固定的几位：

```
struct date{
    unsigned short year:8;
    unsigned short month:4; //只用4位
    unsigned short day:5; //只用5位
};
```

同理，连续的多个位段元素如果加起来没有用到一个对齐元素的长度，则不开创其他空间。上面的year是8位，month是4位，不到一个short的16位，可以共用一个short，但day加上以后超过了16位，所以day单独用一个short。所以一般位段也要要求按照从小到大或者从大到小的次序排列。

※位段不能定义数组。

结构体自包含：结构体里不能包含自身结构，例如下面的写法就是不合法的——

```
struct Test{
    int p;
    struct Test t;
};
```

这个结构体如果定义了变量`struct Test u`，那么u里面有t，t里还有t，无穷无尽，无法确定结构体大小，所以不允许存在。

但结构体里可以包含自身结构的指针——

```
struct Test{
    int p;
    struct Test *next;
};
```

这个结构体里，next没有指向，所以没有空间，不存在递归套娃。指针大小是固定的，所以这个结构体有固定大小，允许存在。

联合体（也叫共用体）是一种自定义数据结构，关键字是union，所有成员共用同一片空间——

```
union Test{
    int a;
    float f;
};
```

写法、定义规则完全等同于结构体，只是所有成员共用一片空间，所以union变量的大小就是成员里最长的那个。

枚举是一种自定义数据结构，关键字是enum，枚举中元素间用逗号隔开。

```
enum {SUN, MON, TUE, WED, TUR, FRI, SAT};
```

枚举是批量定义常量的方法，一般不定义枚举变量（所以枚举使用时大多是匿名的），如果不赋初值，枚举的第一个元素的值为0，后面依次递增1，如果给了第一个元素值，后面会按照给的数值递增1。

※switch小括号里可以是枚举型变量。

声明：C语言中，以下行为作为声明——

```
//结构体声明
struct Test{
  //结构体内容
  int i;
};
//联合体声明
union Test{
  //联合体内容
  int i;
};
//函数声明
```

```
void test();
extern void test();
//全局变量声明
extern int i;
//变量类型名声明
typedef int s32;
```

声明时没有空间，所以无法在声明时赋值。

时间函数：

函数声明	头文件	参数返回值说明	函数作用
`time_t time(time_t *timep)`	time.h	返回时间戳，或者用参数带回时间戳	获取时间戳
`char* ctime(const time_t *timep)`	time.h	传入时间戳，将其转换为当前时间的字符串返回	拿到当前时间的字符串用于打印
`struct tm* localtime(const time_t *timep)`	time.h	传入时间戳，将其转换为一个时间结构体	获取当前时间的结构体，结构体说明见附录

时间戳（timestamp）：UTC时间从1970年1月1日0点0分0秒到当下所经过的秒数。

附录：

struct tm声明：

```
struct tm
{
    int tm_sec;   // seconds after the minute - [0, 60] including leap second
    int tm_min;   // minutes after the hour - [0, 59]
    int tm_hour;  // hours since midnight - [0, 23]
    int tm_mday;  // day of the month - [1, 31]
    int tm_mon;   // months since January - [0, 11]
    int tm_year;  // years since 1900
    int tm_wday;  // days since Sunday - [0, 6]
    int tm_yday;  // days since January 1 - [0, 365]
    int tm_isdst; // daylight savings time flag
};
```