第九章 指针

9.1 指针的引入

1、一般把内存中的一个字节称为一个内存单元

2、为了正确地访问这些内存单元，必须为每个内存单元编上号。根据一个内存单元的编号即可准确地找到该内

存单元。**内存单元的编号也叫做地址**，通常也把这个**地址称为指针**

3、如果在程序中定义了一个变量，在对程序进行编译或运行时，系统就会给这个变量分配内存单元，并确定它的内存地址(编号)

4、**变量的地址就是变量的指针，存放变量地址的变量是指针变量**

5、**内存单元的指针**和**内存单元的内容**是两个不同的概念。可以用一个通俗的例子来说明它们之间的关系。我们到银行去存取款时，银行工作人员将根据我们的帐号去找我们的存款单，找到之后在存单上写入存款、取款的金额。在这里，帐号就是存单的指针，存款数是存单的内容。对于一个内存单元来说，单元的地址即为指针，其中存放的数据才是该单元的内容。

9.2 指针变量的定义和使用

9.2.1 指针变量定义语法

```
数据类型  *指针变量名;
注意:
1、数据类型为C语言支持的所有数据类型
2、指针变量名遵循C语言变量的命名规则
```

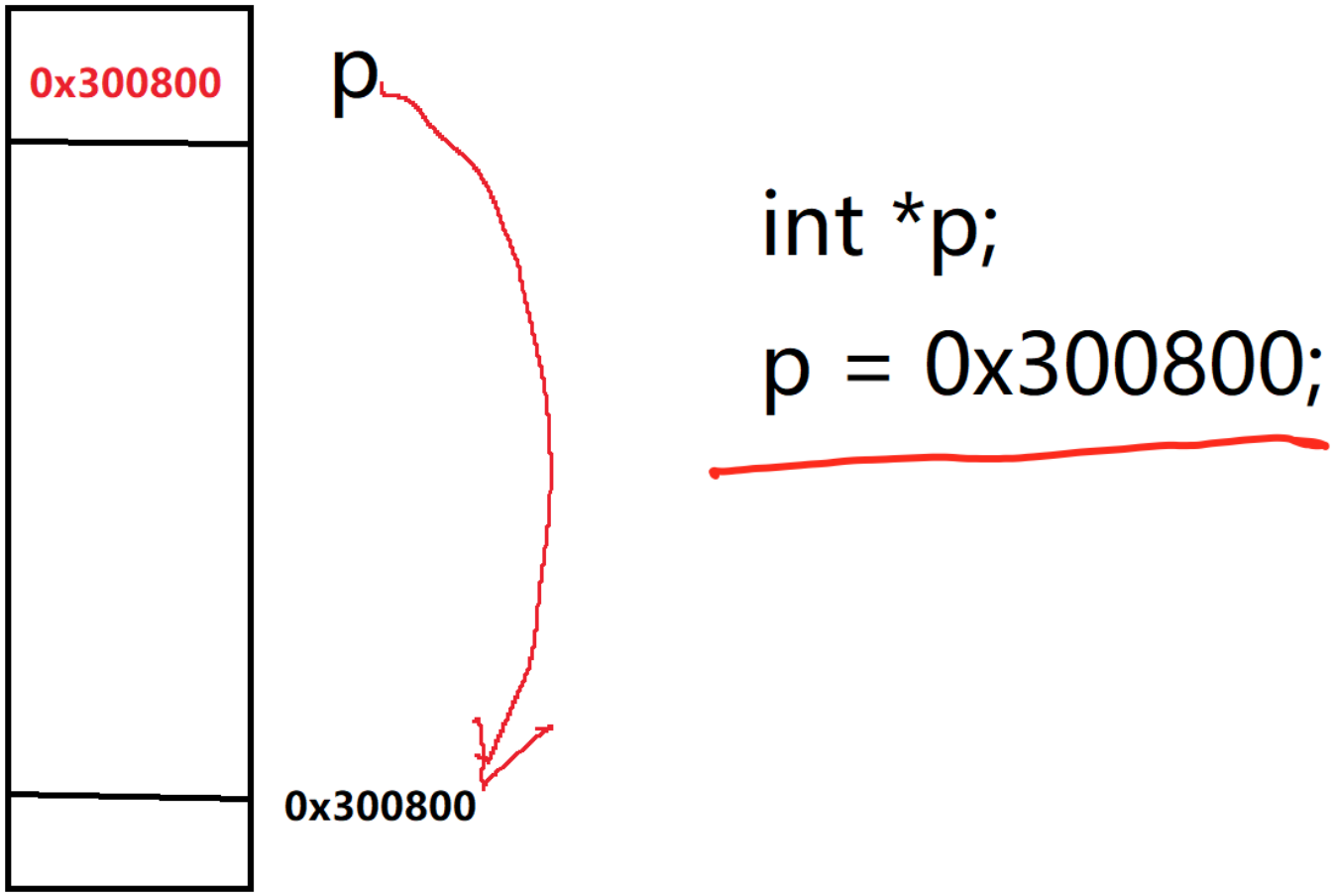
举例：

```
int  *p;  //定义了一个指针变量p，简称指针p，p是变量， int *是类型
char* p2;
```

我们也认为指针是一种数据类型。

9.2.2 指针变量的赋值

1）指针变量的值代表这个指针指向了以这个值为首地址的那块内存空间



2）指针变量赋值为其他变量的地址

```
指针变量 = &变量名;
&:取地址运算符

int a = 10;
int *p = &a;
```

3）指针变量操作指向的内存空间

可以通过指针变量访问和修改所指向的内存空间中的内容

```
*：指针运算符（或称“间接访问” 运算）。
int a = 10;
int *p;
p = &a;
printf("*p: %d\n", *p);

*p = 100;
printf("*p: %d\n", *p);
```

4）指针变量同普通变量一样，定义后如果不进行初始化指针变量的值是不确定的

```
int main() {
    int *p;
    printf("p: %p\n", p);
    return 0;
}
```

```
p: 0x7ffee06f6b18
```

5）野指针

因为指针变量的值是不确定的，所以这个指针我们称之为“野指针”。

野指针的危害：因为指针指向的空间是不确定的，所以指针可能会操作到非法的内存空间，导致程序运行奔溃。

```
int a = 100;
int *p;
*p = 1000;
/*因为p没有初始化/赋值，所以p的值是不确定的，如果此时p的值恰好等于a的地址(p == &a)， 那么*p=1000将a的
如果p的值恰好是内存上一块只读的内存空间，*p = 1000将导致程序异常退出，你可能会看到程序运行报错（段错误/
*/
```

6）空指针

为了标志指针变量没有指向任何变量(空闲可用)，C语言中，可以把NULL赋值给此指针，这样就标志此指针为空指针。

```
int *p = NULL;
```

NULL是一个值为0的宏常量：

```
#define NULL    ((void *)0)
```

注意：空指针的作用是防止指针变量变成野指针。如果用*访问空指针所指向的内存空间也会程序报错

7）笔试题:嵌入式系统经常具有要求程序员去访问某特定的内存位置的特点。在某工程中，要求设置一绝对地址为 0x67a9 的整型变量的值为 0xaa66

```
方法1：
    int *ptr;
ptr = (int *)0x67a9; //在内存地址编号的前面加上(int *)将地址编号这个无符号整型数据强制转换为
                     //(int*)指针类型，这样赋值符号左值和右值的数据类型一致
*ptr = 0xaa66;

方法2：
*(int *)(0x67a9) = 0xaa55;
```

注意：在实际工作中我们一般很少会将一个确定的内存地址赋值给一个指针变量，因为程序员一般不知道哪个内存地址是可用的！！！

9.2.3 不同类型指针变量之间的区别

1、int *p1 和 char *p2 的相同点是什么？

```
int x = 100;
int *p1 = &x;

char y = 'A';
char *p2 = &y;
```

相同点：

- 都是指针变量
- 都是用来保存一个内存地址编号
- 占用的内存空间大小一样

```
int *p1;
char *p2;
printf("%d %d\n", sizeof(p1), sizeof(p2));
```

我们发现p1和p2占用的内存空间为4/8，在32位机器上结果为4， 64位机器上结果为8。

> **思考：为什么指针变量占用的内存空间是 4 或者 8个字节呢？**
>
> 因为指针变量保存的是一个内存地址的编号！
>
> 32位机器内存地址编号最大值为 2^32-1，可以用一个4字节的变量保存
>
> 64位机器内存地址编号最大值为 2^64-1，可以用一个8字节的变量保存

2、int *p1 和 char *p2 的不同点是什么？

- 首先我们应该知道：内存中存储的只是二进制而已
- 之所以有 int float char 等数据类型是程序员希望将存储在内存中的二进制当作某种数据类型来处理而已

```
int x = 65;
printf("%c\n", x);
```

- int *p1的作用就是指针变量p1将他所指向的内存空间中的二进制当作int类型来处理
- char *p2的作用就是指针变量p2将他所指向的内存空间中的二进制当作char类型来处理

3、p1++ 和 p2++的区别

```
int main() {
    int x = 10;
    int *p1 = &x;

    char y = 'A';
    char *p2 = &y;
    printf("p1: %p, p2: %p\n", p1, p2);

    p1++;
    p2++;
    printf("p1: %p, p2: %p\n", p1, p2);

    return 0;
}
```

```
p1: 0x7ffeeaa74af8, p2: 0x7ffeeaa74aef
p1: 0x7ffeeaa74afc, p2: 0x7ffeeaa74af0
```

p1自增后和自增前值相差4

p2自增后和自增前值相差1

> **指针变量+n，不是指针往后偏移n个字节，而是指针变量往后偏移n个数据类型，例如：p1+3，表示指针p1往后偏移3个int类型的数据，指针变量p1的值+12 （ 3*sizeof(int) ）**

9.3 指针和数组

9.3.1 数组的指针

1、一个变量有一个地址，一个数组包含若干元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。所谓**数组的指针是指数组的起始地址。**

2、**数组名表示数组的首地址，因此数组名也是一种指针**

3、通过数组名访问数组中元素

```
int  ch[] = {1,2,3,4};
假如我们想访问ch中的第3个元素：ch[3] == 4
我们也可以通过指针法引用数组中的元素：比如 *(ch + 3)
那么，如果想访问第n个元素呢？
*(ch + n) 注意：n<=sizeof(ch)/sizeof(ch[0])-1
```

> 4、练习：假如有数组int a[4], 编写代码实现如下功能：
> 1、通过从键盘上输入数字对数组a的每一个元素进行赋值
> 2、打印出数组a中每一个元素的地址
> 3、通过指针法将数组a中的每一个元素的值打印出来

```
int a[4];
int i;
for (i = 0; i < 4; i++)
	scanf( “%d” , &a[i]);
for (i = 0; i < 4; i++)
{
	printf( “%p %d\n” , &a[i], *(a+i));
```

```
}
```

5、通过指针变量间接访问数组

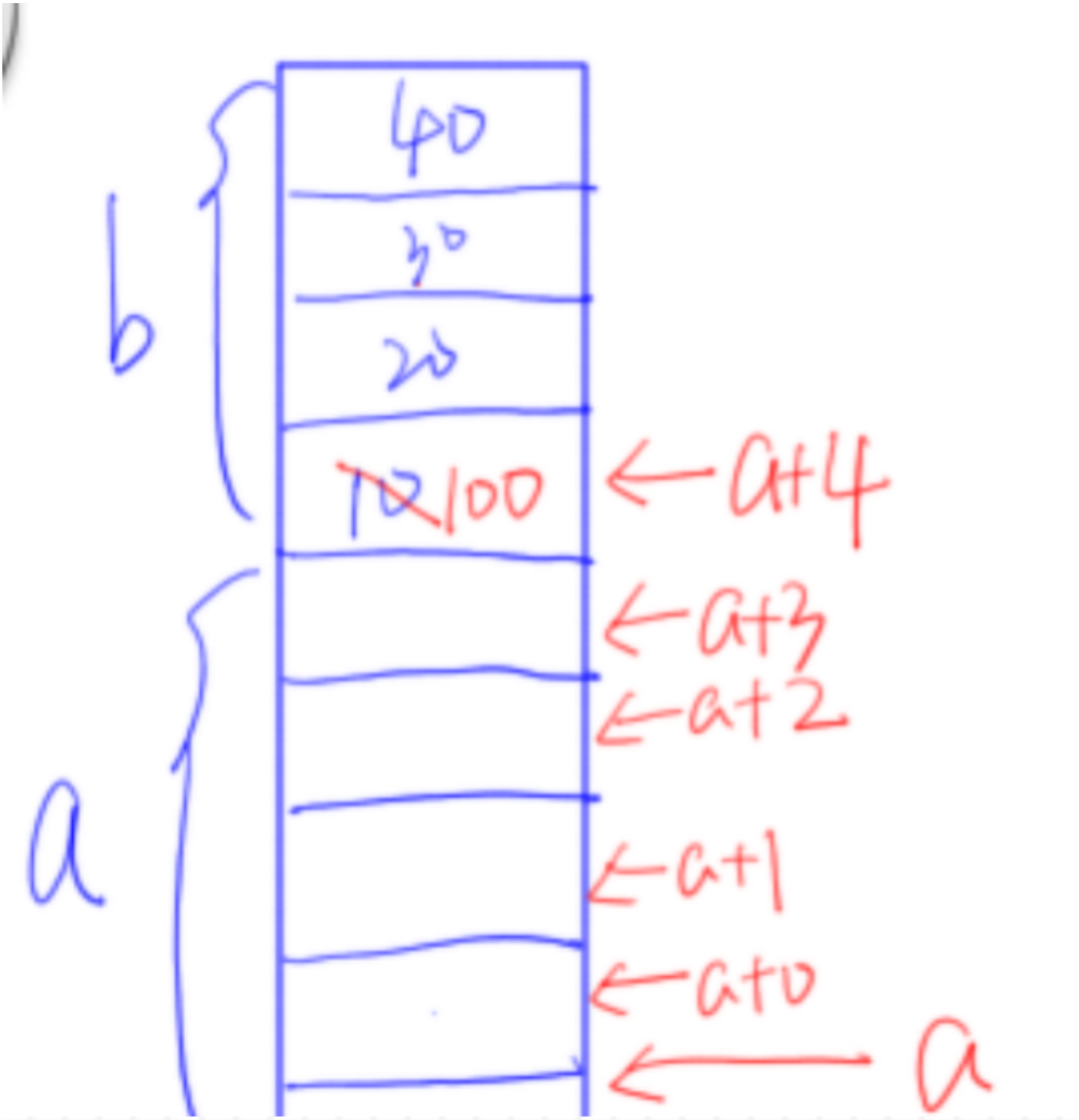
```
int a[4] = {1,2,3,4};
int *p;
p = a;
*(p + 2) = 100;


char ch[] = {'a', 'b', 'c'};
char *p2;
p2 = ch;
*(p2+1) = 'A';
```

```
    int ch[] = {1, 2, 3, 4};
    printf("%d\n", ch[4]);
    printf("%p %p\n", &ch[3], &ch[4]);


    //数组名：数组的首地址
    printf("ch: %p\n", ch);
    //数组中的第0个元素的地址：数组的首地址
    printf("&ch[0]: %p\n", &ch[0]);

    printf("%d %d\n", ch[3], *(ch+3));

    int a[4];
    int i;
    for (i = 0; i < 4; i++)
    {
        scanf("%d", &a[i]); //a+i
        getchar();
    }

    //打印数组中每个元素的地址
    for (i = 0; i < 4; i++)
        printf("%p\n", &a[i]);

    //通过指针法将数组a中的每一个元素的值打印出来
    for (i = 0; i < 4; i++)
        printf("%d\n", *(a+i));

    int *p;
    p = a; //指针p指向了数组a
```

```
//指针指向了一个数组，可以将指针当数组看待
for (i = 0; i < 4; i++)
    printf("%d\n", p[i]); //通过下标发访问数组中的元素
   // printf("%d\n", *(p+i));
```

6、数组指针越界

```
int b[4] = {10, 20, 30, 40};
int a[4];
a[4] = 100;
printf("a[4]: %d\n", a[4]);
printf("b[0]: %d\n", b[0]);

printf("a: %u, b: %u\n", &a, &b); //打印数组a和b的首地址
```



9.3.2 指针数组

1、指针数组顾名思义就是：存放指针的数组，本质是数组，数组中的每个元素都是指针

```
#include <stdio.h>

int main()
{
    int a = 10, b = 20, c = 30;
```

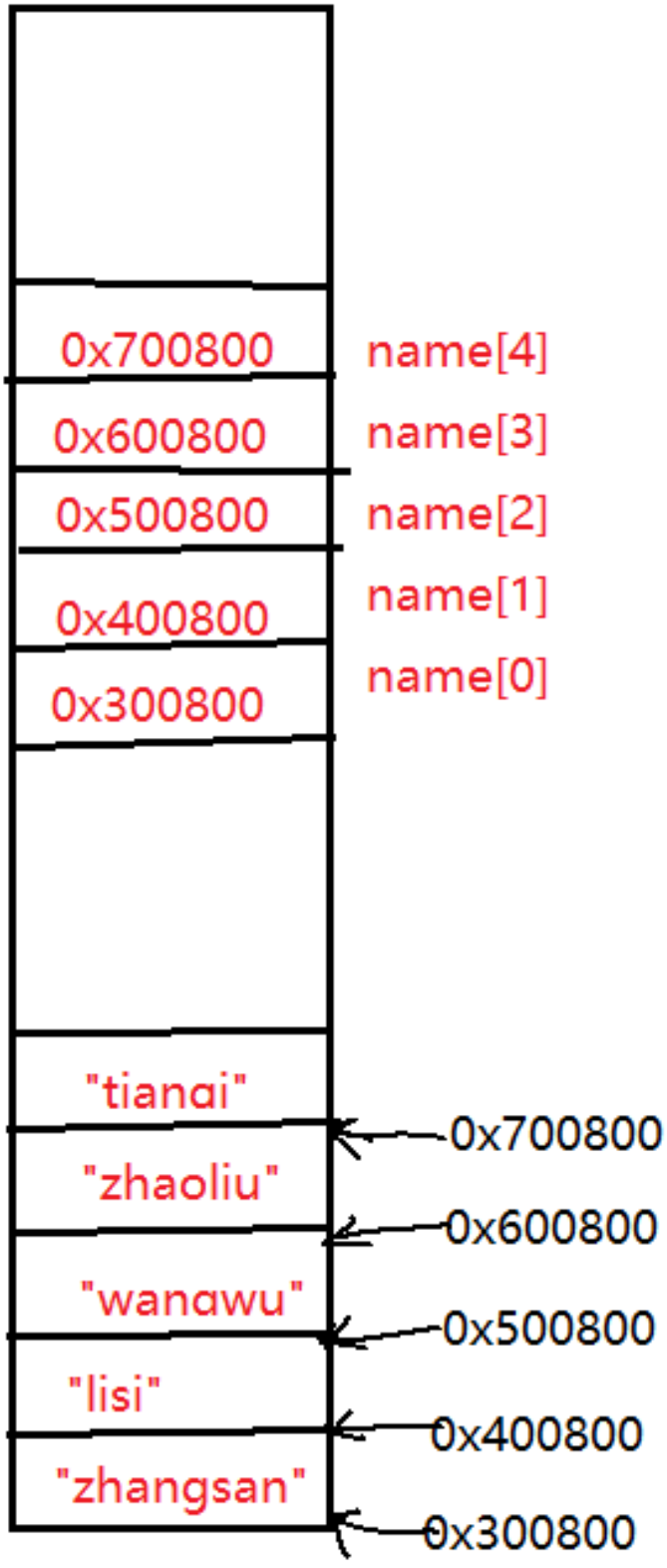
```
    int *p[3];
    p[0] = &a;
    p[1] = &b;
    p[2] = &c;
    return 0;
}
```

2、注意：int *p[3]; 等价于 (int *) p[3]; 因为[] 比 *的优先级要高先与p匹配。

3、思考：如何通过一个数组存储10个人的姓名？

> char *name[10] = {"zhangsan", "lisi", "wangwu", "zhaoliu, "tianqi"};
>
> name数组中保存了10个字符串常量的首地址（注意：没有保存字符串常量而是常量的首地址）

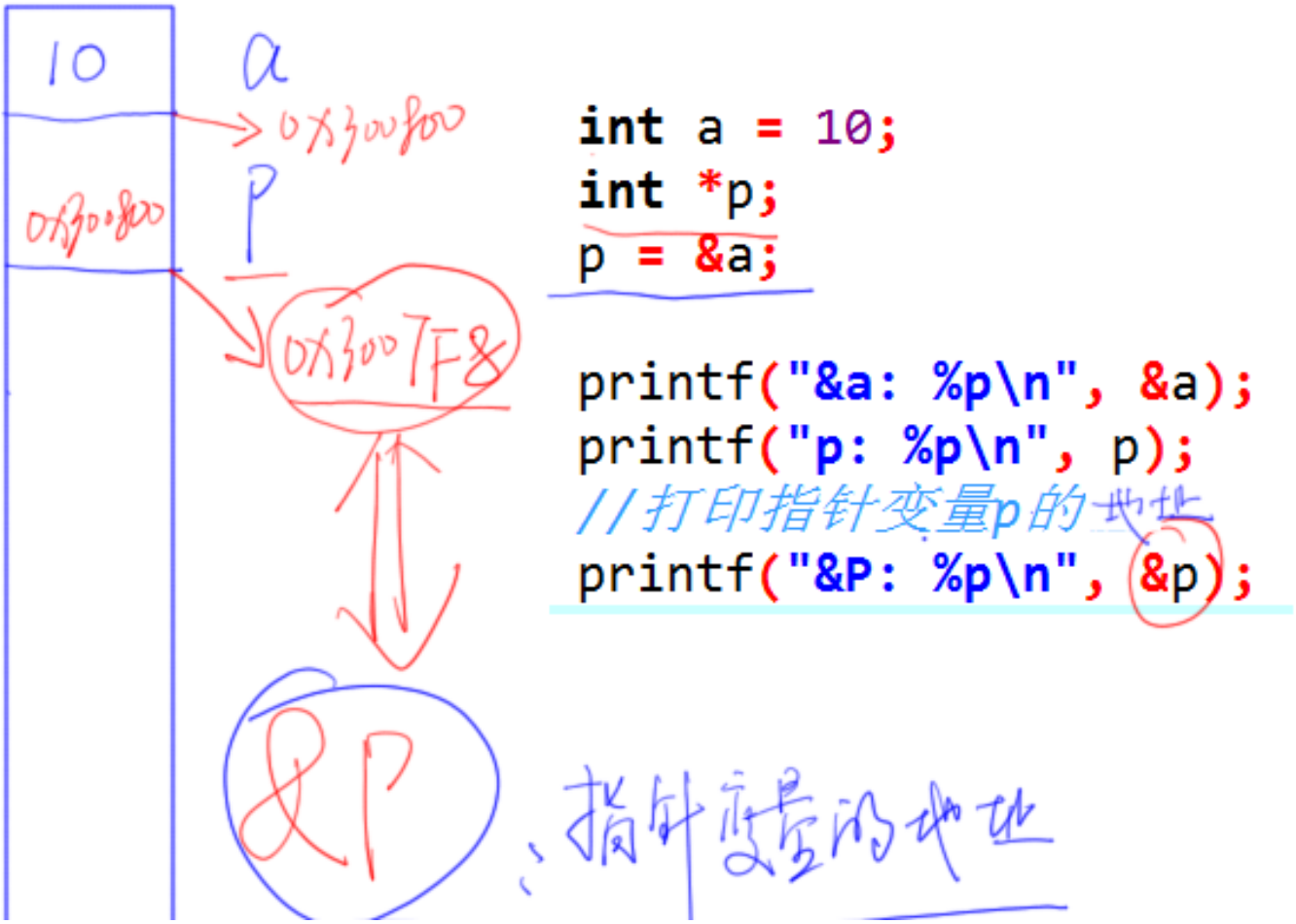
char *name[10] = {"zhangsan", "lisi", "wangwu", "zhaoliu, "tianqi"};



9.4 指针变量的地址

1、我们在定义一个指针变量的时候，编译器会分配一块空间来存储这个指针变量的值，分配的这块内存空间肯定有一个地址编码啦，那么这个地址编码肯定就是这个指针变量的地址啦

```
#include <stdio.h>

int main()
{
    int a = 10;
    int *p;
    p = &a;

    //将指针变量p的值以及变量a的地址打印出来（结果应该是两者相等）
    printf("p: %p, &a: %p\n", p, &a);

    //打印指针变量p的地址（存储指针变量p的内存空间的首地址）
    printf("&p: %p\n", &p);
    return 0;
}
```



2、强调：指针变量p的值保存的是另外一个变量a的地址0x300800，指针变量的地址是存储p这个指针变量的值的那块内存空间的首地址：0x3007F8，这块空间中保存的值是0x300800

9.5 一级指针作为函数的形参

1、函数的形参为数组

如果函数的形参是数组，该形参的定义方法如下：

```
void  func(int a[], int n)
{}
```

我们也可以将形参定义为指针类型：

```
void func(int *a, int n)
{}
```

在实际工作中我们通常使用第二种方法！

2、调用函数时需要传递字符串可将形参设计为char *类型

```
void func(char *p) //调用函数时将字符串的地址赋值给指针变量p
{
    printf("%c\n", p[0]);
}

int main()
{
    func("hello");
    return 0;
}
```

3、当形参为数组时，如果获取数组的长度呢？

```
void func1(int a[])
{
    //不能够通过a获取传递的实参数组的长度的！！
    printf("sizeof(a): %d\n", sizeof(a));
}

void func2(char b[])
{
    //不能够通过a获取传递的实参数组的长度的！！
    printf("sizeof(b): %d\n", sizeof(b));
}
```

> 为什么sizeof(a)和sizeof(b)的值都是8呢？
> 原因：编译器在编译的时候将a和b当做了指针来处理了！！！

4、注意：如果函数的形参为指针，在函数体中一般先对指针的值进行判断，判断指针的值是否为NULL

```
void test1(char *dest, char *src)
{
    if (NULL == dest || NULL == src)
        return ;
    strcpy(dest, src);
}
```

9.6 二级指针

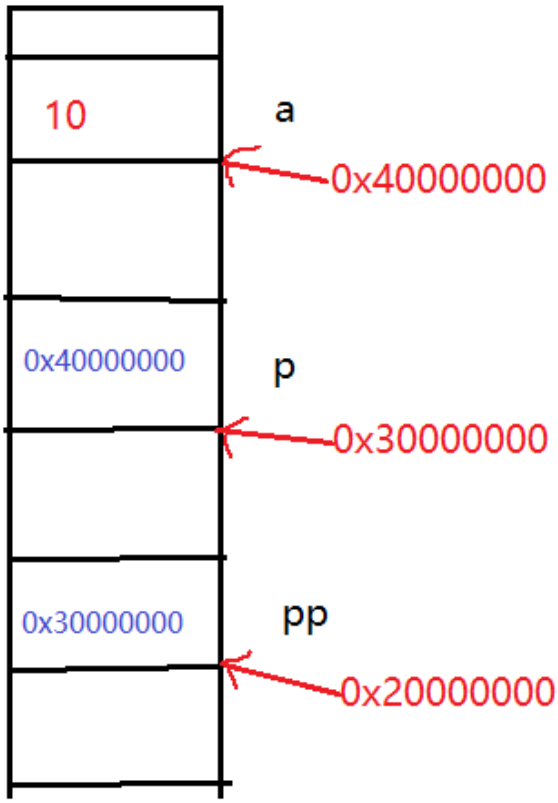
1、用一个指针变量保存一个一级指针变量的地址，这个指针我们称之为二级指针

2、二级指针的定义

> 数据类型 **变量名;

```
int a = 10;
int *p = &a; //p为一级指针，保存变量a的地址
int **pp = &p; //pp为二级指针 保存一级指针p的地址
```

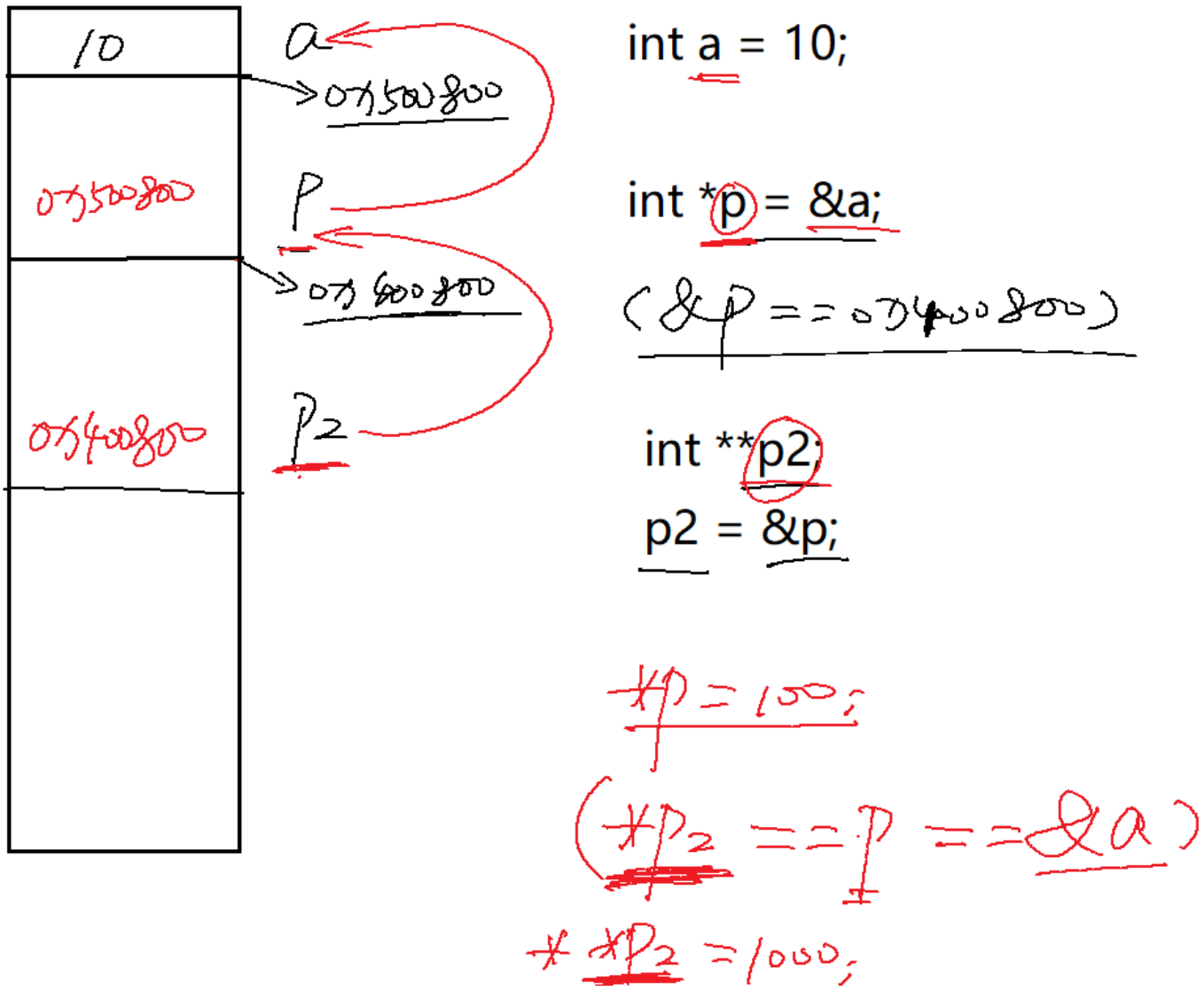
int a = 10;
int *p = &a; //p为一级指针，保存变量a的地址
int **pp = &p; //pp为二级指针 保存一级指针p的地址



3、二级指针的应用

```
//二级指针的使用
    int a = 10;
    int *p = &a;
    int **p2 = &p; //二级指针p2保存了一级指针p的地址（ p2指向了p ）
    int ***p3 = &p2; //三级指针

    //*p2 == p == &a
    printf("%p %p %p\n", *p2, p, &a);
    //**p2 == *p == *(&a) == a
    printf("%d %d %d %d\n", **p2, *p, *(&a), a);
```

```
**p2 = 100;
printf("%d %d %d %d\n", **p2, *p, *(&a), a);

//*p3 == p2 = &p
//**p3 == *p2 == p == &a
//***p3 == **p2 == *p == *(&a) == a
printf("***p3: %d\n", ***p3);
```



9.7 内存分配

1、在实际工作中，如果我们需要存储多个数据很多同学首先想到的是使用数组，但是因为数组的长度在定义完后是固定的所以往往不够灵活。

2、我们可以根据需要存储的数据类型先定义一个指针变量，例如：int *p; 然后根据实际需求使用malloc函数分配空间。

3、**malloc函数：**

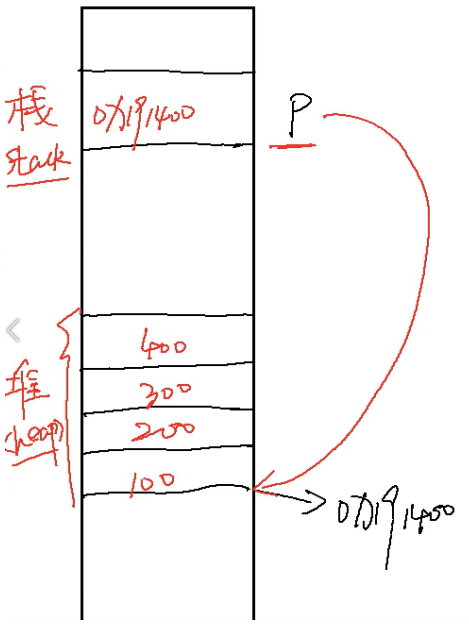
```
#include <stdlib.h>

void *malloc(size_t size);
```

功能：malloc 函数像系统申请size个字节的内存空间，并且返回一个指针，这个指针指向被分配的内存空间的首地址，并且申请的内存空间是在“堆”上的。**堆上的空间是需要手动申请，手动释放的！！否则就会造成内存泄漏。**

```
int *p;

//假如我们需要存储10个int类型的数据
p = (int *)malloc(10*sizeof(int));
```

注意：分配的空间为 10*sizeof(int)， 因为malloc申请分配的空间是以字节为单位的。

```
int *p;
    //通过指针变量p操作一块空间，可以存储4个int数据
    //向系统申请 4*sizeof(int)字节的内存空间
    p = (int *)malloc(4*sizeof(int)); //在堆上申请了4*sizeof(int)字节的内存空间

    //p的值：申请到的堆上的内存空间的首地址 （ 指针p指向申请到的堆上的空间 ）
    printf("p: %p\n", p);

    //通过指针变量p 来操作申请到的堆空间
    p[0] = 100;
    p[1] = 200;
    *(p+2) = 300;
    *(p+3) = 400;
```




```c
int *p;
//通过指针变量p操作一块空间，可以存储4个int数据
//向系统申请 4*sizeof(int)字节的内存空间
p = (int *)malloc( _Size: 4*sizeof(int)); //在堆上申请了4*sizeof(int)字节的内存空间

//p的值：申请到的堆上的内存空间的首地址 （指针p指向申请到的堆上的空间）
printf( _Format: "p: %p\n", p);

//通过指针变量p 来操作申请到的堆空间
p[0] = 100;
p[1] = 200;
*(p+2) = 300;
*(p+3) = 400;
```

栈空间：自动分配 自动回收

堆空间：手动申请 手动释放

4、内存释放：free函数

> #include <stdlib.h>
>
> void free(void *ptr);

功能：释放ptr所指向的内存空间

> 注意：
>
> free函数并不会修改指针变量的值！但是free执行完成以后指针所指向的原来的那块地址空间中的内容是不确定的！！
> 问题：
> 释放空间到底做了什么事情呢？
> 最重要的是：告诉系统这块内存空间可以给别人使用了！！！

```c
int main(int argc, char *argv[])
{
    char *p;
    //malloc分配的空间是在堆上的，需要手动释放
    p = (char *)malloc(10);
    strcpy(p, "hello");
    printf("p所指向的空间的内容：%s\n", p);//结果是hello

    //将p所指向的地址空间的首地址打印出来（就是将指针变量p的值打印出来）
    printf("p的值：%p\n", p);
    free(p);
    //将p所指向的地址空间的首地址打印出来（就是将指针变量p的值打印出来）
    printf("p的值：%p\n", p);

    //仔细观察，下面这条打印语句的结果
    printf("p所指向的空间的内容：%s\n", p); //结果不是hello了

    strcpy(p, "world");
    //再仔细观察，下面这条打印语句的结果
    printf("p所指向的空间的内容：%s\n", p); //结果是world

    return 0;
}
```

free函数调用完以后的使用技巧：

```c
int main(int argc, char *argv[])
{
    char *p;
    //malloc分配的空间是在堆上的，需要手动释放
    p = (char *)malloc(10);
    strcpy(p, "hello");

    //释放申请的内容
    free(p);
    //一般的我们在释放一个指针以后，将该指针变量的值赋值为NULL
    //这样做的目的是防止指针p变成野指针！
    p = NULL;
    /*如此一来，如果我们再继续操作指针p很可能会产生段错误！
    例如执行 strcpy(p, "world");
    可是如果不执行p=NULL 程序不就不会产生段错了吗？ 那是不是不执行p=NULL比较好呢？
    答案肯定是执行p = NULL比较好，因为这样可以防止p这个野指针将别的指针(p1)所指向的
    空间的内容进行修改（因为系统可能会吧原来指针p所释放的空间分配给p1指针)
    尽管不执行p= NULL 程序可能不会出现段错误，但是程序在运行的时候最终的结果很可能不是
    程序员原来所预期的结果，在程序的开发过程中这种错误是很难找到的！！！
    而执行p = NULL，程序在运行的时候产生了段错误，这种错误相对来说是比较容易找到的！
    */
    return 0;
}
```

5、如果之前分配的空间不够了怎么办呢？

我们可以使用**realloc**函数：

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
```

功能：在堆上分配一块size所指定的新的内存空间，空间大小单位为字节，并且还会将ptr所指向的空间中的内容拷贝到新的内存空间中，最后返回新的内存空间的额首地址。

示例代码：

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

int main()
{
    char *p;
    p = (char *)malloc(10);
    strcpy(p, "hello"); //向分配的空间中拷贝字符串
    printf("p所指向空间的首地址: %p\n", p);
    printf("p所指向空间的内容: %s\n", p);

    p = (char *)realloc(p, 20); //重新分配新的空间
    printf("p所指向新的空间的首地址: %p\n", p);
    printf("p所指向新的空间的内容: %s\n", p);

    //注意：分配的新的空间的首地址有可能有之前分配的空间首地址一样，也有可能不一样

    strcat(p, " world"); //追加字符串
    printf("p所指向新的空间的内容: %s\n", p);

    return 0;
}
```

6、思考一种情景，char *dest, *src; 通过一个函数将src所指向的地址空间中的内容拷贝到dest所指向的地址空间中,但是假设在调用函数前我们并不知道src的长度，这个时候我们需要将函数的形参设计为 二级指针！

```
void test2(char **dest, char *src)
{
    //通过二级指针dest给形参一级指针dest分配内存空间！
    *dest = (char *)malloc(strlen(src)+1);
    if (NULL == *dest || NULL == src)
        return ;
    strcpy(*dest, src);
}
```

```
void func4(char **dst)
{
   //*dst == p
   *dst = (char *)malloc(10); //在堆上申请了10个字节
   strcpy(*dst, "hello");
}

int main()
{
   char *p; //指针指向某个函数调用结束后 在函数体中申请的堆空间的首地址
   /*
    * 既然我希望让p指向一块堆空间，其实就是希望对p进行赋值，赋值为在函数中申请的堆空间的首地址
    * 如何在函数中对p进行赋值呢？必须在调用函数的时候传递p的地址！！！！
    * */
   func4(&p);
   printf("%s\n", p);
   free(p);
   return 0;
}
```

7、笔试题1：以下代码有什么问题？？

```
void GetMemory( char *p )
{
p = (char *) malloc( 100 );
}
void Test( void )
{
char *str = NULL;
GetMemory( str );
strcpy( str, "hello world" );
```

```
printf( str );
}
```

8、笔试题2：以下代码有什么问题？？

```
char *GetMemory( void )
{
char p[] = "hello world";
return p;
}
void Test( void )
{
char *str = NULL;
str = GetMemory();
printf("%s",str );
}
```

9、笔试题3：以下代码有什么问题？？

```
void GetMemory( char **p, int num )
{
*p = (char *) malloc( num );
}
void Test( void )
{
char *str = NULL;
GetMemory( &str, 100 );
strcpy( str, "hello" );
printf("%s",str );
}
```

9.8 字符串

9.8.1 字符串的定义及基本使用

1、什么是字符串？

```
被双引号引用的字符集合！例如：” hello” 、” world” ，或者是以'\0'结尾的字符数
组！！！
比如：char ch[] = {'h', 'e', '\0'}
注意：” hello” 中其实在在末尾也有'\0'只是我们看得到
```

> 也就是说：字符串一定是以'\0'结尾的！！
> 如何验证“hello”中有字符'\0'呢？
> printf("%d\n", *(p+5)); 输出的结果为整数0则说明结尾是'\0'

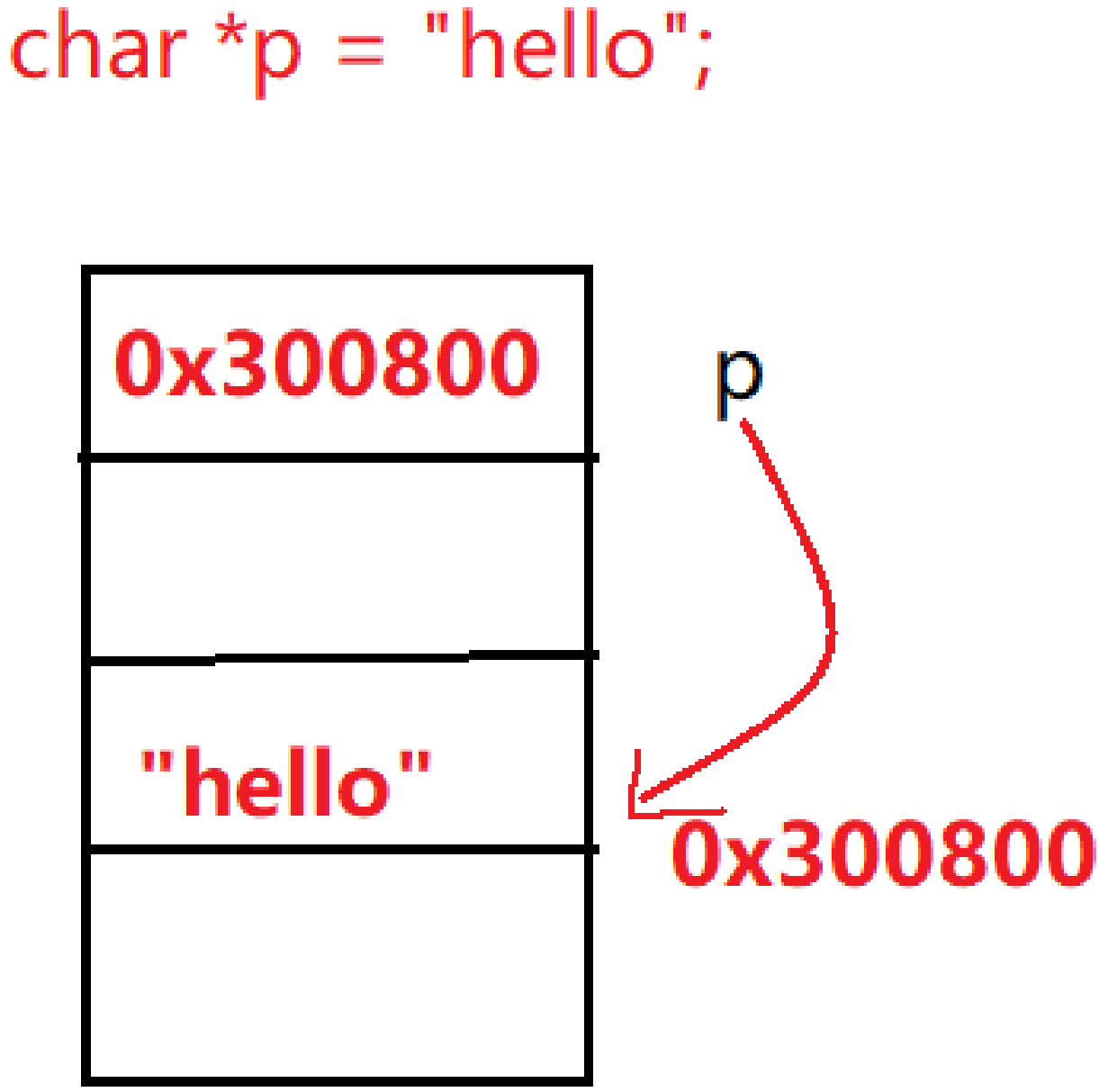
2、字符串和字符数组的联系及区别？

> 可以把字符串当做字符数组一样处理，字符数组不一定可以当做字符串处理，为什么？
> 因为字符数组中不一定有'\0'。
>
> char ch[] = { ‘a’ , 'b', '\0', 'c', 'd'};
>
> 这种情况我们可以吧数组ch当成字符串“ab”
>
> 在实际工作过程中我们经常会有如下需求，我们需要用一个字符数组用来保存多个字符，**并且需要将这多个字符当成字符串**，但是我们在定义字符数组的时候只知道需要保存的字符的最大的数目（假设是30），实际存储的时候存储的字符可能会小于30，如何保障把这些字符当成字符串呢？
>
> **c har name[30] ;**
>
> **memset(name, 0, sizeof(name));**
>
> **注意：在定义字符数组长度的时候通常要比实际要存储的字节数的数目+1（因为最后需要留一个字节存 ’ \0 ’ ）**

3、字符串的操作

```
int main()
{
    //字符串的操作：
    char *str;
    char ch[] = {"hello"};
    str = ch;
    //可以通过指针str操作ch
    //方法1：
    str[1] = 'a'; //当一个指针指向了一个数组以后，我们就可以通过指针使用下标法直接访问数组中的元素
    //方法2：
    *(str + 1) = 'a';

    //也可以将字符串常量赋值给一个指针变量
    char *p = "hello";
    /*注意：p是一个指针变量，应该保存的是一个地址，因此 char *p = "hello"; 并不是将字符串"hello"赋值给
      指针变量p，而是将存储字符串 "hello"的内存空间的首地址赋值给p
    */
```

```
    return 0;
}
```



4、将字符数组中的每个元素赋值为'\0'

```
#include <string.h>

void *memset(void *s, int c, size_t n);
```

功能：将指针s所指向的内存空间中的n个字节填充成c

示例代码：

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

int main()
{
    char ch[10];
    memset(ch, 0, sizeof(ch)); //memset(ch, 0, sizeof(ch));
    return 0;
}
```

5、请分析下面代码有何问题？

```
char *p = "hello";
p[0] = 'A';
```

9.8.2 常见的字符串处理函数

1、atoi函数

> #include <stdlib.h>
>
> int atoi(const char *nptr);

功能：将字符串转换成int类型的整数，返回转换后的整数值

```
int main()
{
    char *p = "1234";
    int val = atoi(p);
    printf("%d\n", val);

    return 0;
}
```

练习：编写代码实现atoi函数

2、atof函数

> #include <stdlib.h>
>
> double atof(const char *nptr)

功能：将字符串转换成double类型的浮点数，返回转换后的浮点数

```
int main()
{
    char *p = "123.456";
    double val = atof(p);
    printf("%f\n", val);
    return 0;
}
```

3、strlen函数

> #include <string.h>
>
> size_t strlen(const char *s);

功能：计算字符串的长度

```
int main()
{
    char *ch = "hello";
    char arr[] = {'A', 'B', 'C', '\0', 'D'};
    printf("%d %d\n", strlen(ch), strlen(arr));
    return 0;
}
```

思考：为什么strlen(arr)得结果是3呢？

练习：编程实现strlen函数

```
int my_strlen(char ch[]) //int my_strlen(char *ch)
{
        int cnt = 0;
        int i = 0;
        while (ch[i] != '\0')
        {
                cnt++;
                i++;
        }
        return cnt;
}
```

4、strcpy函数

> #include <string.h>
>
> char *strcpy(char *dest, const char *src);

> 功能：把src所指向的字符串复制到dest所指向的空间中，'\0'也会拷贝过去
>
> 参数：
>
> dest：目的字符串首地址
>
> src：源字符首地址
>
> 返回值：
>
> 成功：返回dest字符串的首地址
>
> 失败：NULL

示例代码：

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

int main()
{
    char *src = "hello";
    char dst[10] = {0};

    strcpy(src, dst);
    printf("dst: %s\n", dst);

    return 0;
}
```

注意：

1、dst的空间一定要大于从src中需要拷贝的内容所占用的空间，至少大1个字节（因为要给'\0'预留空间）

```
int main() {
    /*
    char *src = "hello";

    char dst[10];
    strcpy(dst, src);
    printf("%s\n", dst);

    */
```

```
    char a[4];
    char b[4]; //hell

    strcpy(b, "hello");

    printf("b: %s\n", b); //b: hello
    printf("a: %s\n", a); //a: o

    return 0;
}
```

2、stcpy的实现

```
char *strncpy(char *dest, const char *src, size_t n)
        {
            size_t i;

            for (i = 0; i < n && src[i] != '\0'; i++)
                dest[i] = src[i];
            for ( ; i < n; i++)
                dest[i] = '\0';

            return dest;
        }
```

5、strncpy函数

> #include <string.h>
>
> char *strncpy(char *dest, const char *src, *size_t* n);
>
> 功能：把src指向字符串的前n个字符复制到dest所指向的空间中，是否拷贝结束符看指定的长度是否包含'\0'。
>
> 参数：
>
> dest：目的字符串首地址
>
> src：源字符首地址
>
> n：指定需要拷贝字符串个数
> 返回值：

> 成功：返回dest字符串的首地址
>
> 失败：NULL

注意：
1、strncpy最多拷贝n个字节，如果前n个字节中没有'\0',最终也不会拷贝'\0'
2、如果src的长度小于n，则会写入一些'\0'以保障总共写入n个Bytes
strncpy在拷贝的时候如果前面的n个字节中已经有'\0'了，则只拷贝到'\0',但是依然会往dest中继续写入'\0'以保障总共写入n个Bytes

strncpy的实现：

```
char *strncpy(char *dest, const char *src, size_t n)
        {
            size_t i;

            for (i = 0; i < n && src[i] != '\0'; i++)
                dest[i] = src[i];
            for ( ; i < n; i++)
                dest[i] = '\0';

            return dest;
        }
```

6、strcpy和strncpy的区别

- 如果你能够100%肯定dest的空间比src的空间大，可以使用strcpy
- 在实际工作中为了避免溢出情况的产生我们尽量多使用strncpy。
  问题又来了！如果使用strncpy的时候n比dest所指向的内存空间的大小要大那不是还是会差生溢出吗？那么如何去规避这种溢出情况的产生呢？
  方法：先判断dest的长度len和n的大小,如果len>=n,则拷贝n个元素，如果len<n，则拷贝len个元素！

7、strcat函数

> #include <string.h>
>
> char *strcat(char *dest, const char *src);
>
> 功能：将src字符串连接到dest的尾部， ‘\0’ 也会追加过去
>
> 参数：

> dest：目的字符串首地址
>
> src：源字符首地址
> 返回值：
>
> 成功：返回dest字符串的首地址
>
> 失败：NULL

示例代码：

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

int main()
{
    char ch[20]; //char *ch;
    memset(ch, 0, sizeof(ch)); //ch = (char *)malloc(20);

    strcpy(ch, "hello");
    strcat(ch, "world");
    printf("%s\n", ch);
    return 0;
}
```

注意：ch的空间要足够大！

strcat的实现：

```
char *strncat(char *dest, const char *src, size_t n)
{
            size_t dest_len = strlen(dest);
            size_t i;

            for (i = 0 ; i < n && src[i] != '\0' ; i++)
                dest[dest_len + i] = src[i];
            dest[dest_len + i] = '\0';

            return dest;
        }
```

8、strncat函数

> #include <string.h>

> char *strcat(char *dest, const char *src);
>
> 功能：将src字符串连接到dest的尾部，‘\0’也会追加过去
>
> 参数：
>
> dest：目的字符串首地址
>
> src：源字符首地址
> 返回值：
>
> 成功：返回dest字符串的首地址
>
> 失败：NULL

注意：

- 如果src中的内容长度为m小于n个字节，那么只追加m+1个字节（最后会自动追加一个'\0'）
- 如果src中的内容长度为m小于n个字节，那么追加n+1个字节（最后会自动追加一个'\0'）

示例代码：

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

int main()
{
    char *p;
    p = (char *)malloc(20);
    strcpy(p, "hello");
    strncat(p, "world", 3);
    printf("%s\n", p);

    memset(p, 0, 20); //这是什么意思呢？
    strcpy(p, "hello");
    strncat(p, "world", 7);
    printf("%s\n", p);

    return 0;
}
```

strcat的实现：

```
char * strncat(char *dest, const char *src, size_t n)
        {
```

```
    size_t dest_len = strlen(dest);
    size_t i;

    for (i = 0 ; i < n && src[i] != '\0' ; i++)
        dest[dest_len + i] = src[i];
    dest[dest_len + i] = '\0';

    return dest;
}
```

9、strcmp函数

> #include <string.h>
>
> int strcmp(const char *s1, const char *s2);
>
> 功能：比较 s1 和 s2 的大小，比较的是字符ASCII码大小。
>
> 参数：
>
> s1：字符串1首地址
>
> s2：字符串2首地址
> 返回值：
>
> 相等：0
>
> 大于：>0 在不同操作系统strcmp结果会不同 返回ASCII差值
>
> 小于：<0

原理：

> strcmp的执行的逻辑：将s1和s2中对应位置的字符一一比较，直到两个字符串全部遍历完成（如果还没有发现有不同的字符则说明两者相等），或者有两个字符不相等，则比较结束。如果s1中的字符的ASCII码比s2中的大则返回1，否则返回-1
>
> 注意：有的编译器两个字符串不同的时不是返回1或者-1， 而是返回两个字符串中第一个不相等的两个字符的ASCII码的差

```
printf("%d\n", strcmp("hello", "hello")); //结果为0
printf("%d\n", strcmp("hello", "heLlo")); //结果为1
printf("%d\n", strcmp("Hello", "heLlo")); //结果为-1
```

10、strncmp函数

> #include <string.h>
>
> int strncmp(const char *s1, const char *s2, *size_t* n);
>
> 功能：比较 s1 和 s2 前n个字符的大小，比较的是字符ASCII码大小。
>
> 参数：
>
> s1：字符串1首地址
>
> s2：字符串2首地址
>
> n：指定比较字符串的数量
>
> 返回值：
>
> 相等：0
>
> 大于： > 0
>
> 小于： < 0

练习1：

> 在键盘上输入一串字符串，判断这个字符串和在程序中所设定的字符串是否相等（最多比较6个字符）（不区分大小写）
>
> char ch1[10];
>
> char key[] = "A18Cd9";
>
> memset(ch1, 0, sizeof(ch));
>
> printf("请输入验证码：%s\n", key);
>
> scanf("%s", ch1);
>
> printf("%s\n", ch1);

练习2：

> 在键盘上输入一串字符串，将其中的非字母和非数字的字符删除！
>
> char ch1[10];

> memset(ch1, 0, sizeof(ch));
>
> printf(“请在键盘上输入一串字符（长度小于10）\n”);
>
> scanf("%s", ch1);
>
> printf("%s\n", ch1);

11、strstr函数

> #include <string.h>
>
> char *strstr(const char *haystack, const char *needle);
>
> 功能：在字符串haystack中查找字符串needle出现的位置
>
> 参数：
>
> haystack：源字符串首地址
>
> needle：匹配字符串首地址
>
> 返回值：
>
> 成功：返回第一次出现的needle地址
>
> 失败：NULL

我们经常使用strstr来判断某个字符串是否时另外一个字符串的字串！

```
char src[] = "ddddabcd123abcd333abcd";
char *p = strstr(src, "abcd");
printf("p = %s\n", p);
```

12、strtok函数

> #include <string.h>
>
> char *strtok(char *str, const char *delim);
>
> 功能：来将字符串分割成一个个片段。当strtok()在参数s的字符串中发现参数delim中包含的分割字符时, 则会将该字符改为\0 字符，当连续出现多个时只替换第一个为\0。
>
> 参数：
>
> str：指向欲分割的字符串

> delim：为分割字符串中包含的所有字符
>
> 返回值：
>
> 成功：分割后字符串首地址
>
> 失败：NULL

注意：

- 在第一次调用时：strtok()必需给予参数s字符串
- 往后的调用则将参数s设置成NULL，每次调用成功则返回指向被分割出片段的指针

```
    char a[100] = "abc-efg-hijk-lmn";
    char *s = strtok(a, "-");//将"-"分割的子串取出
    while (s != NULL)
    {
        printf("%s\n", s);
        s = strtok(NULL, "-");
    }
```

字符串知识点完整代码：

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

int main() {
//strcpy函数
    /*
    char *src = "hello";

    char dst[10];
    strcpy(dst, src);
    printf("%s\n", dst);


    char a[4];
    char b[4]; //hell

    strcpy(b, "hello");

    printf("b: %s\n", b); //b: hello
    printf("a: %s\n", a); //a: o
*/
```

```
 //strncpy
 /*
 char *src = "hello\0world";
 char dst[10];
 //char *dst;
// dst = malloc(10);
 memset(dst, 'A', 10);
 printf("dst: %s\n", dst);
 strncpy(dst, src, 3);
 printf("dst: %s\n", dst);

 memset(dst, 'A', 10);
 strncpy(dst, src, 8);
 printf("dst: %s\n", dst); //hello\0AAAA

 char *src1 = "hello";
// memset(dst, 'A', 10);
 memset(dst, 0, 10);
 strncpy(dst, src, sizeof(dst)-1);
 printf("dst: %s\n", dst);

 int i;
 for (i = 0; i < 10; i++)
     printf("%c ", dst[i]);
 printf("\n");
 */

 //strcat 字符串拼接
 /*
 char dst[20];
 memset(dst, 0, sizeof(dst));
 char *src = "he\0llo";

 strcpy(dst, "world");
 strcat(dst, src);
 printf("dst: %s\n", dst);
 */

 //strcmp函数
 /*
 char *s1 = "hello world";
 char *s2 = "hello";
 printf("%d\n", strcmp(s2, s1));
 //注意strcmp有个很大的坑：返回值 不同的编译器s1 和 s2 不等的时候返回值是不一样的
 //有的编译器s1 < s2时，返回-1，但是有的编译器返回的是s1和s2中不等的那两个字符的ASCII码的差  s1:heLlo  s2
 //有的编译器s1 > s2时，返回 1，但是有的编译器返回的是s1和s2中不等的那两个字符的ASCII码的差  s1:hello  s2
```

```
//因此为了提高程序的可移植性！！ 同学们千万不要使用如下代码判断s1是否大于s2  if (strcmp(s1, s2) == 1) 而要

char *stus[] = {"zhangsan", "lisi", "wangwu", "liusan", "huangsan"};

int i;
for (i = 0; i < 3; i++)
{
    if (strcmp(stus[i], "lisi") == 0)
        printf("hello , lisi: %d\n", i);
}

printf("%d\n", strncmp(s1, s2, 5));
 */

//strstr函数
/*
char *stus[] = {"zhangsan", "lisi", "wangwu", "liusan", "huangsan"};
int i;
char *p;
for (i = 0; i < 5; i++)
{
    //找出名字中带san的学生
    if (p = strstr(stus[i], "san"))
    {
        printf("%s\n", stus[i]);
        printf("%p, %p\n", stus[i], p);
    }
}
 */

char data[] = {"##name=zhangsan;score=99.5;age=18##"};
//第一步用;分割
char *p;
int n = 0;
int len;
p = strtok(data, ";");
printf("%s\n", p);
len = strlen(p);
while (*p != '=' && n <= len)
{
    p++;
    n++;
}
if (n <= len)
{
    p++;
    char name[10];
```

```
    memset(name, 0, 10);
    strcpy(name, p);
    printf("name: %s\n", name);
}

//第二次对剩下的数据使用;分割
char name[10];
int age;
float score;
while (p =strtok(NULL, ";"))
{
    char *tmp = p;
    printf("%s\n", p);
    len = strlen(p);
    n = 0;
    while (*p != '=' && n <= len) //找=
    {
        p++;
        n++;
    }
    if (n <= len)
    {
        p++;
        if (strstr(tmp, "age=") != NULL)
        {
            age = atoi(p);
            printf("age: %d\n", age);
        }
        else if (strstr(tmp, "score=") != NULL)
        {
            score = atof(p);
            printf("score: %f\n", score);
        }
    }
}
/*
p = strtok(NULL, ";");
printf("%s\n", p);
len = strlen(p);
n = 0;
while (*p != '=' && n <= len)
{
    p++;
    n++;
}
if (n <= len)
{
```

```
        p++;
        int age;
        age = atoi(p); //18
        printf("age: %d\n", age);
    }

    p = strtok(NULL, ";");
    printf("%s\n", p);
    len = strlen(p);
    n = 0;
    while (*p != '=' && n <= len)
    {
        p++;
        n++;
    }
    if (n <= len)
    {
        p++;
        float score;
        score = atof(p);
        printf("score: %f\n", score);
    }
     */
    return 0;
}
```

9.9 指针和函数

9.9.1 函数指针

1、基本概念

> 什么是函数指针呢？
> 本质：指针
> 作用：用来指向一个函数

2、定义一个函数指针类型

> typedef int (*FUNC)(int, int);
>
> 定义了一个函数指针类型，类型名称为FUNC，该函数指针类型的变量可以指向这么一类函数：返回值为int，形参为int, int

3、定义函数指针变量

> FUNC f; //f是函数指针变量

4、通过函数指针变量调用函数

```
typedef int (*FUNC)(int, int);

int test(int x, int y)
{
    return x>y?x:y;
}

int main()
{
    FUNC f;
    f = test;
    int x ;
    x = f(10, 20);
    return 0;
}
```

5、练习

> 假如有a(), b(), c(), d()四个函数，编写代码实现：随机调用其中的函数

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

#include <time.h>

typedef int (*FUNC)(int, int);

int max(int x, int y)
{
    return x>y?x:y;
}

int add(int x, int y)
{
    return x+y;
}

typedef void (*SKILL)();
//假设我开发了一个游戏，某个英雄有4个技能，每个技能的发射都是独立的函数（4个技能就有4个函数），我要随机
void skill1()
```

```
{
    printf("skill1\n");
}
void skill2()
{
    printf("skill2\n");
}
void skill3()
{
    printf("skill3\n");
}
void skill4()
{
    printf("skill4\n");
}

int main()
{
    FUNC f;
    f = max;

    printf("%d\n", f(10, 20)); //max(10, 20);

    f = add;
    printf("%d\n", f(10, 20));

    //s是个数组，是个函数指针数组，数组中的每个元素都是一个函数指针
    SKILL s[4] = {skill1, skill2, skill3, skill4};

    srand(time(NULL));
    int n = rand()%4; //rand函数能够产生一个随机数
    s[n]();

    return 0;
}
```

9.9.2 指针函数

> 本质：函数，返回值是指针的函数(单纯的文字游戏)
>
> void *f();
> int *f();
> char *f();
>
> strcpy strncpy strcat strncat strstr strtok 这些都是指针函数

9.10 练习

> 假如有数组char *ptr[4];使用malloc对数组p中的每一个元素进行内存分配
> for(i = 0; i < 4; i++)
> p[i] = (char *)malloc(20);
> 分别将数组p中每一个元素所指向的地址空间拷贝如下字符串：
> ZHAngsan WanGWu zhaOLiU TianQI
> 实现如下功能：
> 1、将所有的大写转换为小写（不能使用字符带小写转换函数）
> 2、对数组p中的每一个元素指向的字符串按照字母表的先后顺序进行排序
> （将 zhangsan wangwu zhaoliu tianqi 按照字母表进行排序）

9.11 笔试题

> a) 一个整型数（An integer） int a;
> b)一个指向整型数的指针（ A pointer to an integer）int *p;
> c)一个指向指针的的指针，它指向的指针是指向一个整型数（ A pointer to a pointer toan integer） int **p; //二级指针*
> *d)一个有 10 个整型数的数组（ An array of 10 integers）int a[10];*
> *e) 一个有 10 个指针的数组，该指针是指向一个整型数的。（An array of 10 pointers to integers）*
> *本质是数组 int* p[10];
>
> f) 一个指向有 10 个整型数数组的指针（ A pointer to an array of 10 integers）本质：指针， int (*p)[10]; //int[10] (*p)
> g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数（A pointer to a function that takes an integer as an argument and returns an integer）本质：指针 int (*p)(int); //p是指针变量 int(int) (*p);
> typedef int (*FUNC)(int); FUNC p;
> h)一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数（ An array of ten pointers to functions that take an integer argument and return an integer ）
> 本质：数组， int (*p[10])(int); // int(int) (*p[10]);
> typedef int (*FUNC)(int);
> FUNC f[10];