结构

声明结构类型

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    struct date {
        int month;
        int day;
        int year;
    };

    struct date today;

    today.month = 07;
    today.day = 31;
    today.year = 2014;

    printf("Today's date is %i-%i-%i.\n",
        today.year,today.month,today.day);

    return 0;
}
```

初学者最常见的

在函数内/外?

```c
#include <stdio.h>

struct date {
    int month;
    int day;
    int year;
};

int main(int argc, char const *argv[])
{
    struct date today;

    today.month = 07;
    today.day = 31;
    today.year = 2014;

    printf("Today's date is %i-%i-%i.\n",
        today.year,today.month,today.day);

    return 0;
}
```

- 和本地变量一样，在函数内部声明的结构类型只能在函数内部使用
- 所以通常在函数外部声明结构类型这样就可以被多个函数所使用了

声明结构的形式

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

struct point p1, p2;
```

p1 和 p2 都是point
里面有x和y的值

```
struct {
    int x;
    int y;
} p1, p2;
```

p1 和 p2都是一种无名结构，里面有x和y

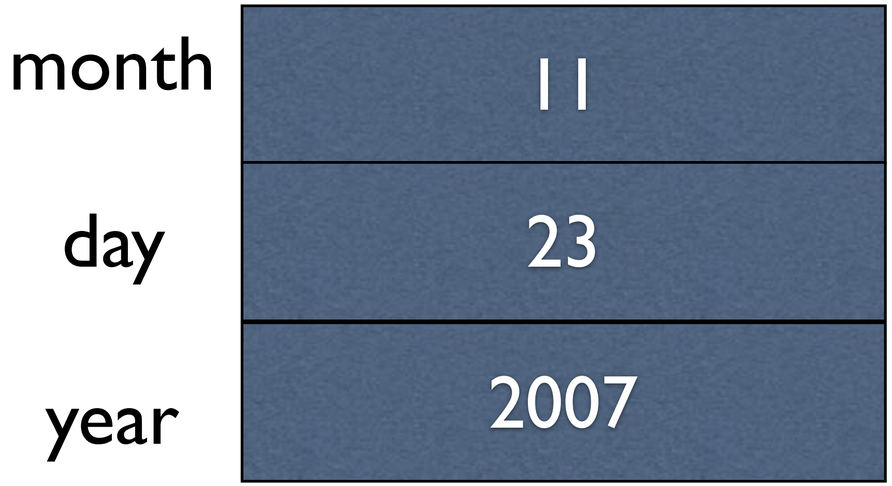
```
struct  point {
    int x;
    int y;
} p1, p2;
```

p1和p2都是point
里面有x和y的值t

对于第一和第三种形式，都声明了结构point。但是第二种形式没有声明point，只是定义了两个变量

结构变量

```
struct date today;
today.month=06;
today.day=19;
today.year=2005;
```



结构的初始化

```c
#include <stdio.h>

struct date {
    int month;
    int day;
    int year;
};

int main(int argc, char const *argv[])
{
    struct date today = {07,31,2014};
    struct date thismonth = {.month=7, .year=2014};

    printf("Today's date is %i-%i-%i.\n",
        today.year,today.month,today.day);
    printf("This month is %i-%i-%i.\n",
        thismonth.year,thismonth.month,thismonth.day);

    return 0;
}
```

结构成员

- 结构和数组有点像
- 数组用[]运算符和下标访问其成员
  - a[0] = 10;
- 结构用.运算符和名字访问其成员
  - today.day
  - student.firstName
  - p1.x
  - p1.y

结构运算

- 要访问整个结构，直接用结构变量的名字
- 对于整个结构，可以做赋值、取地址，也可以传递给函数参数
  - p1 = (struct point){5, 10}; // 相当于p1.x = 5; p1.y = 10;
  - p1 = p2; // 相当于p1.x = p2.x; p1.y = p2.y;

数组无法做这两种运算！

复合字面量

- today = (struct date) {9,25,2004};
- today = (struct date)
  {.month=9, .day=25, .year=2004};

结构指针

- 和数组不同，结构变量的名字并不是结构变量的地址，必须使用&运算符
- struct date *pDate = &today;

结构与函数

结构作为函数参数

```
int numberOfDays(struct date d)
```

- 整个结构可以作为参数的值传入函数
- 这时候是在函数内新建一个结构变量，并复制调用者的结构的值
- 也可以返回一个结构
- 这与数组完全不同

输入结构

- 没有直接的方式可以一次scanf一个结构
- 如果我们打算写一个函数来读入结构
  - —>
- 但是读入的结构如何送回来呢?
- 记住C在函数调用时是传值的
  - 所以函数中的p与main中的y是不同的
  - 在函数读入了p的数值之后，没有任何东西回到main，所以y还是 {0, 0}

```
#include <stdio.h>

struct point  {
    int x;
    int y;  };

void getStruct(struct point);
void output(struct point);
void main( )    {
    struct point y = {0, 0};
    getStruct(y);
    output(y);  }

void getStruct(struct point p)   {
    scanf("%d", &p.x);
    scanf("%d", &p.y);
    printf("%d, %d", p.x, p.y);  }

void output(struct point p)   {
    printf("%d, %d", p.x, p.y);  }
```

解决的方案

- 之前的方案，把一个结构传入了函数，然后在函数中操作，但是没有返回回去
  - 问题在于传入函数的是外面那个结构的克隆体，而不是指针
    - 传入结构和传入数组是不同的
- 在这个输入函数中，完全可以创建一个临时的结构变量，然后把这个结构返回给调用者

```
void main( )
{
      struct point y = {0, 0};
      y = inputPoint( );
      output(y);
}
```

```
struct point inputPoint( )
{
      struct point temp;
      scanf("%d", &temp.x);
      scanf("%d", &temp.y);
      return temp;
}
```

也可以把y的地址传给函数，函数的参数类型是指向一个结构的指针。
不过那样的话，访问结构的成员的方式需要做出调整。

结构指针作为参数

- K & R 说过 (p. 131)
  - “If a large structure is to be passed to a function, it is generally more efficient to pass a pointer than to copy the whole structure”

指向结构的指针

```
struct date {
    int month;
    int day;
    int year;
} myday;

struct date *p = &myday;

(*p).month = 12;
p->month = 12;
```

- 用->表示指针所指的结构变量中的成员

结构指针参数

```
void main( )
{
    struct point y = {0, 0};
    inputPoint(&y);
    output(y);
}
```

```
struct point* inputPoint(struct point *p)
{
        scanf("%d", &(p->x));
        scanf("%d", &(p->y));
        return p;
}
```

- 好处是传入传出只是一个指针的大小
- 如果需要保护传入的结构不被函数修改
  - const struct point *p
- 返回传入的指针是一种套路

结构中的结构

结构数组

```
struct date dates[100];
struct date dates[] = {
    {4,5,2005},{2,4,2005}};
```

结构中的结构

```
struct dateAndTime {
    struct date sdate;
    struct time stime;
};
```

嵌套的结构

```
struct point {
    int x;
    int y;
 };
struct rectangle {
    struct point pt1;
    struct point pt2;
};
```

如果有变量

```
  struct rectangle r;
```

就可以有：

r.pt1.x、r.pt1.y,

r.pt2.x 和 r.pt2.y

如果有变量定义：

```
struct rectangle r, *rp;
rp = &r;
```

那么下面的四种形式是等价的：

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```

但是没有rp->pt1->x （因为pt1不是指针）

结构中的结构的数组

```
#include <stdio.h>

struct point{
    int x;
    int y;
};

struct rectangle {
    struct point p1;
    struct point p2;
};

void printRect(struct rectangle r)
{
    printf("<%d, %d> to <%d, %d>\n", r.p1.x, r.p1.y, r.p2.x, r.p2.y);
}

int main(int argc, char const *argv[])
{
    int i;
    struct rectangle rects[ ] = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}; // 2 rectangles
    for(i=0;i<2;i++) printRect(rects[i]);
}
```