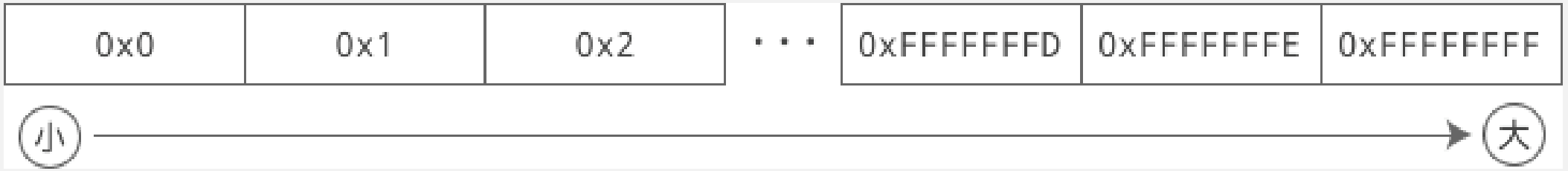
中山大學
SUN YAT-SEN UNIVERSITY

指针

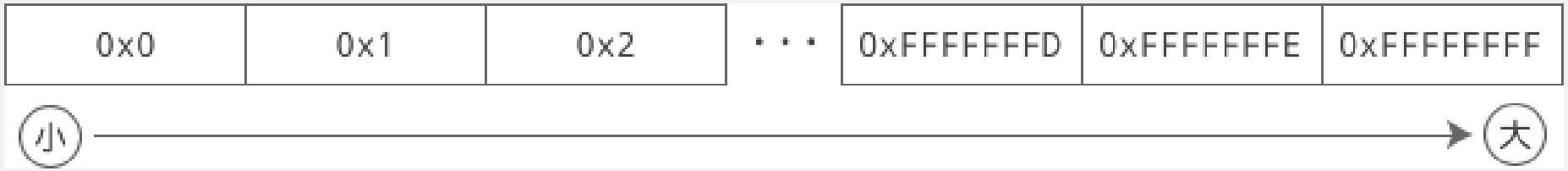
中山大学计算机学院

讲课人：万海

Address and Pointer



Address and Pointer




```
#include <stdio.h>

int main(){
    int a = 100;
    int b = 0;
    char str[20] = "www.sysu.edu.cn";
    printf("%d, %d, %d\n", sizeof(a), sizeof(b), sizeof(str));
    printf("%#X, %#X, %#X\n", &a, &b, str);
    return 0;
}
```

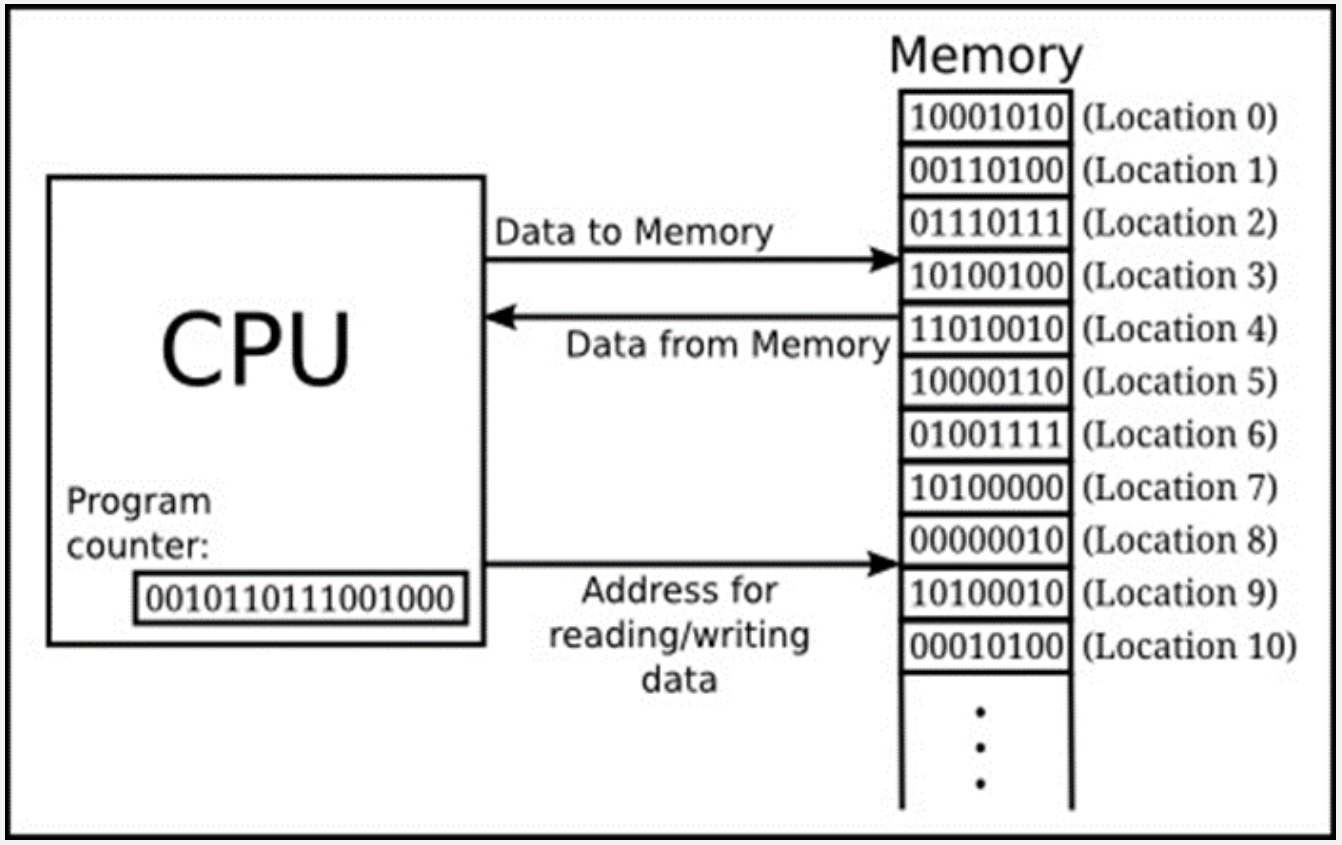
4,4,20

0X28FE4C, 0X28FE48, 0X28FE30

%#X：以十六进制形式输出，并附带前缀0X

%p：专门用来以十六进制形式输出地址

一切都是地址



CPU 访问内存时需要的是地址，而不是变量名和函数名。

变量名和函数名只是地址的一种助记符，当源文件被编译和链接成可执行程序后，它们都会被替换成地址。

编译和链接过程的一项重要任务就是找到这些名称所对应的地址。

指向指针的指针

```
#include <stdio.h>

int main(){
    int a =100;
    int *p1 = &a;
    int **p2 = &p1;
    int ***p3 = &p2;

    printf("%d, %d, %d, %d\n", a, *p1, **p2, ***p3);
    printf("&p2 = %#X, p3 = %#X\n", &p2, p3);
    printf("&p1 = %#X, p2 = %#X, *p3 = %#X\n", &p1, p2, *p3);
    printf(" &a = %#X, p1 = %#X, *p2 = %#X, **p3 = %#X\n", &a, p1, *p2, **p3);
    return 0;
}
```

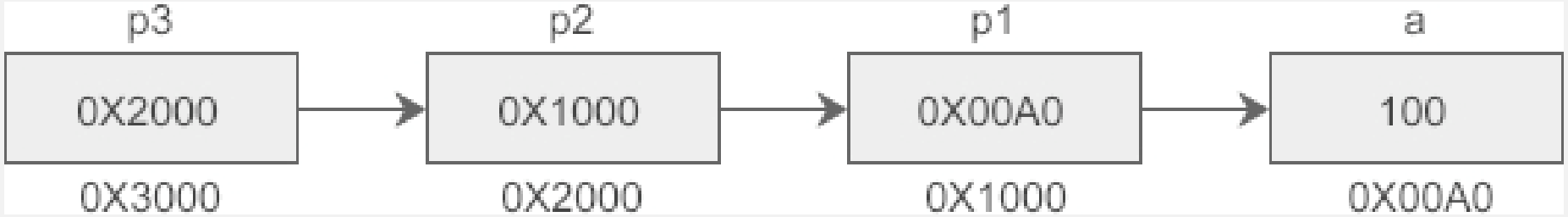
指向指针的指针

```
#include <stdio.h>

int main(){
    int a =100;
    int *p1 = &a;
    int **p2 = &p1;
    int ***p3 = &p2;

    printf("%d, %d, %d, %d\n", a, *p1, **p2, ***p3);
    printf("&p2 = %#X, p3 = %#X\n", &p2, p3);
    printf("&p1 = %#X, p2 = %#X, *p3 = %#X\n", &p1, p2, *p3);
    printf(" &a = %#X, p1 = %#X, *p2 = %#X, **p3 = %#X\n", &a, p1, *p2, **p3);
    return 0;
}
```

***p3等价于*(*(*p3))



数组和指针再理解

```
#include <stdio.h>

int main(){
    int a[6] = {0, 1, 2, 3, 4, 5};
    int *p = a;
    int len_a = sizeof(a) / sizeof(int);
    int len_p = sizeof(p) / sizeof(int);
    printf("len_a = %d, len_p = %d\n", len_a, len_p);
    return 0;
}
```

len_a = 6, len_p = 1

数组是一系列数据的集合，没有开始和结束标志；p 仅仅是一个指向 int 类型的指针。

数组 a，它的类型是int [6]；指针变量 p，它的类型是int *

数组和指针再理解

```
#include <stdio.h>

int main(){
    int a[6] = {0, 1, 2, 3, 4, 5};
    int *p = a;
    int len_a = sizeof(a) / sizeof(int);
    int len_p = sizeof(p) / sizeof(int);
    printf("len_a = %d, len_p = %d\n", len_a, len_p);
    return 0;
}
```

数组转换为指针（数组下标[ ]）

```
int a = {1, 2, 3, 4, 5}, *p, i = 2;
```

a[i]

```
p = a;
p[i];
```

```
p = a;
*(p + i);
```

```
p = a + i;
*p;
```

数组转换为指针（数组作函数参数）

```
void func(int *parr){ ..... }
void func(int arr[]){ ..... }
void func(int arr[5]){ ..... }
```

C标准规定:作为“类型的数组”的形参应该调整为“类型的指针”

以上三种形式的函数定义是等价的

数组转换为指针

```
#include <stdio.h>
int main(){
    char *lines[5] = {
        "Sun Yat-sen University",
        "Programming",
        "Techniques",
        "is",
        "the best"
    };
    char *str1 = lines[1];                  char *str2 = *(lines + 3);
    char c1 = *(*(lines + 4) + 6);                  char c2 = (*lines + 5)[5];
    char c3 = *lines[0] + 2;

    printf("str1 = %s\n", str1);                    printf("str2 = %s\n", str2);
    printf("  c1 = %c\n", c1);              printf("  c2 = %c\n", c2);
    printf("  c3 = %c\n", c3);
    return 0;
}
```

数组转换为指针

```
#include <stdio.h>
int main(){
    char *lines[5] = {
        "Sun Yat-sen University",
        "Programming",
        "Techniques",
        "is",
        "the best"
    };
    char *str1 = lines[1];
    char c1 = *(*(lines + 4) + 6);
    char c3 = *lines[0] + 2;

    printf("str1 = %s\n", str1);
    printf("  c1 = %c\n", c1);
    printf("  c3 = %c\n", c3);
    return 0;
}
```

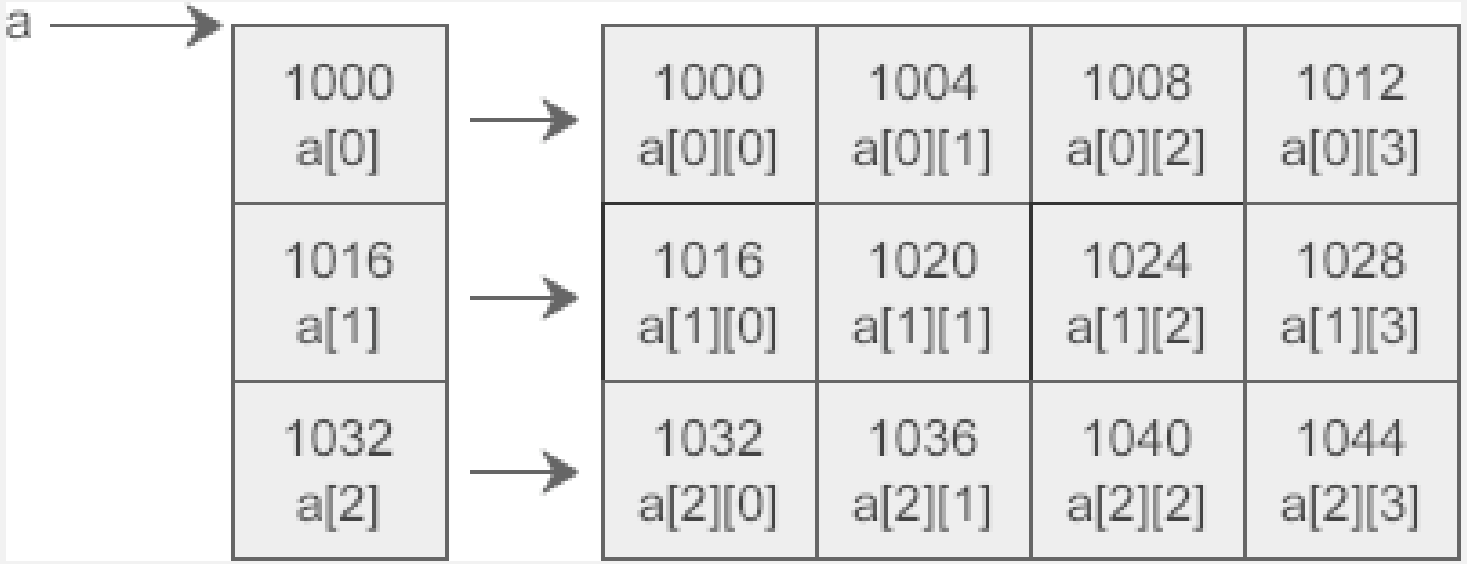
```
char *str2 = *(lines + 3);
        char c2 = (*lines + 5)[5];

        printf("str2 = %s\n", str2);
printf("  c2 = %c\n", c2);
```

```
str1 = Programming
str2 = is
 c1 = s
 c2 = n
 c3 = U
```

二维数组指针

int a[3][4] = { {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };

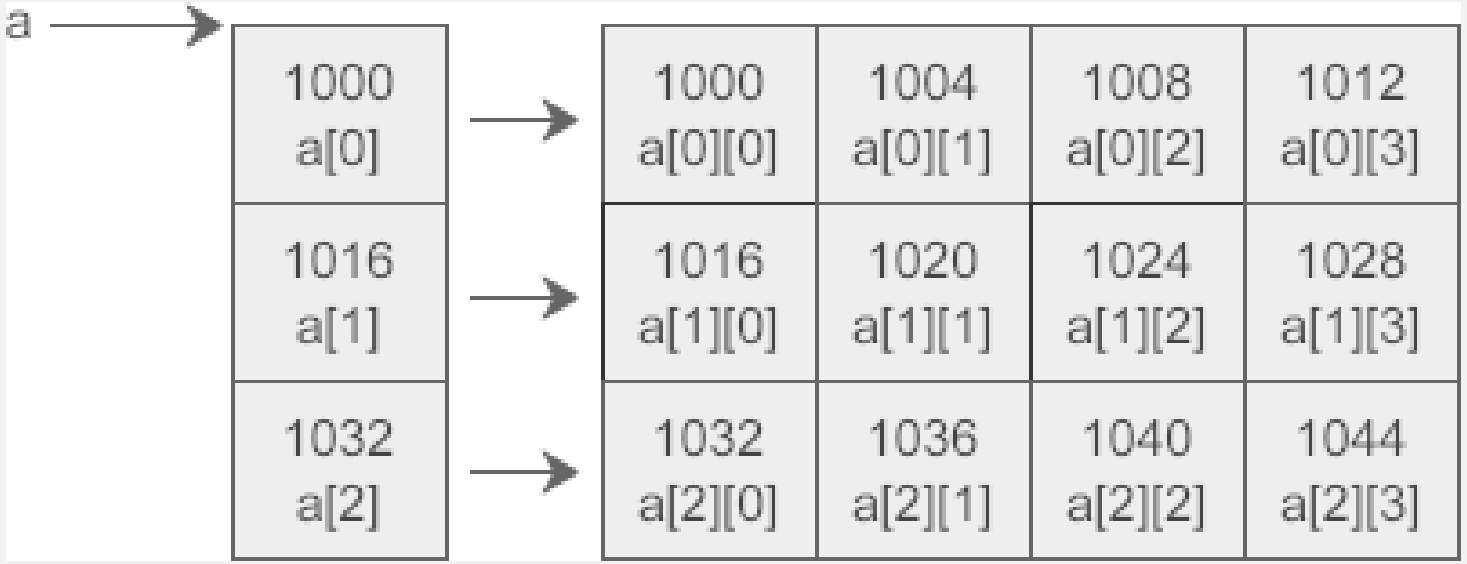


int (*p)[4] = a;

括号中的*表明 p 是一个指针，它指向一个数组，数组的类型为int [4]，这正是 a 所包含的每个一维数组的类型。

二维数组指针

int a[3][4] = { {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11} };



int *p[4]

p 就成了一个指针数组，而不是二维数组指针

[ ]的优先级高于*，( )是必须要加的

二维数组指针

int *(p1[5]); //指针数组，可以去掉括号直接写作 int *p1[5];
int (*p2)[5]; //二维数组指针，不能去掉括号

a+i == p+i
a[i] == p[i] == *(a+i) == *(p+i)
a[i][j] == p[i][j] == *(a[i]+j) == *(p[i]+j) == *(*(a+i)+j) == *(*(p+i)+j)

继续挑战指针

```
int *p1[6]; //指针数组
int *(p2[6]); //指针数组，和上面的形式等价
int (*p3)[6]; //二维数组指针
int (*p4)(int, int); //函数指针
```

继续挑战指针

```
int *p1[6]; //指针数组
int *(p2[6]); //指针数组，和上面的形式等价
int (*p3)[6]; //二维数组指针
int (*p4)(int, int); //函数指针
```

```
char *(* c[10])(int **p);
int (*(*(*pfunc)(int *))[5])(int *);
```

再复习下运算符

优先级	运算符	结合律
从高到低排列	() [] -> .	从左至右
	! ~ ++ -- （类型） sizeof + - * &	从右至左
	* / %	从左至右
	+ -	从左至右
	<< >>	从左至右
	< <= > >=	从左至右
	== !=	从左至右
	&	从左至右
	^	从左至右
	|	从左至右
	&&	从左至右
	||	从右至左
	?:	从右至左
	= += -= *= /= %= &= ^= |= <<= >>=	从左至右

- 定义中被括号( )括起来的那部分。
- 后缀操作符：括号( )表示这是一个函数，方括号[ ]表示这是一个数组。
- 前缀操作符：星号*表示“指向xxx的指针”。

继续挑战指针

```
int *p1[6]; //指针数组
int *(p2[6]); //指针数组，和上面的形式等价
int (*p3)[6]; //二维数组指针
int (*p4)(int, int); //函数指针
```

```
char *(* c[10])(int **p);
                char *func(int **p);

int (*(*(*pfunc)(int *))[5])(int *);
                int func(int *);
```