********** 注 **********

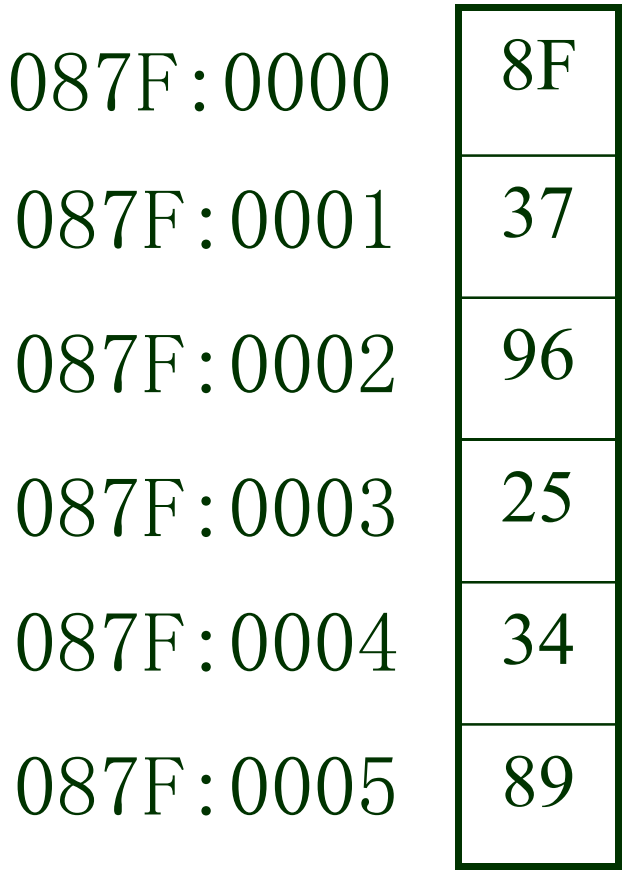
编程环境为CodeBlock

一、地 址

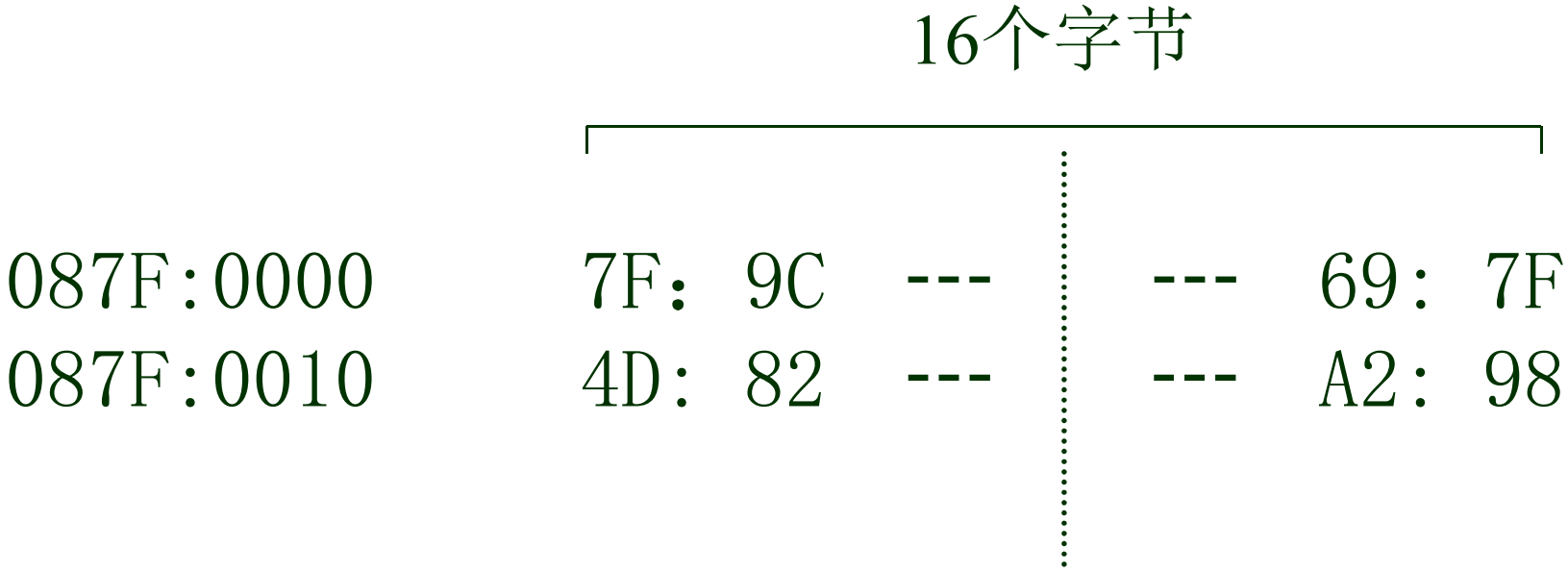
找人怎么找？

第一次上课找教室、调课（调教室）

考试地点


087F:0000
8F
087F:0001
37
087F:0002
96
087F:0003
25
087F:0004
34
087F:0005
89

内存单元的地址及其所指的内容！



二、 指　　针

1、 指针的定义

定义：指针是用以定位其它变量的地址变量

内涵：

1）指针是一个变量

2）值为地址

3）通过它，可以查找到其它变量的值

4) 其所指的变量有类型上的区别，

因此：

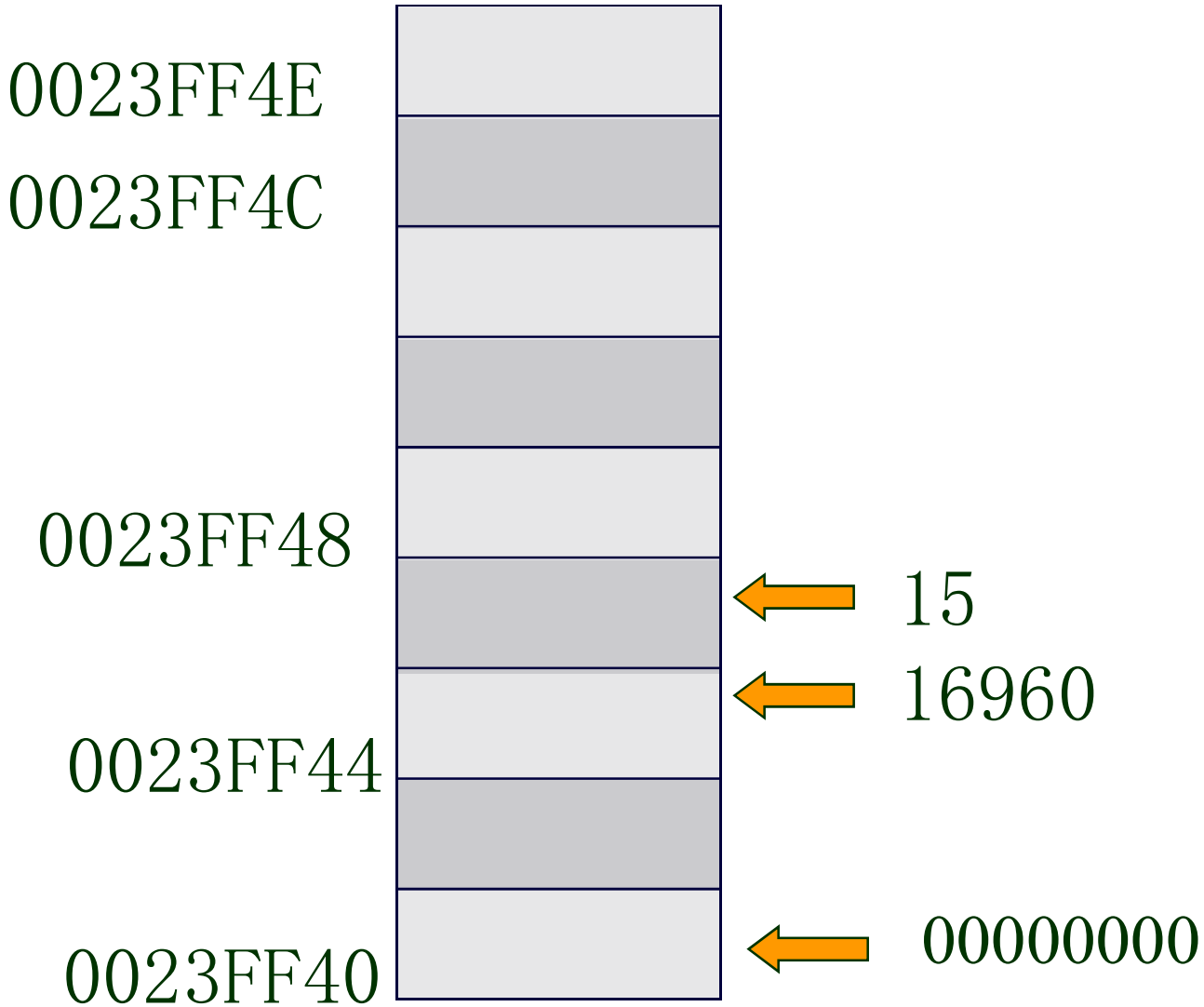
“所指变量所占内存大小”因类型不同而有所区别。

2、C中指针的表示方法及示例

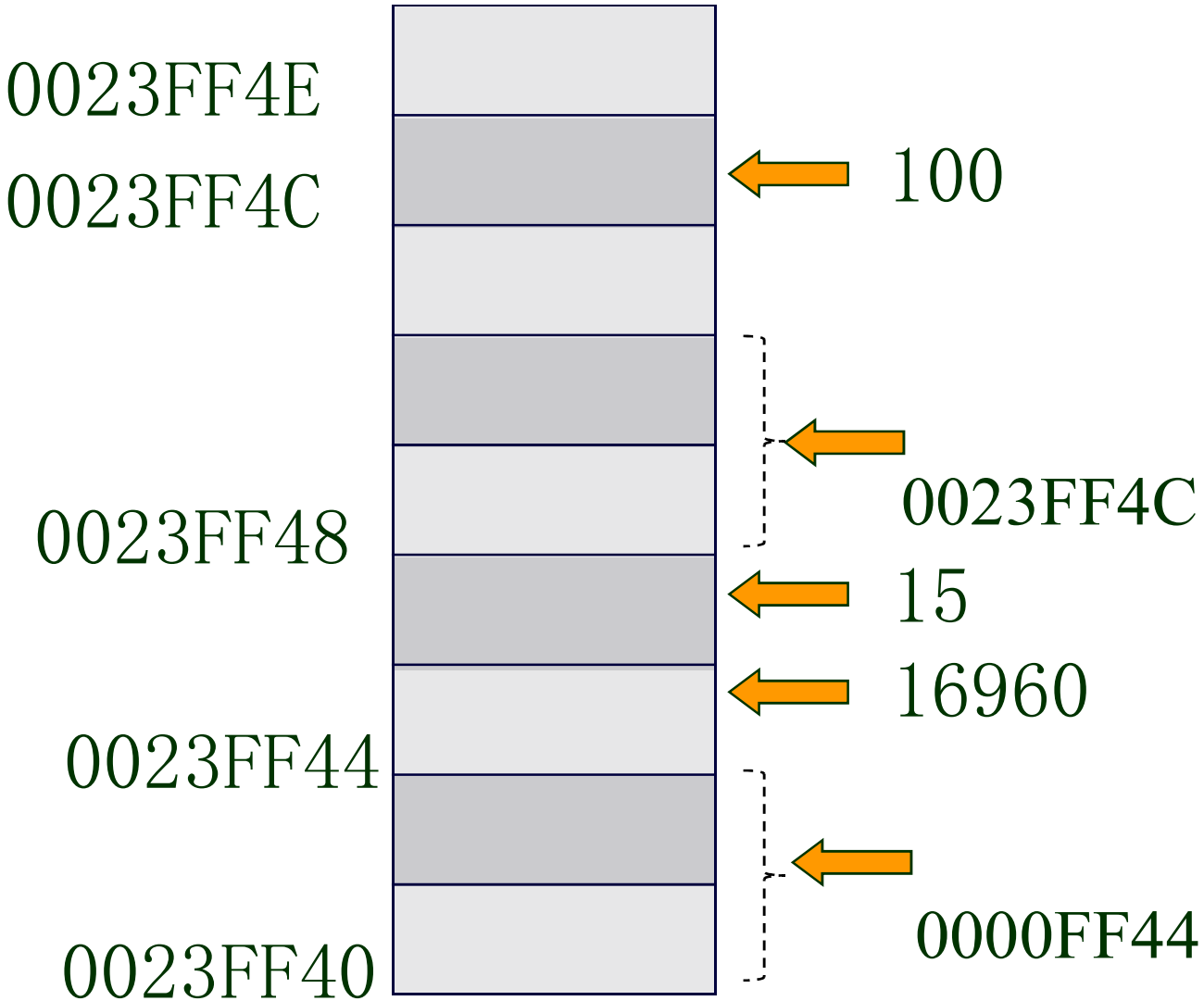
基本类型

char	**一个字节**
short int	**一般（至少）16位**
long int	**一般（至少）32位**
int	**或16位，或32位，决定于系统**
float	
double	

```
short int start, id,*a_id;
long int score =1000000;
short int *c=NULL;

printf("\n %d %d %p %ld %p",start, id,a_id,score,c);
printf("\n %p %p %p %p %p \n",
                                & start, &id,&a_id,&score,&c);

id=100;
a_id=&id;
c= &score;

printf(“整型值及其所在地址分别为%d %p \n",*a_id,a_id);
printf(“长整型值及其所在地址分别为%ld %p \n",score,&score);
printf(“长整型值中前二个字节值与地址为 %d %p \n",*c,c);
printf(“后二个字节的值与地址为%d %p \n",*(c+1),c+1);
printf("%d \n",score>>16);
```




```
short int start, id,*a_id;
long int score =1000000;
int *c=NULL;
```



id=100;

a_id=&id;

c= &score;

3、 *、&的作用

*：

1）用于定义指针变量；

2）用于取某地址处所存放的元素值 (与类型有关)

&：

1）取元素所占内存的首地址

2）其它含义

4、 指针运算

1） 赋值

2）++　　例：　p++;

3）--　　例：　p--;

4）比较　指向同一目标的指针才可以进行比较。

==；＜，＞等

增加的size与所指的类型相对应

5、 指针与数组

char name[8], *p;

p=name;

1）把name数组的首地址(第一个元素所在内存的地址)赋给p。

2）编译系统自动识别类型转换并赋值

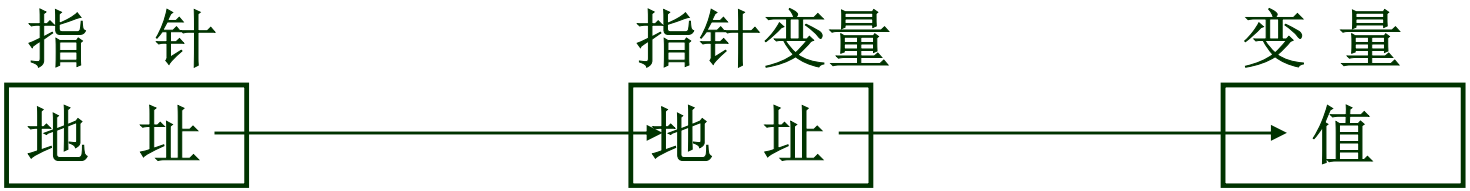
int *x[10];

1）产生一个包含10个整型指针的数组。

6、 多级间指

1、单级间址

2、多级间址




```
int x,*p,**q;
x=10;
p=&x;
q=&p;
```

三、结构

1、 结构的定义

定义：用于定义复合类型的语法元素。

内涵：

1）在内存中所实现的抽象数据类型

2）大小为相对简单的类型所占内存的和

2、 C中的结构及示例

```
struct person
{
  char name[8];
  int id;
  int math;
  int datastructure;
};
```

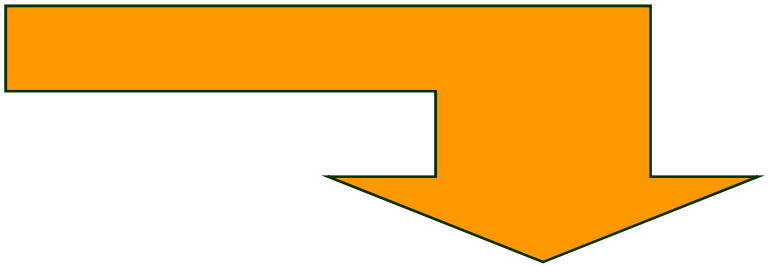
```
int main()
{
  struct person Student;
  scanf("%s", Student.name);
  scanf("%d",&Student.id);
  scanf("%d",&Student.math);
  scanf("%d",&Student.datastructure);
  printf(“ The scores of %s(%d)are %d %d \n",
         Student.name,Student.id,
         Student.math, Student.datastructure);
}
```

3、“.”运算

“.” 根据名字引用结构体中的元素

4、嵌套(多级)复合结构

```
struct person
{
 char name[8];
 int id;
 int math;
 int datastructure;
};
```



```
struct class
{
    struct person people[35];
    int classid;
    char department[8];
};
```

5、结构与指针、数组

1）结构数组

struct person people[35];

2）结构与指针

struct person *student;

3）“->”运算符

指针指向一个结构时，使用“->”进行引用。

如：

student->name

student->id

6、例

struct.txt

四、函数与数组、结构及指针

1、传值调用与引用调用

有两种方法可以把变元传给子程序：

1、传值调用(Call by value)

把变元的值复制到子程序形式参数中，所以子程序中形式参数的任何改变都不会影响调用时所使用的变量。

形式参数象其它局部变量那样，随着函数进入而建立，随着函数退出而消忘。

C语言使用传值调用来传递变元。

引用调用(Call by reference)

把变元的地址复制给形式参数。

在子程序中，这个地址用来访问调用中所使用的实际变元。

所以形式参数的变化影响调用时所使用的变量的内容。

2、C语言中的引用调用

通过把指针传给变元的方法实现引用调用。

指针可以象其它值那样向函数传递。

```
void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```

3、数组变元

当把一个数组名用作函数变元时，就把数组的地址传给了函数。

函数内部的代码可以操作、修改调用函数时所使用的数组的实际内容。

是C语言中函数传值或引用调用规则的唯一的反常

4、结构与函数

关于结构，至少有3种传递方式：分别传结构中的元素、结构整体传送、传指针，3种方式各有其优缺点。

1）
```
struct point makepoint(int x, int y){
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;}
```

2）
```
struct point addpoint(struct point p1, struct point p2){
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;}
```

3)当结构体包含的内容较大时，使用指针来传。