X 语言程序设计

本节内容

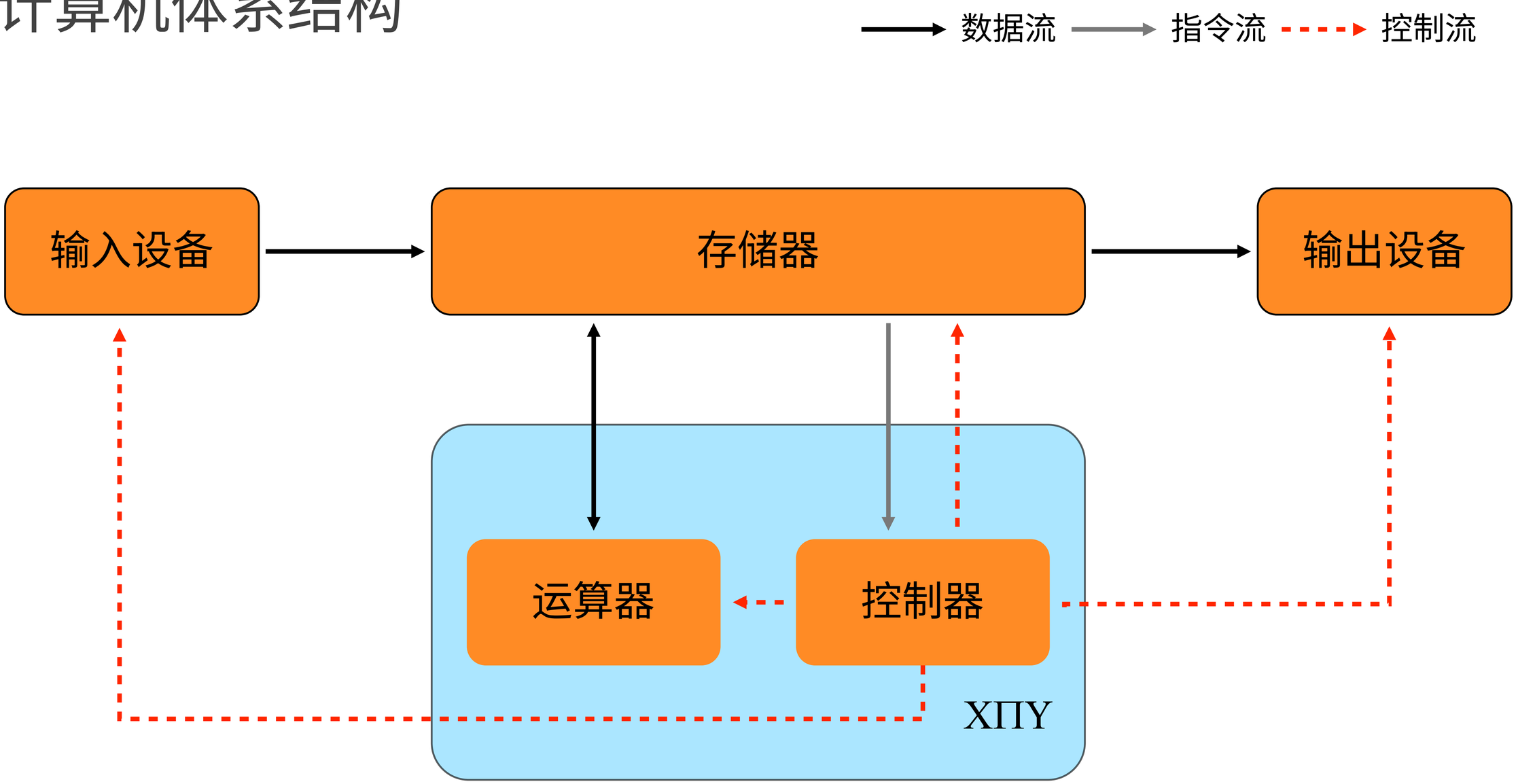
内存地址概述

指针定义与使用

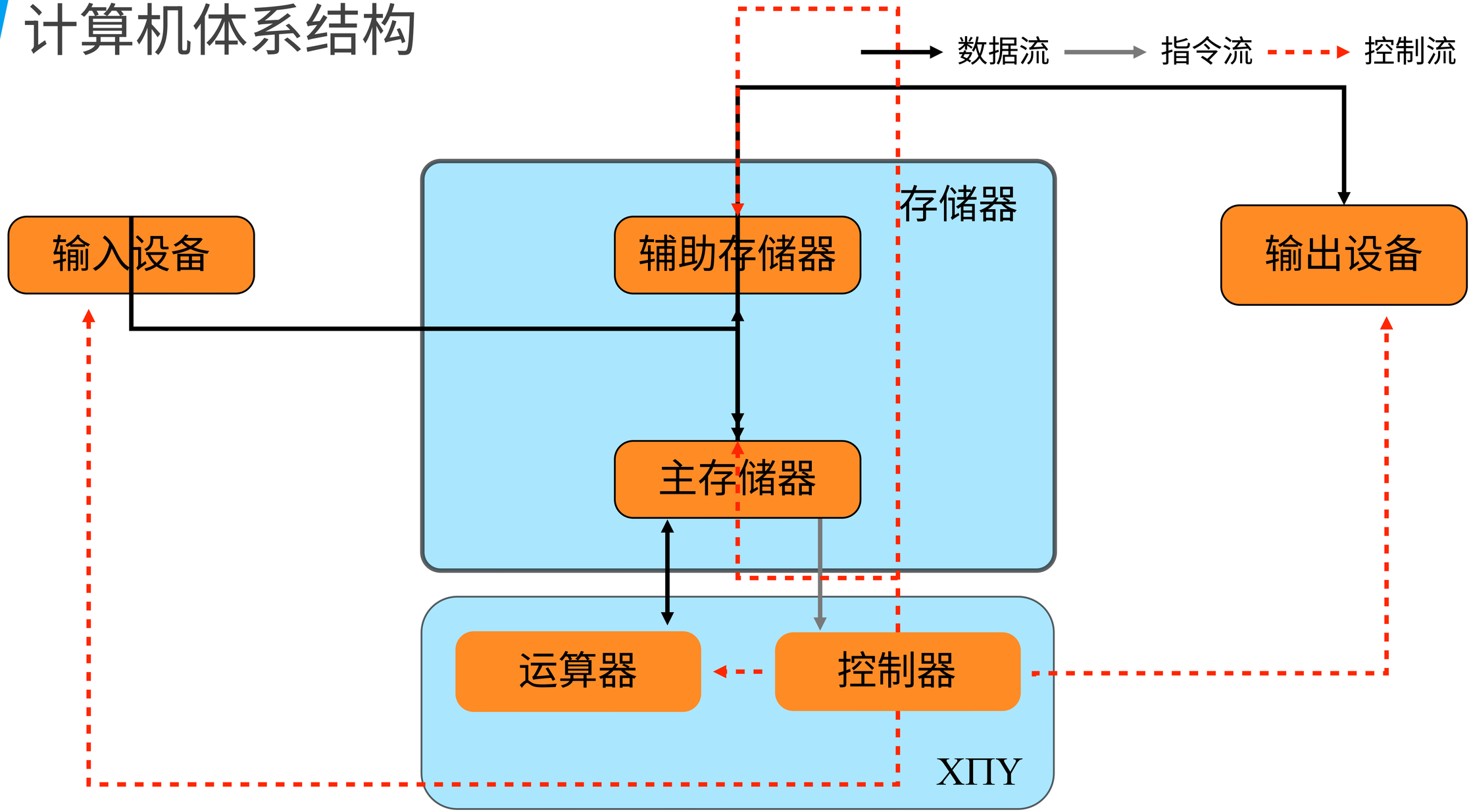
指针的应用

字符串处理函数

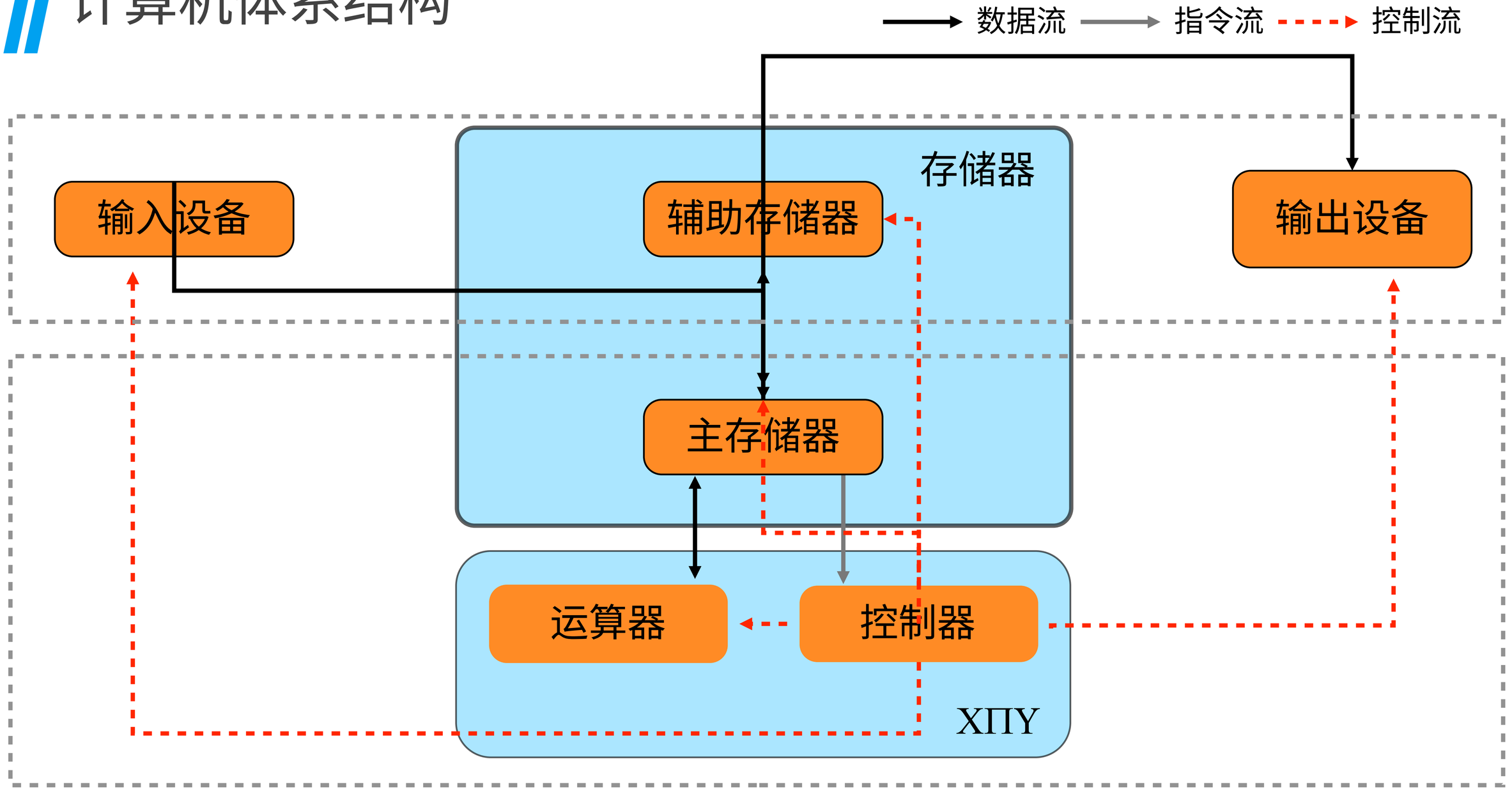
计算机体系结构



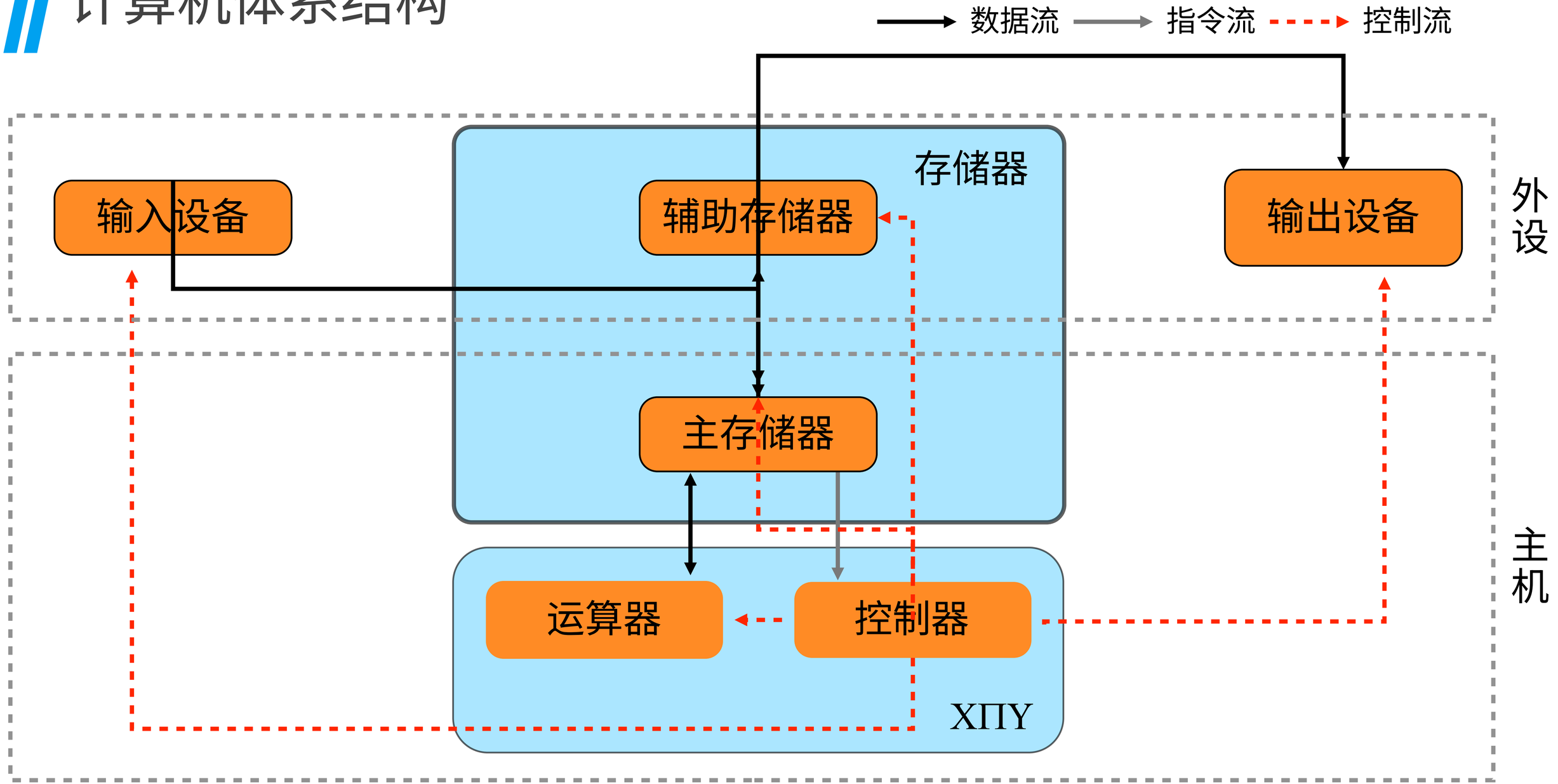
计算机体系结构



计算机体系结构



计算机体系结构

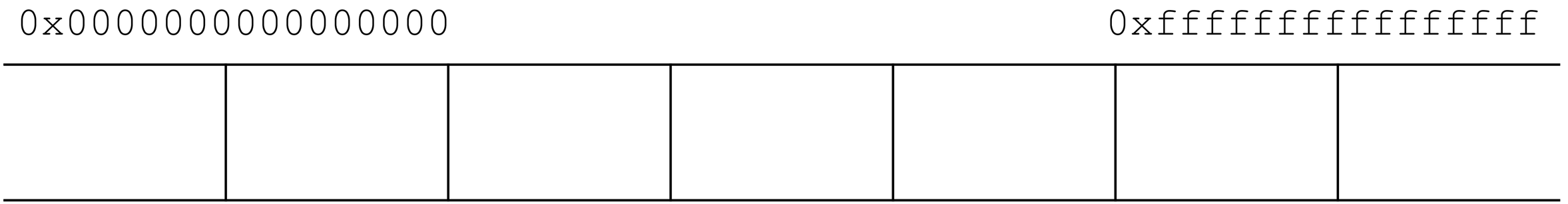


虚拟内存地址

内存条、显卡、各种适配卡都有其各自的存储地址空间。

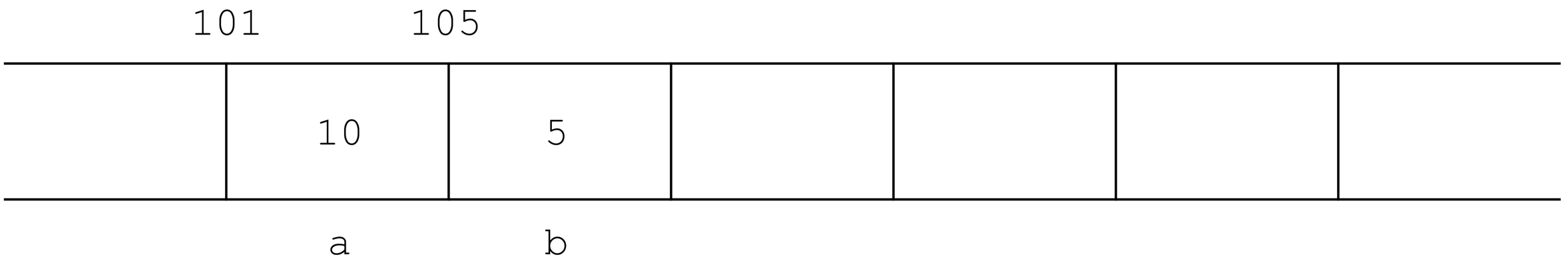
操作系统将这些设备的存储地址空间抽象成一个巨大的一维数组空间。

对于内存的每一个字节会分配一个 32 位或 64 位的编号，这个编号称为内存地址。



虚拟内存地址

```
int main()
{
    int a = 10;
    int b = 5;
    return 1;
}
```

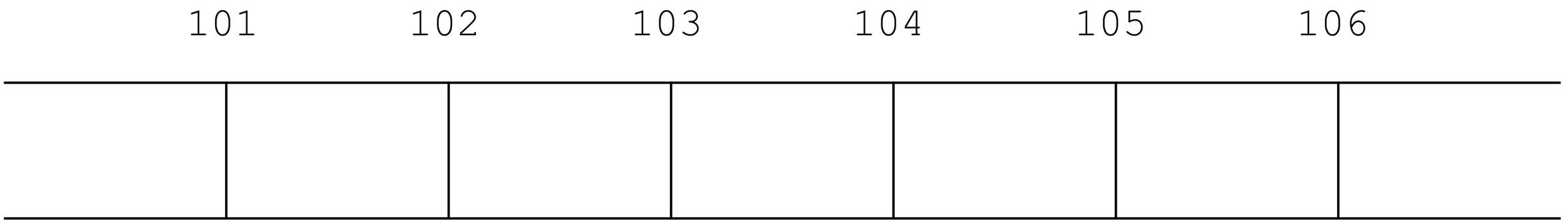


X 程序员新手和老手的其中之一差别在于是否对指针有深刻的理解、高效的使用指针。

理解指针的关键在于理解 X 程序如何管理内存。

指针的定义

指针是用来存放内存地址的变量。

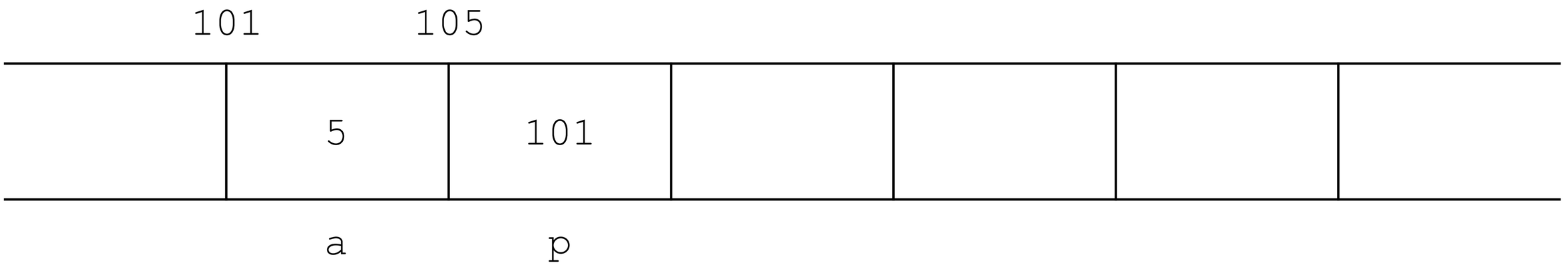


指针的声明

```
int a;    // 声明一个整型变量
int *p;   // 声明一个指针变量，该指针可以存放一个 int 类型值的内存地址
// 星号  *  两边的空格无关紧要，下面的声明是等价的
int* p;
int *p;
int * p;
int*p;
```

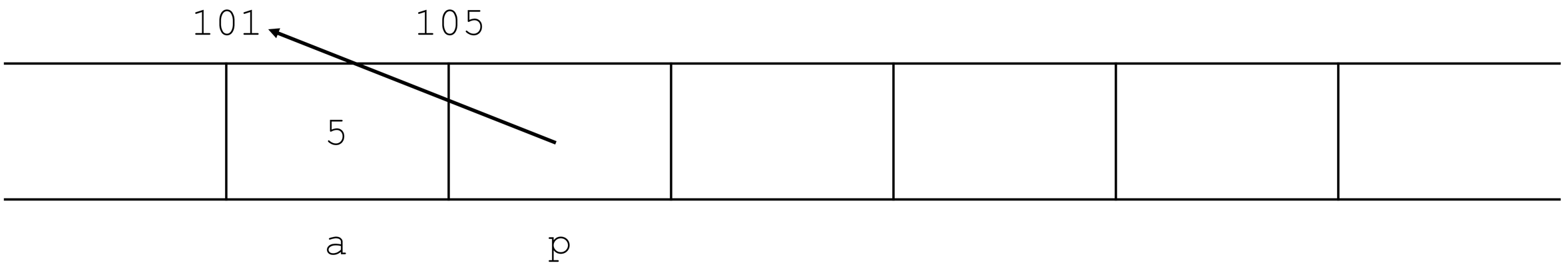
指针的使用

```
int a;
int *p;
a = 5;
p = &a;
```



指针的使用

```
int a;
int *p;
a = 5;
p = &a;
```



间接引用操作符

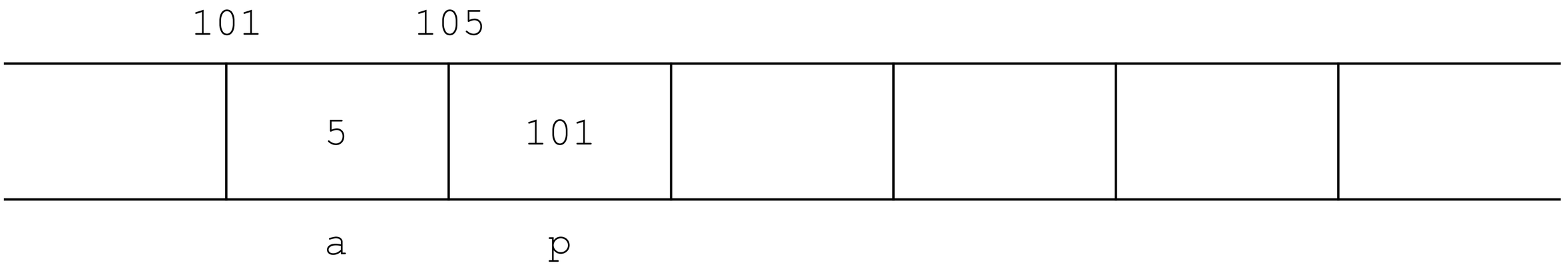
*

间接引用操作符 * 返回指针变量的指向地址的值

通常把这个操作叫做“解引用指针”

英文叫做 δερεφερενχινγ α ποιντερ

```
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```



间接引用操作符

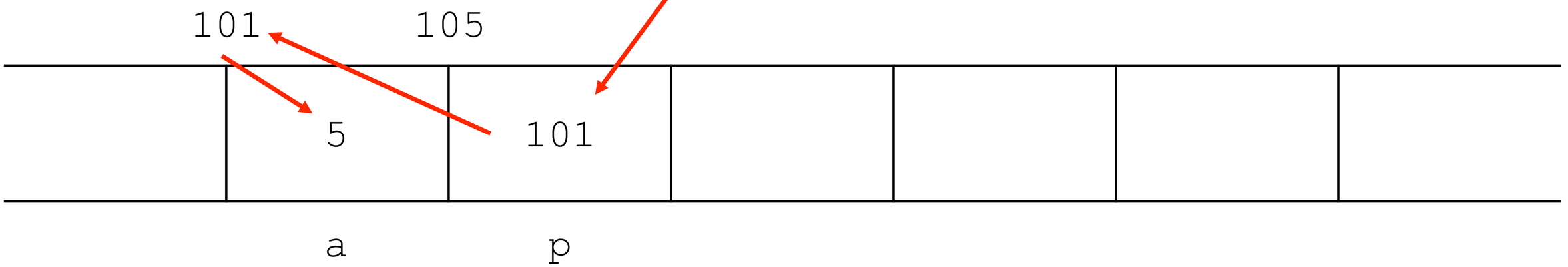
*

间接引用操作符 * 返回指针变量的指向地址的值

通常把这个操作叫做“解引用指针”

英文叫做 dereferencing a pointer

```
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```



间接引用操作符

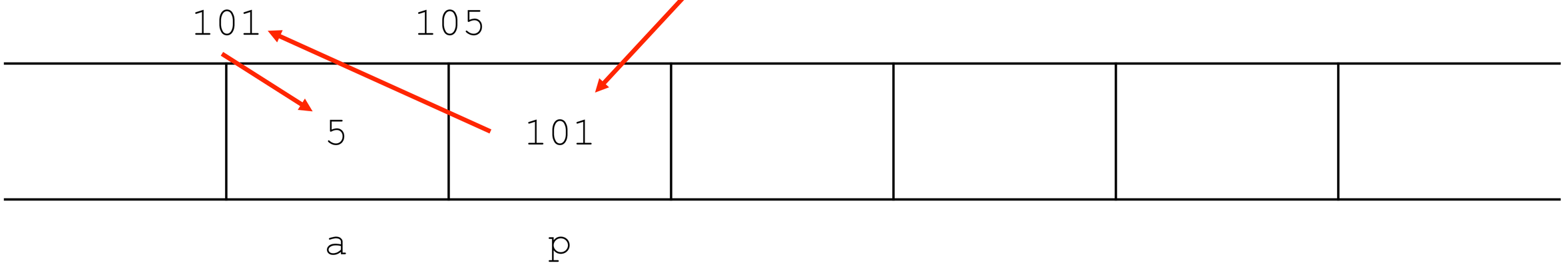
*

间接引用操作符 * 返回指针变量的指向地址的值

通常把这个操作叫做“解引用指针”

英文叫做 δερεφερενχινγ α ποιντερ

```
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```



间接引用操作符

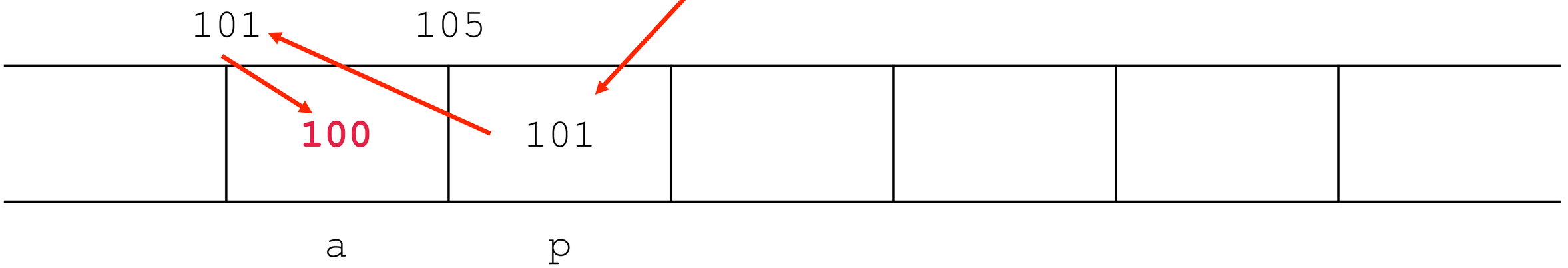
*

间接引用操作符 * 返回指针变量的指向地址的值

通常把这个操作叫做“解引用指针”

英文叫做 dereferencing a pointer

```
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```



间接引用操作符

*

间接引用操作符 * 返回指针变量的指向地址的值

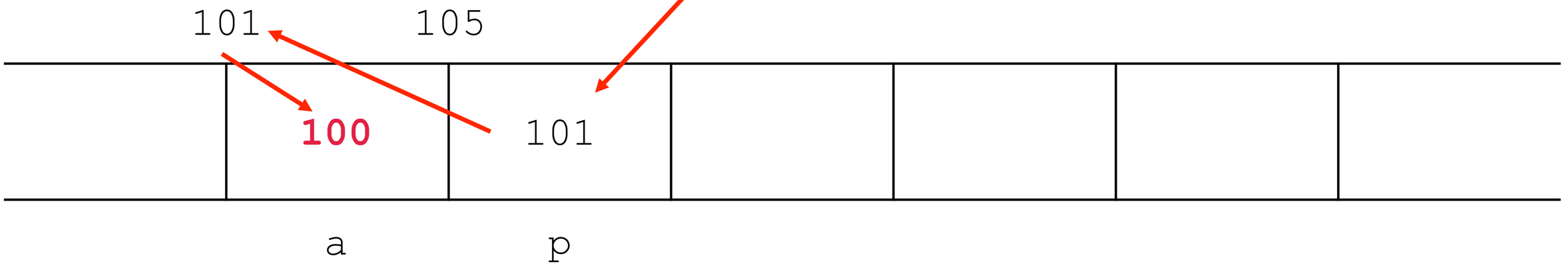
通常把这个操作叫做“解引用指针”

英文叫做 δερεφερενχινγ α ποιντερ

```
int a = 5;
int *p = &a;
printf("%d\n", *p);
*p = 100;
printf("%d\n", a)
```

从指定的地址读取

存储到指定的地址



指针的大小

使用 σιζεοφ() 获取指针的大小，得到的总是 4 或 8

σιζεοφ() 获取的是指针变量指向存储地址的大小

在 32 位平台，所有的指针地址都是 32 位（4 字节）

在 64 位平台，所有的指针地址都是 64 位（8 字节）

野指针与空指针

随便将任意数值赋值给一个指针变量，该指针就成了野指针，这样的赋值没有意义

从语法规则角度，野指针不会直接报错，只有在解引用野指针时才会出现问题

将 NYΛΛ 赋值给一个指针变量，该指针变量可以被称为空指针

指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

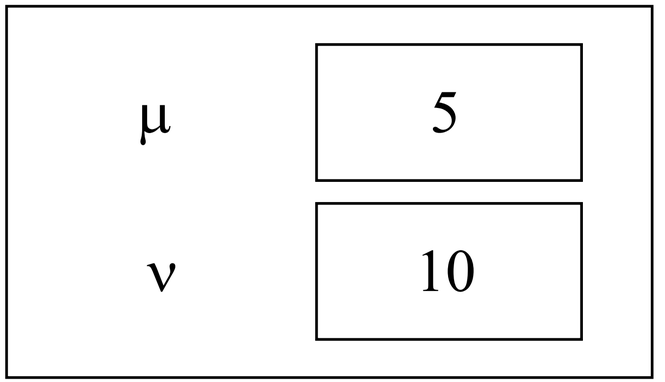
指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```c
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

μαιν



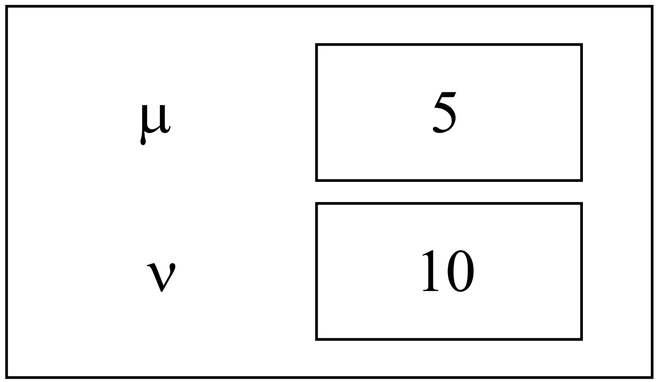
指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

μαιν



指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```c
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

μαιν

μ	5
ν	10

σωαπ

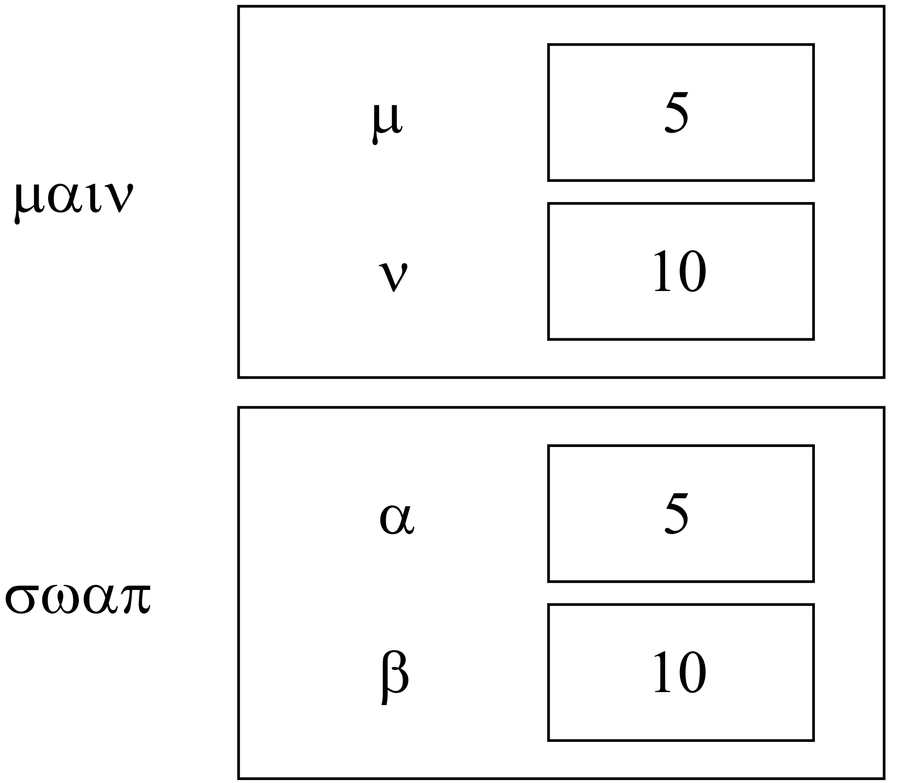
α	5
β	10

指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```c
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

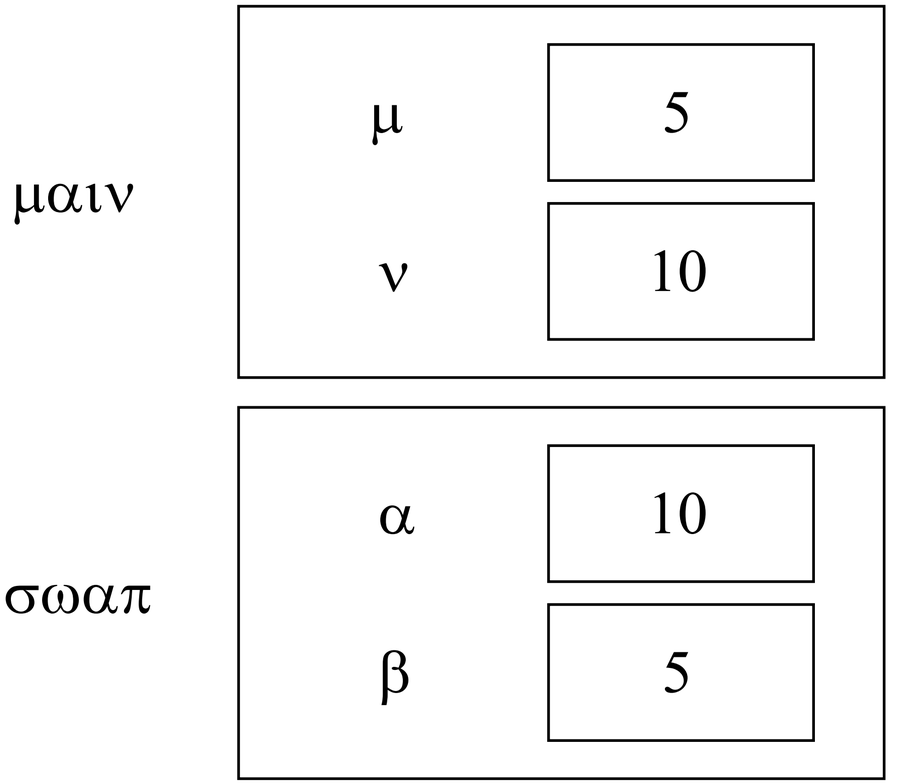


指针与函数

设计一个函数，传入两个 int 参数，并交换这两个参数的值。

```c
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("a=%d, b=%d\n", a,b);
}

int main(int argc, char const *argv[])
{
    int m = 5;
    int n = 10;
    swap(m, n);
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```



指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

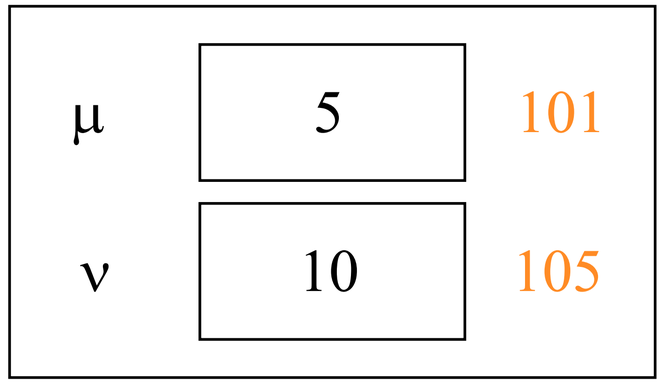
```
 3  void swap(int *a, int *b)
 4  {
 5      int temp;
 6      temp = *a;
 7      *a = *b;
 8      *b = temp;
 9      printf("a=%d, b=%d\n", *a, *b);
10  }
11
12  int main(int argc, char const *argv[])
13  {
14      int m = 5;
15      int n = 10;
16      swap(&m, &n);
17      printf("m=%d, n=%d\n", m, n);
18      return 0;
19  }
```

指针与函数

设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```
3   void swap(int *a, int *b)
4   {
5       int temp;
6       temp = *a;
7       *a = *b;
8       *b = temp;
9       printf("a=%d, b=%d\n", *a, *b);
10  }
11
12  int main(int argc, char const *argv[])
13  {
14      int m = 5;
15      int n = 10;
16      swap(&m, &n);
17      printf("m=%d, n=%d\n", m, n);
18      return 0;
19  }
```

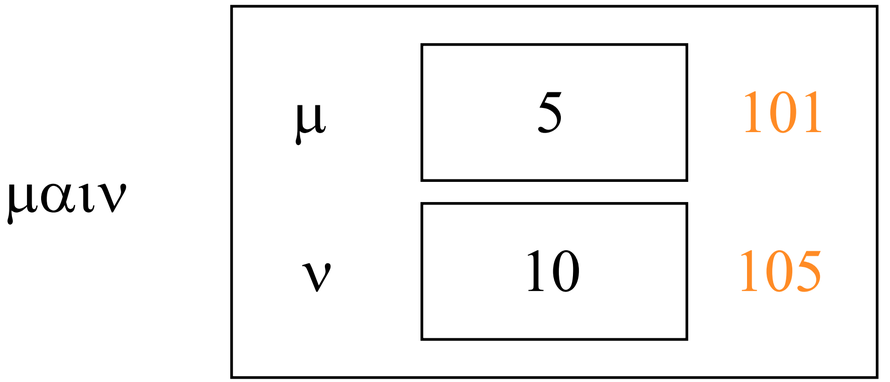
μαιν



指针与函数

设计一个函数，传入两个 int 参数，并交换这两个参数的值。

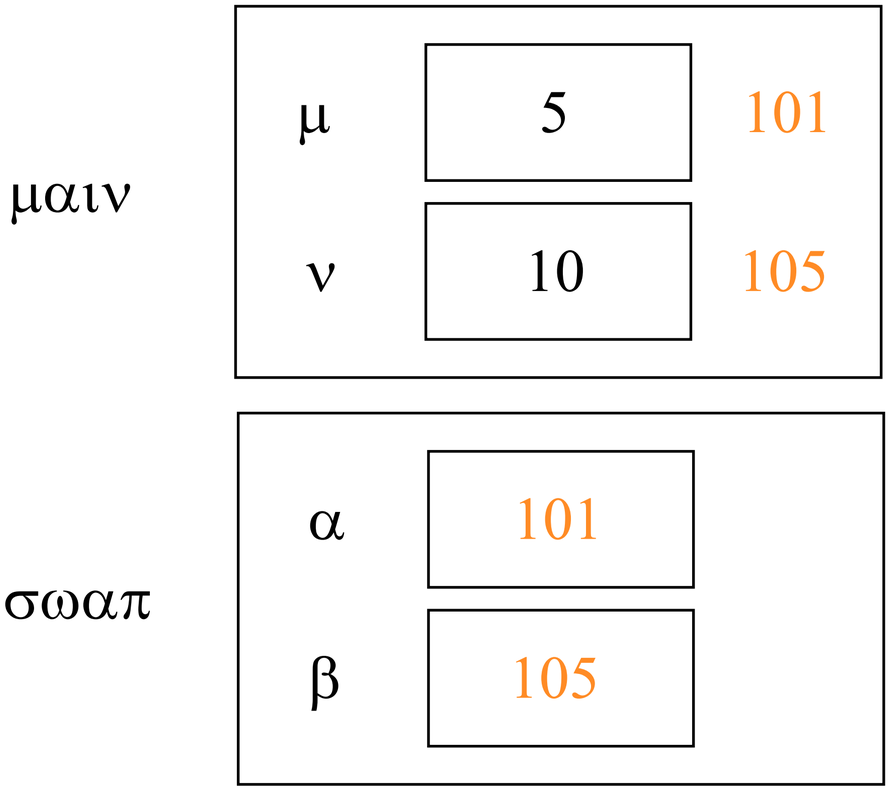
```
3  void swap(int *a, int *b)
4  {
5      int temp;
6      temp = *a;
7      *a = *b;
8      *b = temp;
9      printf("a=%d, b=%d\n", *a, *b);
10 }
11
12 int main(int argc, char const *argv[])
13 {
14     int m = 5;
15     int n = 10;
16     swap(&m, &n);
17     printf("m=%d, n=%d\n", m, n);
18     return 0;
19 }
```



指针与函数

设计一个函数，传入两个 int 参数，并交换这两个参数的值。

```
3  void swap(int *a, int *b)
4  {
5      int temp;
6      temp = *a;
7      *a = *b;
8      *b = temp;
9      printf("a=%d, b=%d\n", *a, *b);
10 }
11
12 int main(int argc, char const *argv[])
13 {
14     int m = 5;
15     int n = 10;
16     swap(&m, &n);
17     printf("m=%d, n=%d\n", m, n);
18     return 0;
19 }
```

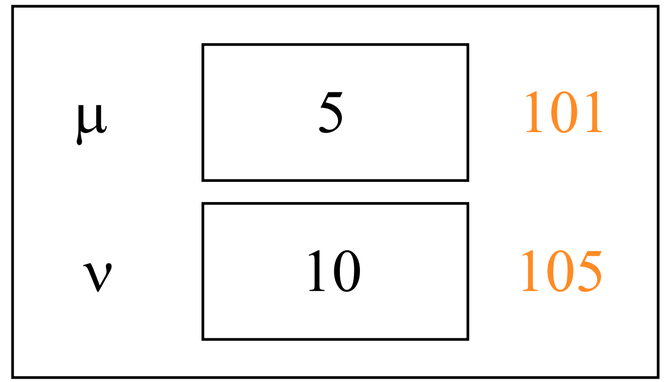


指针与函数

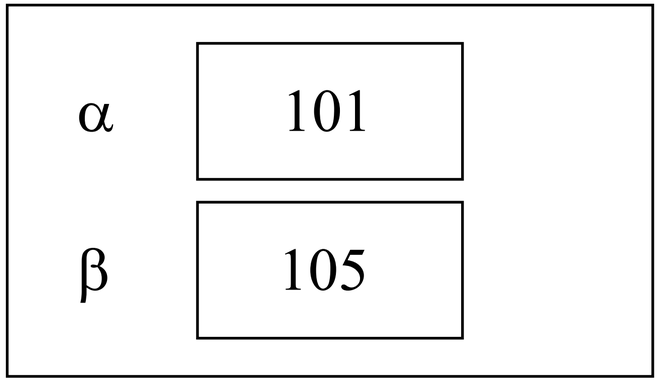
设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```
3  void swap(int *a, int *b)
4  {
5      int temp;
6      temp = *a;
7      *a = *b;
8      *b = temp;
9      printf("a=%d, b=%d\n", *a, *b);
10 }
11
12 int main(int argc, char const *argv[])
13 {
14     int m = 5;
15     int n = 10;
16     swap(&m, &n);
17     printf("m=%d, n=%d\n", m, n);
18     return 0;
19 }
```

μαιν



σωαπ

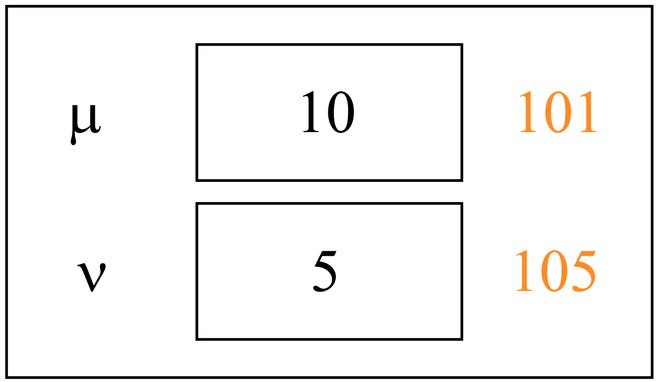


指针与函数

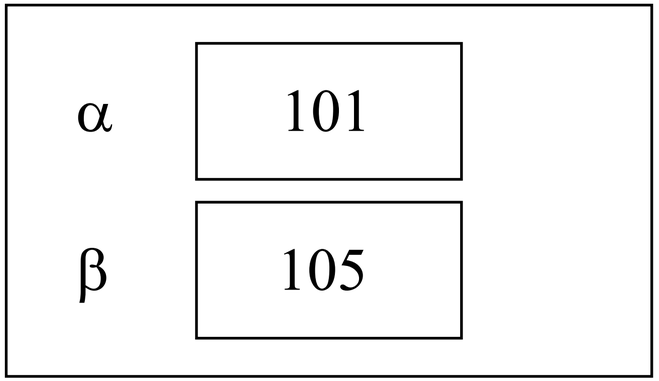
设计一个函数，传入两个 ιντ 参数，并交换这两个参数的值。

```
3   void swap(int *a, int *b)
4   {
5       int temp;
6       temp = *a;
7       *a = *b;
8       *b = temp;
9       printf("a=%d, b=%d\n", *a, *b);
10  }
11
12  int main(int argc, char const *argv[])
13  {
14      int m = 5;
15      int n = 10;
16      swap(&m, &n);
17      printf("m=%d, n=%d\n", m, n);
18      return 0;
19  }
```

μαιν



σωαπ



指针与数组

直接调用数组名，相当于获取数组的首地址

```
int a[] = {1, 2, 3, 4}
printf(“%p\n”, a);
printf(“%p\n”, &a[0]);
```

指针与数组

在 X 语言中，指针与数组的关系十分密切。

通过数组下标能完成的操作都可以通过指针完成。

一般来说，用指针编写的程序比用数组下标编写的程序执行速度快。

指针与数组

给指针加上一个整数，实际上加的是这个整数和指针数据类型对应字节数的乘积。

```c
int a = 5;
int *p = &a;
printf("%p\n", p);
p++;
printf("%p\n", p);
return 0;
```

p++ ⟶ p = p + 1 ⟶ p = p + 1 × 4

指针与数组

有符号数	无符号数	32 位	64 位
χηαρ	υνσιγνεδ χηαρ	1	1
σηορτ	υνσιγνεδ σηορτ	2	2
ιντ	υνσιγνεδ ιντ	4	4
λονγ	υνσιγνεδ λονγ	4	8
φλοατ		4	8
δουβλε		8	8

指针与字符串

```
char[] str = "HelloWorld";
char* p = str;
```

```
char* p2 = "HelloKitty";
```

字符串处理函数－获取字符串长度

```
size_t strlen(char* str)
```

返回 στρ 长
度

字符串处理函数－追加

```
char* strcat(char *dest, const char *src)
```

将 σρχ 连接在 δεστ 的后面，返回 δεστ 的首地址

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

将 σρχ 前 ν 个字符连接在 δεστ 的后面，返回 δεστ 的首地址

字符串处理函数—拷贝

```
char* strcpy(char *dest, const char *src)
```

将 σρχ 拷贝并替换到 δεστ，返回 δεστ 的首地址

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

将 σρχ 中的前 ν 个字符拷贝到 δεστ，返回 δεστ 的首地址

字符串处理函数—比较

```
int strcmp(cont char *s1, const char *s2)
```

比较 σ1 和 σ2 的大小，相等返回 0 ，大于返回大于 0 的数，小于返回小于 0 的数。

字符串处理函数－查找

```
char* strchr(cont char *s, int c)
```

返回字符 χ 在字符串 σ 中第一次出现位置的地址

```
char* strstr(cont char *haystack, const char *needle)
```

返回字符串 $\nu\varepsilon\varepsilon\delta\lambda\varepsilon$ 在字符串 $\eta\alpha\psi\sigma\tau\alpha\chi\kappa$ 中第一次出现的位置