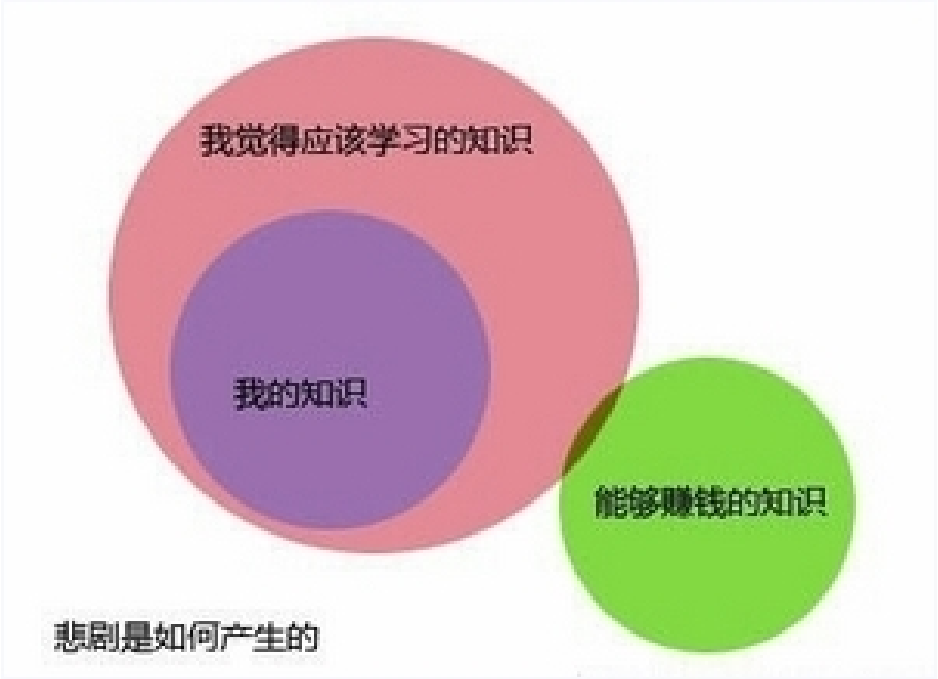
第8章 指针（4）

复习回顾

- 上次课的内容：
  - ◆指针与数组
  - ◆指向数组的指针作函数参数
  - ◆通过指针引用字符串
  - ◆动态内存管理及其函数
    - ●malloc ……
  - ◆我们所学所知的不应该仅局限于课堂和课本



malloc函数应用实例

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int count, *array;
    array = (int *) malloc(10*sizeof(int));
    if (array == NULL)
    {
        printf("不能成功分配存储空间！");
        return 0;
    }
    for (count=0; count<10; count++)
        array[count] = count;
    for (count=0; count<10; count++)
        printf("%d ", array[count]);
    printf("\n");
    free(array);
    return 0;
}
```

```
0 1 2 3 4 5 6 7 8 9
Press any key to continue
```

calloc函数

2．calloc函数

- 其函数原型为

```
void *calloc(unsigned n,unsigned size);
```

- 其作用是在内存的动态存储区中分配n个长度为size的连续空间，这个空间一般比较大，足以保存一个数组。

calloc函数

- 用calloc函数可以为一维数组开辟动态存储空间，n为数组元素个数，每个元素长度为size。这就是动态数组。函数返回指向所分配域的起始位置的指针；如果分配不成功，返回NULL。
- 比如：

```
p = calloc(50, 4);
```

表示开辟50×4个字节的临时分配域，把起始地址赋给指针变量p

calloc函数应用实例

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i, *array;
    array = (int *) calloc(10, sizeof(int));
    if (array == NULL)
    {
        printf("不能成功分配存储空间！");
        return 0;
    }
    printf("%d\n", array[9]);
    for (i=0; i<10; i++)
        array[i] = i;
    printf("%d\n", array[9]);
    free(array);
    return 0;
}
```

```
0
9
Press any key to continue
```

malloc与calloc的区别

- malloc()和calloc()的主要区别是前者不能初始化所分配的内存空间，而后者能。
  - 由malloc()函数分配的内存空间可能遗留各种各样的数据。
  - calloc() 函数会将所分配的内存空间中的每一位都初始化为零

free函数

3．free函数

- 其函数原型为

```
void free(void *p);
```

- 其作用是**释放指针变量p所指向的动态空间**，使这部分空间能重新被其他变量使用。p应是最近一次调用calloc或malloc函数时得到的函数返回值。

free函数

- 调用free函数的一般形式

```
free(p);
```

- 释放指针变量p所指向的已分配的动态空间
- free函数无返回值

free与内存分配函数的对应关系

- 注意：一夫一妻制！
  - 一次分配（malloc/calloc），一次释放
- 如果：分配两次，free一次
  - 内存泄露！因为分配的内存没有释放。
- 如果：分配一次，free两次
  - 系统报错！因为尝试释放没有分配的内存。

释放内存之后应该做什么

- 如果使用free函数将指针变量p指向的内存释放，那我们需要把p的值变为NULL：p＝NULL;
- 例如

```
char * p = (char*) malloc(100);
strcpy(p, "Hello");
free(p);    //p所指的内存被释放，但是p所指的地址仍然不变
…
if (NULL != p) //没起到防错作用
{
    strcpy(p, “world”); //出错
}
```

杜绝“野指针”

- 理解野指针的“野”
  - 野孩子：没人要，没人管，调皮捣蛋
  - 野狗：没主人，没有链子拴着，喜欢四处咬人
  - 野指针：指向一块不可用内存的指针
- 如何杜绝野指针p？
  - 很简单，却很容易被忽略：p=NULL;

realloc函数

4. realloc函数

- 其函数原型为

```
void *realloc(void *p,unsigned int size);
```

- 如果已经通过malloc函数或calloc函数获得了动态空间，想改变其大小，可以用recalloc函数重新分配。

realloc函数

- 用realloc函数将p所指向的动态空间的大小改变为size。p的值不变。**如果重分配不成功，返回NULL。**
- 如

```
realloc(p,50);
```

将p所指向的已分配的动态空间改为50字节

void指针类型

- C99允许定义一个基类型为void的指针变量（即void*型变量）
- 不要把“指向void类型”理解为能指向“任何类型”，而应理解为“指向空类型”或“不指向确定类型”

```
int a=3;                          //定义a为整型变量
int *p1=&a                        //p1指向int型变量
char *p2;                         //p2指向char型变量
void * p3;                        //p3为无类型指针变量
p3=(void *)p1;                    //将p1的值转为void*类型，赋值给p3
p2=(char*)p3;                     //将p2的值转为char*类型，赋值给p2
printf(“%d”, *p1);                //合法，输出a的值
p3=&a; printf(“%d”,*p3);          //错误，p3是无类型的
```

动态内存分配实例

例：建立动态数组，输入5个学生的成绩，另外用一个函放数检查其中有无低于60分的，输出不合格的成绩。

动态内存分配实例

- 解题思路：
  - 用malloc函数开辟一个动态自由区域，用来存5个学生的成绩，会得到这个动态域第一个字节的地址，它的基类型是void型。
  - 用一个基类型为int的指针变量p1来指向动态数组，但必须先把malloc函数返回的void指针转换为整型指针，然后赋给p1
  - 把p1传递给函数check，使check可以访问，判断并输出动态数组各元素的值。

- 源代码：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    void check(int *);
    int *p1,i;
    p1=(int *)malloc(5*sizeof(int));
    for (i=0;i<5;i++)
        scanf("%d",p1+i);
    check(p1);
    return 0;
}
```

```
void check(int *p)
{
    int i;
    printf("They are fail:");
    for (i=0;i<5;i++)
        if (p[i]<60)
            printf("%d ",p[i]);
    printf("\n");
}
```

```
67 98 59 78 57
They are fail:59 57
```

sizeof运算符

- 单目运算符，运算对象可以是一个表达式或者括号内的类型名
  - ◆用于数据类型的一般形式
    - ●sizeof(类型名)
    - ●如 sizeof(int); sizeof(double);
  - ◆用于表达式的一般形式
    - ●sizeof(表达式) 或 sizeof 表达式
    - ●如 sizeof(n+1); sizeof n;
    - ●注意：sizeof n+1; 等价于 sizeof(n)+1 ！

经常被冤枉的关键字sizeof

- 由于后面常跟括号，常年被误以为是函数
- 可借助编译器确定它的身份
  - 设有int i=0；下列哪个表达式是错误的？

    a. sizeof(int)

    b. sizeof(i)

    c. sizeof int　　错！

    d. sizeof i

sizeof的结果

- 若运算对象是基本类型或基本类型变量，其结果等于该类型变量规定占用的字节数
- 若运算对象是指针，结果依赖于系统，32位计算机指针字节数为4，64位系统则字节数为8
- 若运算对象是具有数组类型（或除函数形式参数外的数组名），其结果是数组的总字节数
- 若运算对象是函数中的数组形参，结果是其指针的大小

用sizeof辨析数组形参真面目

- 下列程序运行结果是什么？

```
#include <stdio.h>

void fun(int b[100])
{
    printf("sizeof(b)=%d\n", sizeof(b));
}

int main()
{
    int b[100];
    fun(b);
    return 0;
}
```

```
sizeof(b)=4
```

常见的内存错误之一

- 误把变量名当地址作参数使用

```
int i, n;
char c[10];
char str[10][100];

scanf("%d", n);          //错！

for (i=0; i<10; i++)
{
    scanf("%s", c[i]);   //错！
    scanf("%s", c);      //正确
    scanf("%s", str[i]); //正确
}
```

常见的内存错误之二

- 为指针分配的内存太小

```
char *p1 = "abcdefg";
//错！
char *p2 = (char*)malloc(sizeof(char)*strlen(p1));
strcpy(p2, p1);


char *p1 = "abcdefg";
//正确！
char *p2 = (char*)malloc(sizeof(char)*(strlen(p1)+1));
strcpy(p2, p1);
```

常见的内存错误之三

- 内存分配成功但并未初始化

```
int i, n, *p;

scanf("%d", &n);
p = (int*)malloc(n*sizeof(int));
if (p != NULL)
{
    for (i=0; i<n; i++)
    {
        printf("%d ", p[i]); //未初始化，输出随机值
    }
    printf("\n");
}
```

常见的内存错误之四

➢内存越界

```
int i, n, *p;

scanf("%d", &n);

p = (int*)malloc(n*sizeof(int));

for (i=0; i<=n; i++) //发生越界
{
    p[i] = i;
}
```

提示：for循环的循环变量尽量采用半开半闭的区间，如果不是特殊情况，循环变量尽量从0开始。

常见的内存错误之五

➢内存泄露

- ◆由malloc系列函数分配的内存，没有及时用free释放
- ◆几乎很难避免，无论老手还是新手，甚至windows，office这样的商用软件都有这种现象
- ◆一般情况下似乎不严重，重启软件就可以解决
  - ●如果是汽车制动系统的控制软件呢？
  - ●如果是心脏起搏器的控制软件呢？

用一张图揭示内存分配的秘密

- **栈**：局部变量
- **堆**：动态内存管理
- **静态区**：全局变量
- **常量区**：常量（只读）
- **代码区**：指令（只读）

高地址

容量较小

低地址

有关指针的小结

1.首先要准确地弄清楚指针的含义。

- ◆指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替
- ◆变量的指针就是变量的地址，指针变量就是地址类型的变量
- ◆务必区别 指针 和 指针变量：
  - ●指针就是地址本身
  - ●而指针变量是用来存放地址的变量。

有关指针的小结

2. 什么叫“指向”？地址就意味着指向

- 因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。
- 但应注意：只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

有关指针的小结

3.void *指针是一种特殊的指针

- void类型指针不指向任何类型的数据
- 如果需要用此地址指向某类型的数据，应先对地址进行**类型转换**
- 可以在程序中进行显式的类型转换，也可以由编译系统自动进行隐式转换。无论用哪种转换，必须先了解要进行的类型转换

有关指针的小结

4. 要深入掌握在对数组的操作中怎样正确地使用指针，搞清楚指针的指向。一维数组名代表数组首元素的地址

```
int *p,a[10];  p=a;
```

- p是指向int类型的指针变量，p只能指向数组中的元素，而不是指向整个数组。在进行赋值时一定要先确定赋值号两侧的类型是否相同，是否允许赋值。
- 对“p=a;”，准确地说应该是：p指向a数组的首元素

有关指针的小结

5.指针运算

(1)指针变量加（减）一个整数

例如：p++,p--,p+i,p-i,p+=i,p-=i等均是指针变量加（减）一个整数。

- 将该指针变量的原值(是一个地址)和它指向的变量所占用的存储单元的字节数（或乘以i的值）相加（减）。

有关指针的小结

5.指针运算

(2)指针变量赋值

- 将一个变量地址赋给一个指针变量
- 不应把一个整数赋给指针变量
  - 唯一的例外是将指针变量赋值为0

有关指针的小结

5.指针运算

（3）两个指针变量可以相减

- 如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数

有关指针的小结

5.指针运算

(4) 两个指针变量比较

- 若两个指针指向同一个数组的元素，则可以进行比较
- 指向前面的元素的指针变量“小于”指向后面元素的指针变量
- 如果p1和p2不指向同一数组则比较无意义

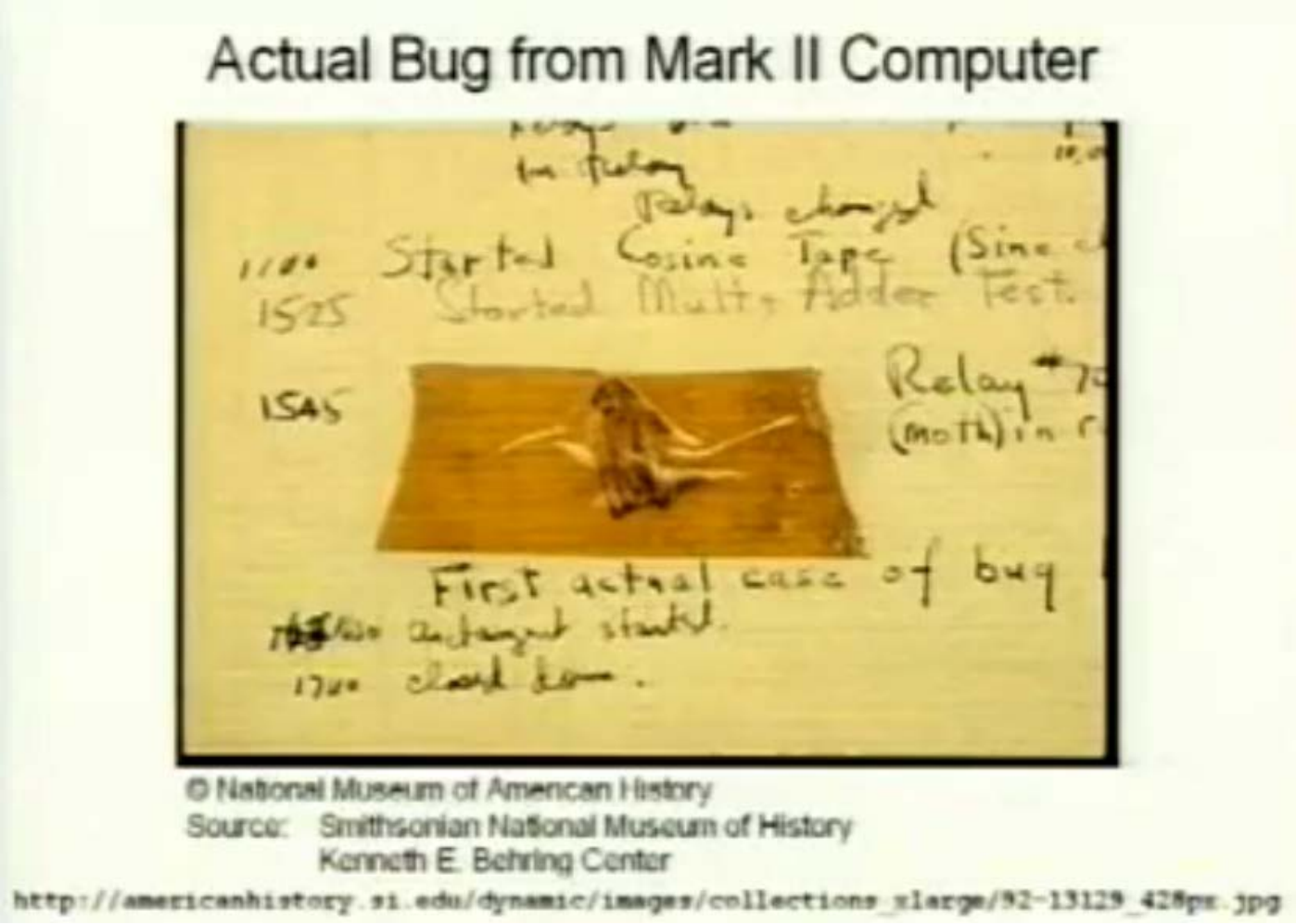
一段编译通过却运行错误的程序

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.     int i, n, fac;
6.     scanf("%f, &n);
7.     if (n==0);
8.     {
9.         printf("Invalid n!\n");
10.        return 0;
11.    }
12.
13.    for (i=1; i<n; i++)
14.    {
15.        fac = fac*i;
16.        printf("%d!=%d\n",n,fac);
17.    }
18.    return 0;
19.}
```

什么叫bug

- 程序中隐藏着的缺陷或问题
  - 编译器通常无法帮我们发现它们
  - 写程序痛苦的主要根源
  - 脑筋急转弯
    - 问题：程序员最不喜欢康熙的哪个儿子？
    - 解答：八阿哥！

为什么叫Debug


Actual Bug from Mark II Computer
1100 Started Cosine Tape
1525 Started Multy Adder Test.
1545
Relay
(moth)
First actual case of bug
1700 closed down.
© National Museum of American History
Source: Smithsonian National Museum of History
Kenneth E. Behring Center
http://americanhistory.si.edu/dynamic/images/collections_xlarge/92-13129_428px.jpg

从修理电视看调试的重要性

新手

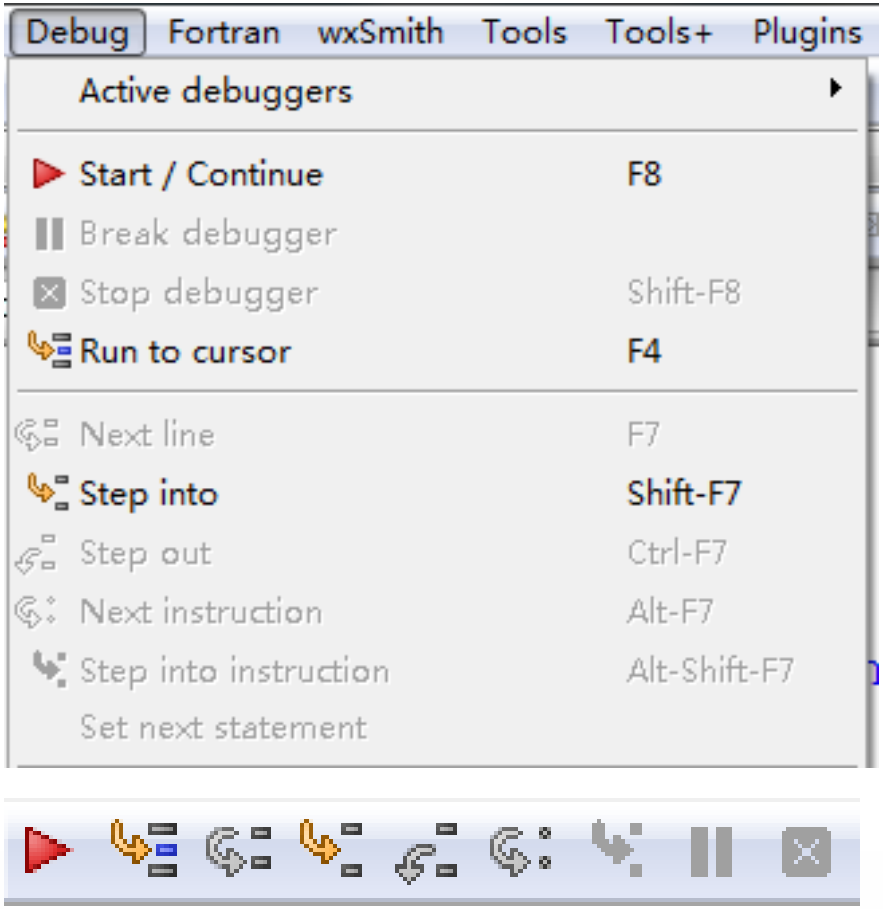
老手

调试（debug）：除错的法宝

- 什么时候需要调试？
  - 编译链接正确，而运行结果错误的时候
- 调试的基本操作
  - 跟踪执行：用来判断分支或循环的执行路径是否与期望的相符。
  - 设置断点：用来在怀疑有错的某条语句之前或之后设置断点，使程序执行到断点处暂停下来，再进一步调试；
  - 观察数据：观察某些变量的状态，看是否与预料的结果相符。

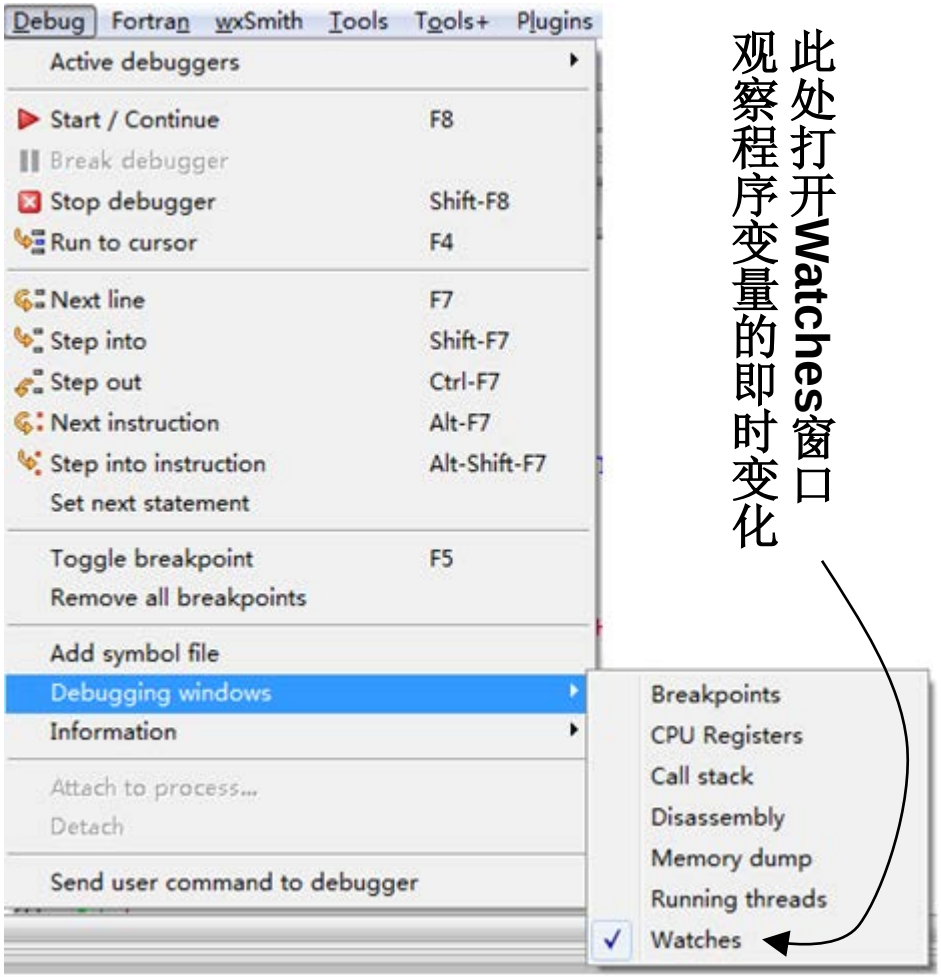
在CodeBlocks上启动调试

- 如右下图，选择该项将弹出子菜单，其中含有用于启动调试器运行的几个选项。
  - **Start/Continue(F8)**选项用于从当前语句开始执行程序，直到遇到断点或遇到程序结束；
  - **Step Into(Shift-F7)**选项开始单步执行程序，并在遇到函数调用时进入函数内部再从头单步执行
  - **Run to Cursor(F4)**选项使程序运行到当前鼠标光标所在行时暂停其执行(注意，使用该选项前，要先将鼠标光标设置到某一个你希望暂停的程序行处)



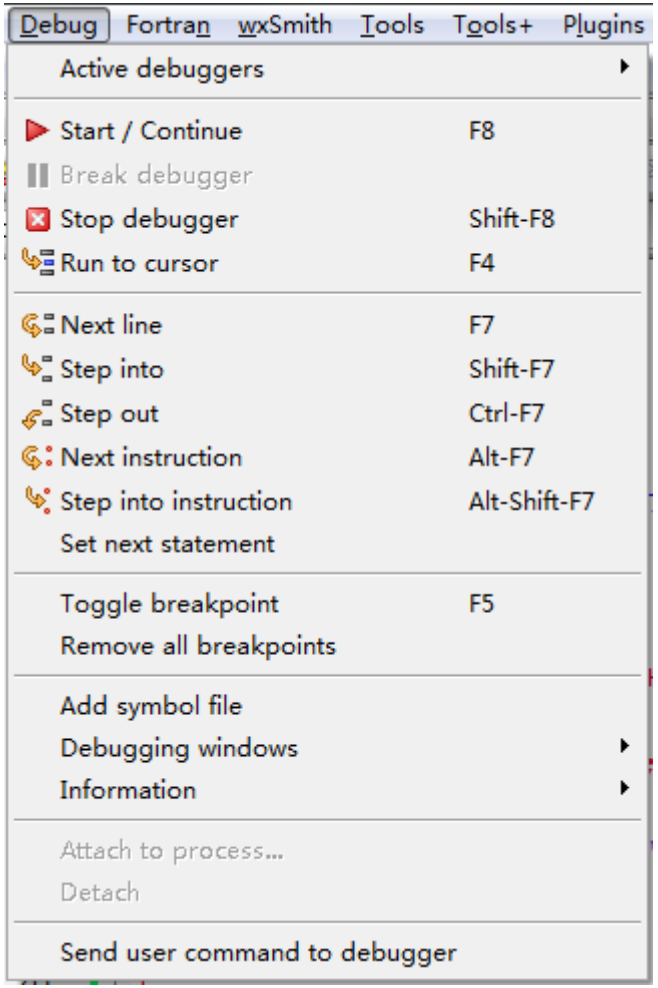
C::B中调试相关的操作（1）

- **Start/Continue**：快捷键F8。从当前语句启动继续运行程序，直到遇到断点或遇到程序结束而停止（注意，当程序运行到诸如scanf之类的等待输入的语句时，需先在控制台窗口输入，此时Start/Continue按钮为灰色无效状态）
- **Stop Debugging**：快捷键Shift+F8。中断当前的调试过程并返回正常的编辑状态(注意，系统将自动关闭调试器)。



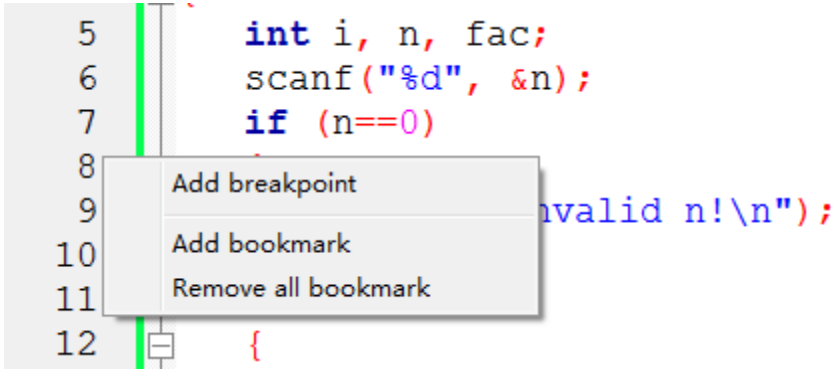
C::B中调试相关的操作（2）

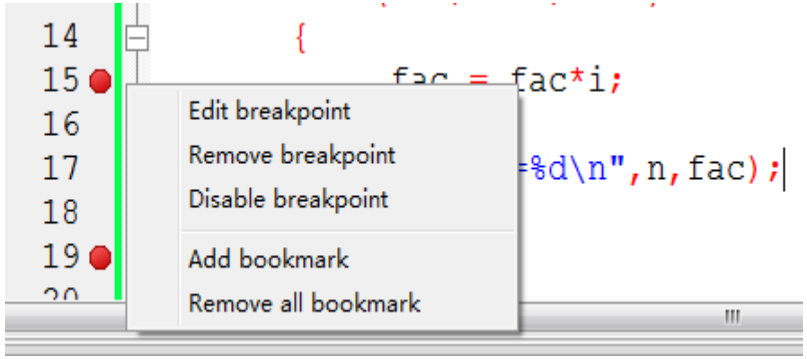
- **Next line**：快捷键F7。单步执行程序，但当执行到函数调用语句时，不进入那一函数内部，而是一步直接执行完该函数后，接着再执行函数调用语句后面的语句
- **Step Into**：快捷键Shift+F7。单步执行程序，并在遇到函数调用语句时，进入那一函数内部，并从头单步执行。
- **Step Out**：快捷键Control+F7。与Step Into配合使用，当执行进入到函数内部，单步执行若干步之后，若发现不再需要进行单步调试的话，通过该选项可以从函数内部返回(到函数调用语句的下一语句处停止)。
- **Run to Cursor**：快捷键F4。使程序运行到当前鼠标光标所在行时暂停其执行(注意，使用该选项前，要先将鼠标光标设置到某一个你希望暂停的程序行处)。事实上，相当于设置了一个临时断点。



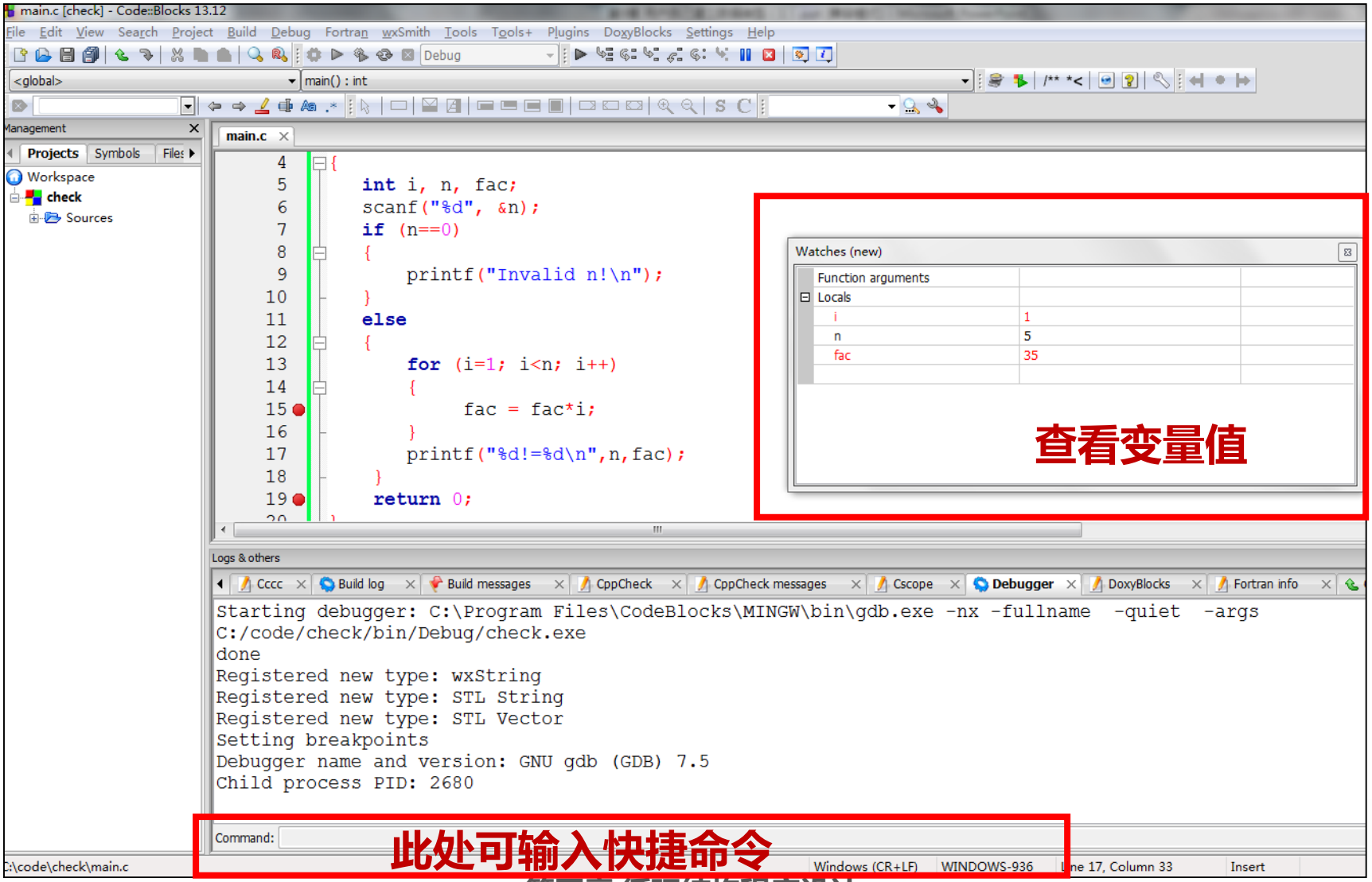
C::B中调试相关的操作（3）

- **Toggle Breakpoint**：
  - 快捷键F5。其功能是在光标所在的当前行设置或取消固定断点;程序行前有一个圆形的红点标志，表示已经该行设置了固定断点
  - 亦可在代码窗口行号旁边单击鼠标右键来添加/删除或编辑断点





Code::Blocks调试界面



调试的基本技巧提示

- 一开始情况未明，先在main函数第一个语句放置一个断点，用Start（快捷键F8）进入调试模式然后用Next Line单步执行，观察执行路径与变量变化状态。
- 确定可能出错的语句后，可用Run to Cursor（快捷键F4），或采用断点结合Continue（快捷键F8）的方式跳过已经确定为正确的语句，直接执行到可能出错的语句之前，再用Next Line单步执行详细调试。
- 需调试被调用的自定义函数时，可使用Step Into进入自定义函数详细调试，在函数内部，可使用Step Out迅速从本函数中返回
- 此处省略很多很多……
- 更多的技巧和经验来自于自己经过调试实践得到的总结和体会。

编程不仅仅是玩玩数字


fcomplex Csqrt(fcomplex z)
fcomplex c;
float w;
if ((z.r == 0.0) && (z.i == 0.0))
c.r=0.0;
c.i=0.0;

回顾我们学过的数据类型

目前学过的数组类型
- 基本类型（不可分解）
  - 整型
    - 短整型（short）
    - 整 型（int）
    - 长整型（long）
    - 无符号整型(unsigned int)
  - 浮点型（实型）
    - 单精度型（float）
    - 双精度型（double）
  - 字符类型（char）
  - 空值型(void)
- 数组类型（[]）
- 指针类型（地址，*）

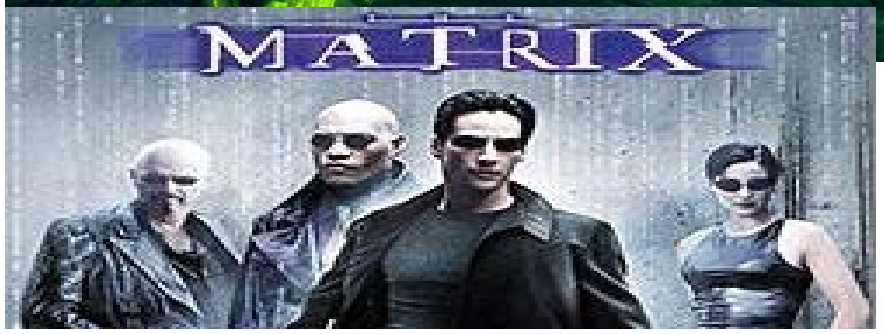
用这些类型怎么描述现实生活中的事物？

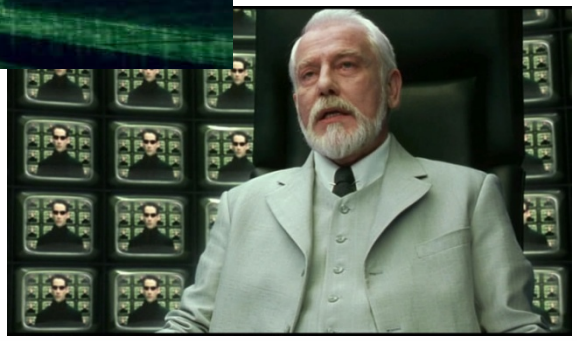
比如：茶？

爱喝茶的C语言大师


struct tea quila =
{"tealeaves", "milk",
"sugar", "water", "tequila"};
Herald
Tribune

程序员的最高境界

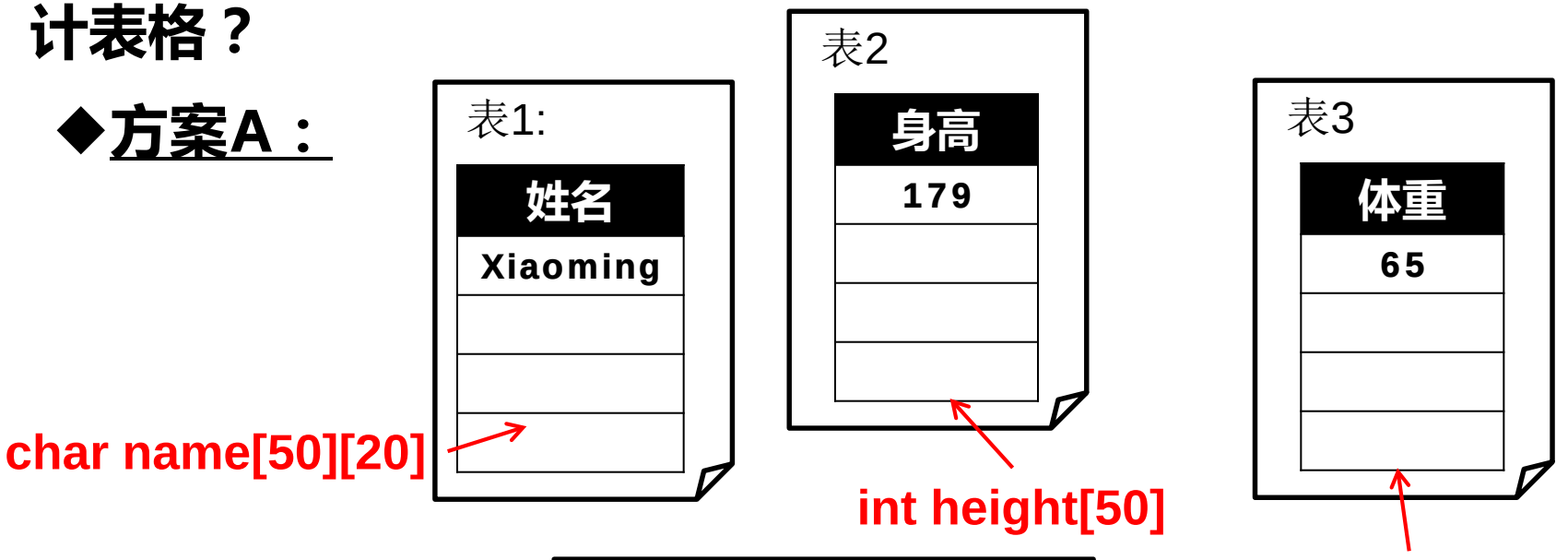




现实例子：设计体检卡片

- 假设现在要汇总班上50名同学的体检信息，你会如何设计表格？

◆方案A：



◆方案B：

√

什么是结构体

- 现实生活中，一个事物往往具有多个属性
  - 一辆汽车：品牌（字符串）、车型（字符串）、马力（整型）等
  - 一名学生：学号（字符串）、姓名（字符串）、性别（字符型）、年龄（整型）、成绩（整型或浮点型）等
- 结构体是由一系列具有相同类型或不同类型的数据构成的数据集合，允许用户根据自己需要建立数据类型，简称结构。

结构体之《水浒传》

```
struct hero
{
   int number;
   char sex;
   char star_name[20];
   char name[20];
   char nickname[20];
   char position[20];
   char weapon[20];
};
```

Songjiang
1
M
天魁星
宋江
及时雨
总头领
无

Luda
13
M
天孤星
鲁智深
花和尚
步军头领
水磨禅杖

Husanniang
59
F
地慧星
扈三娘
一丈青
马军头领
日月双刀

结构体类型的一般形式

➢声明一个结构体类型的一般形式为：

```
struct  结构体名
{
        成员表列
};
```

类型名 成员名;

```
struct 结构体类型名
{
        数据类型  成员1;
        数据类型  成员2;
        ……
        数据类型  成员N;
};
```

结构体类型实例

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

- 由程序设计者指定了一个结构体类型 struct Student
- 它包括num,name,sex,age,score,addr等不同类型的成员
- 声明结构体类型时别漏掉右大括号后面的分号！

关于结构体类型的说明

(1)结构体类型并非只有一种，而是可以设计出许多种结构体类型。

例如

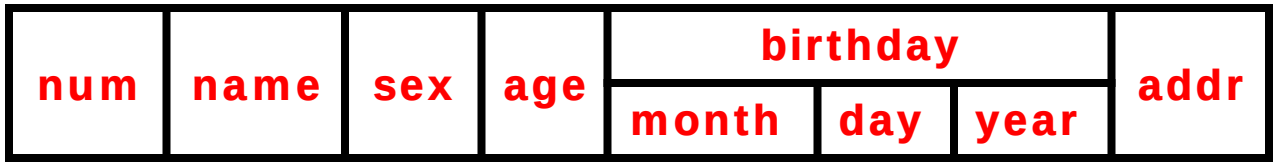
```
struct Teacher
struct Worker
struct Date
```

等结构体类型，各自包含不同的成员

关于结构体类型的说明

(2) 成员可以属于另一个结构体类型。

```
struct Date
{
    int month;  int day;  int year;
};
struct Stu
{
    int num;
    char name[20];
    char sex;
    int age;
    struct Date birthday;
    char addr[30];
};
```



为什么要定义结构体类型变量

- 建立了一个结构体类型，相当于创建了一个模型，并没有定义变量，其中并无具体数据，系统对之也不分配存储单元。
  - “相当于设计好了图纸，但并未建成具体的房屋。”
- 为了能在程序中使用结构体类型的数据，应当定义结构体类型的变量，并在其中存放具体的数据。

定义结构体类型变量的方式

1. 先声明结构体类型，再定义该类型变量

- 刚才已声明结构体类型struct Student，可以用它来定义变量

```
struct Student  student1,student2;
```

结构体类型名

结构体变量名

定义结构体类型变量的方式

1. 先声明结构体类型，再定义该类型变量

- 刚才已声明结构体类型struct Student，可以用它来定义变量

```
struct Student  student1,student2;
```

student1

?	?	?	?	?	?
num	name	sex	age	score	addr

student2

?	?	?	?	?	?

定义结构体类型变量的方式

2.在声明类型的同时定义变量

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} student1, student2;
```

定义结构体类型变量的方式

3. 不指定类型名而直接定义结构体类型变量

➢其一般形式为：

```
struct
{
    成员表列
}变量名表列;
```

```
struct
{
    int number;
    char sex;
    char star_name[20];
    char name[20];
    char nickname[20];
    char position[20];
    char weapon[20];
}wusong, sunerniang;
```

指定了一个无名的结构体类型 。

关于结构体类型变量的说明

(1)结构体类型与结构体变量是不同的概念，不要混同。

- 只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。
- 在编译时，对类型是不分配空间的，只对变量分配空间。

关于结构体类型变量的说明

(2) 结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象。

例如 struct point {int x; int y; };

和 int x;

(3) 对结构体变量中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。

结构体概念示例

- 现实生活中，一个事物通常具有多个相同或不同种类的属性
  - 比如一个人具有
    - 姓名：char name[20];
    - 身高：int height;
    - 体重：int weight;
    - ……

中文名	黄晓明
外文名	Huang Xiaoming
身　高	179cm
体　重	65kg

如何定义结构体类型

关键字 → `struct` 类型名称 → `Person`

```
struct Person
{
    char name[20];
    int height;
    int weight;
};
```

起始花括号 → `{`

结束花括号 → `}`

结构体成员 → `char name[20];` `int height;` `int weight;`

末尾必不可少的分号 → `;`

抽象成

中文名
外文名
身　高
体　重

共同属性

如何定义结构体变量示例

```
struct Person
{
     char name[20];
    int height;
    int weight;
};
```

这是类型定义

方式1：先定义类型，后定义变量

```
struct Person MrRight;
```

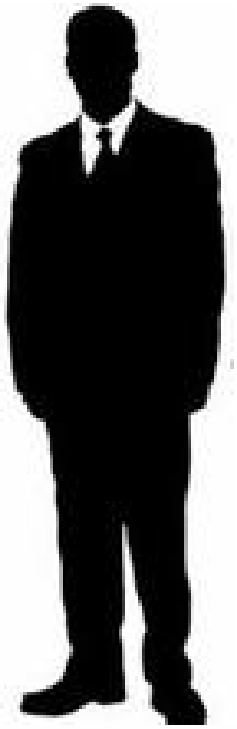
这是类型名　　这是变量名

方式2：同时定义类型和变量

```
struct Person
{
     char name[20];
     int height;
    int weight;
} MrRight;
```

此时类型名可省略

这是类型定义　　这是变量名



作业 2017/12/11

- 按下列要求编写程序，提交手写源代码

1. 定义一个表示“时刻”的结构体变量（包括年、月、日、时、分、秒），实现该变量的输入。假定该年度1月1日0点0分0秒是第一秒，计算该时刻在本年中是第几秒。