# 目录

[目录 1](#_Toc520736752)

[第6章 C语言-指针 2](#_Toc520736753)

[6.1 C语言指针概述 2](#_Toc520736754)

[6.1.1 何谓指针 2](#_Toc520736755)

[6.1.2 何谓地址 2](#_Toc520736756)

[6.2 指针的定义与初始化 2](#_Toc520736757)

[6.3 指针的使用 3](#_Toc520736758)

[6.3.1 指针与变量关系 3](#_Toc520736759)

[6.3.2 指针与数组关系 3](#_Toc520736760)

[6.3.3 指针的运算 4](#_Toc520736761)

[6.4 函数的形参列表为指针类型 7](#_Toc520736762)

[6.5 const的用法 8](#_Toc520736763)

[6.6 指针指向二维数组 10](#_Toc520736764)

[6.7 数组指针 11](#_Toc520736765)

[6.7.1 何谓数组指针 11](#_Toc520736766)

[6.7.2 数组指针的形式 11](#_Toc520736767)

[6.7.3 数组指针如何使用 11](#_Toc520736768)

[6.8 指针数组 13](#_Toc520736769)

[6.8.1 指针数组的使用 13](#_Toc520736770)

[6.9 函数指针 14](#_Toc520736771)

[6.9.1 函数指针定义 14](#_Toc520736772)

[6.9.2 函数指针的使用 15](#_Toc520736773)

[6.9.3 回调函数 15](#_Toc520736774)

[6.10 指针函数 17](#_Toc520736775)

# C语言-指针

## C语言指针概述

指针就是地址

1. 在计算机内存中，系统会为每一个字节进行编号，这个以十六进制的形式表示，固定不变，并且独一无二，这些编号的长度是一样，这个编号就是地址 🡪 指针
2. 对变量的两种访问方式
3. 直接访问：通过变量名来操作空间，此时操作变量名就是操作该空间
4. 间接访问：通过地址来对空间进行访问，从实现对空间的操作

int a = 0, int b = 0;

a = 10; //写

b = a + 10;//读

操作空间的方法有两种：直接操作和间接操作

1. 指针的分类
2. 指针常量：指向不可以改变的指针， 数组名、函数名
3. 指针变量：指向可以改变的指针，存储地址；

## 指针变量的定义与初始化

1. 定义

整型：int

整型定义一个变量：整型 变量名； int a；定义了一个整型变量a

int a = 0;

p = &a;

int\* p; \*p = 10;

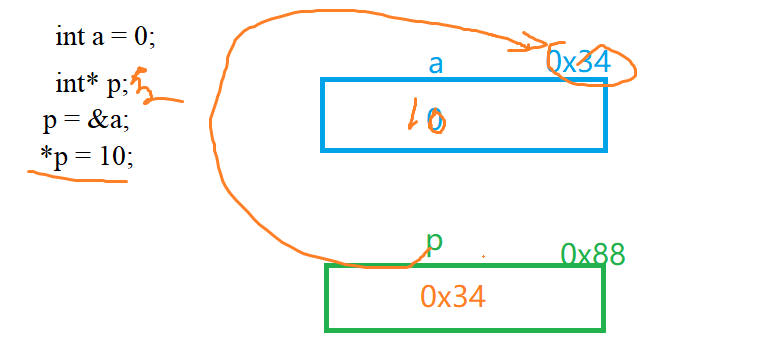
指针类型：数据类型 \* 地址类型

数据类型： (指针的指向对象空间的偏移量)

\*：除了定义时表示是指针类型外，其它都表示取地址对象的空间

指针定义一个变量：指针类型 变量名；数据类型 \*指针名；

指针名：标识符的命名规则，具有含义，一般用p开头



举例

int \*p；表示定义了一个int \*类型的变量p；指针指向的对象的空间类型是int类型

char\*p；表示定义了一个char \*类型的变量p；指针指向的对象的空间类型是char类型

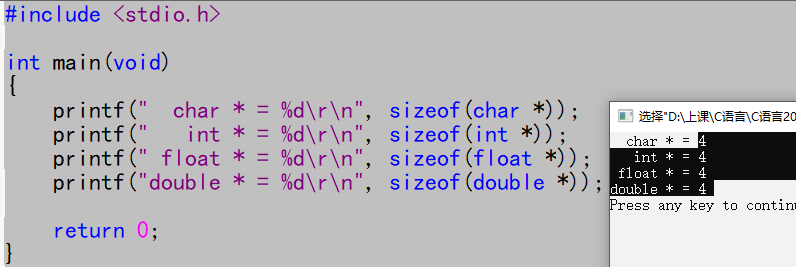
float\*p；表示定义了一个float \*类型的变量p；指针指向的对象的空间类型是float类型

1. 初始化

普通变量：int a = 0;

指针变量：int \*p = NULL;//空地址

1. 注意
2. 每一个指针变量(存地址，地址4字节)所占空间大小都是4字节；（面试、考试）



1. 变量指针：变量的地址

指针变量：存储地址的变量

1. 指针的数据类型要与访问空间的类型要一致，否则会报警告，但是有些就要这样用

int a[5];

int \*p = a;

## 指针变量的使用

### 指针变量与普通变量关系

1. 指针变量与普通变量

初始化：数据类型 \*指针名 = &变量名 int a; int \*p = &a;

赋值：数据类型 \*指针名；指针名 = &变量名； int a; int \*p; p = &a;

1. 注意
2. \*指针名：访问指针指向对象的空间 \*p = 10;
3. \*指针名放在“=”左边，对指针指向对象空间进行写操作 \*p = 10;
4. \*指针名放在“=”右边，对指针指向对象空间进行读操作

### 指针变量与一维数组关系

1. 指针变量与一维数组建立关系

初始化：数据类型 \*指针名 = 数组名； int a[5]; int \*p = a;

赋值：数据类型 \*指针名； 指针名 = 数组名； int a[5]; int \*p; p = a;

1. 注意
2. 当指针变量指向数组首元素地址时，把指针名当成数组名来用(空间的类型和指针的数据类型相同时)
3. 数组名是一个指针常量，指向不能改变 a = 0x34 a+1
4. 指针变量的偏移的字节数，由指针变量定义时的数据类型来决定
5. 指针变量 🡪 变量 数组名 🡪 常量

### 指针变量的运算

+、-：地址会发生偏移，但是指针变量里面地址不变 p+1

++、--：地址会发生偏移，指针变量里面的地址会改变 p++；

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a[4] = {1, 2, 3, 4};  int \*p = a;  int i;    printf("p = %p\r\n", p);  printf("a = %p\r\n", a);  for(i = 0; i < 4; i++)  {  printf("%d ", \*p++);  }  printf("\r\n");  printf("p = %p\r\n", p);  return 0;  } |

课堂练习

|  |
| --- |
| #include <stdio.h>  int main(void)  {  unsigned char a,b,c,d,f,g,x,y,z;  unsigned char ch[16] = {9, 5, 1, 2, 5,300, 4, 151}; ch:unsigned char \*  unsigned char \*sp,\*ss;  unsigned int str[3] = {3, 7, 9}; str: unsigned int \*  sp = ch;  ss = str; //unsigned int \* to unsigned char \*  a = \*sp;  c = \*sp++;  d = \*(sp + 1);  f = \*sp;  g = \*(sp++);  x = \*(sp + sizeof(sp));  y = \*(sp - sizeof(\*sp));  z = \*(sp + sizeof(ss));    printf("a = %d\r\n", a);  printf("b = %d\r\n", b);  printf("c = %d\r\n", c);  printf("d = %d\r\n", d);  printf("f = %d\r\n", f);  printf("g = %d\r\n", g);  printf("x = %d\r\n", x);  printf("y = %d\r\n", y);  printf("z = %d\r\n", z);  return 0;  } |

## 函数的形参列表为指针类型

实参和形参的关系

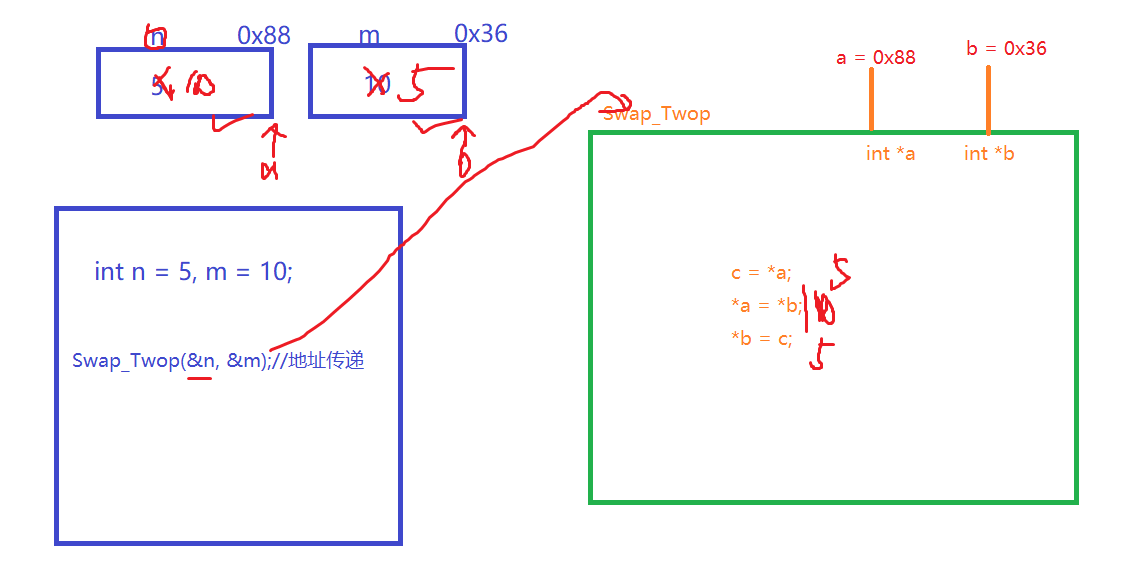
将实参赋值给形参的的过程

两种传递方式？

1. 值传递
2. 地址传递

改变形参的值会影响实参

此时操作形参就是在操作实参



## const的用法

const：只读关键字，修饰变量为只读变量，不能进行写操作只能进行读操作

const int a = 15;

作用：保护该变量不被后面程序代码改变

注意：用const修饰变量时，必须要进行初始化

1. 修饰普通变量:可以将const放在数据类型的前面，也可以放在数据类型的后面

const int a = 0;

int const a = 0;

作用都是修饰变量a，将a变量变成只读变量

|  |
| --- |
| int main(void)  {  int const a = 10;//修饰变量a为只读变量 只可以进行读操作，不能进行写操作  int b = 15;  a = 100;//不能进行写操作  // b = a;//可以进行写操作  printf("a = %d\r\n", a);  printf("b = %d\r\n", b);  return 0;  } |

1. 修饰指针变量：修饰谁，谁就为只读
2. const int \*p; //不看基本数据类型 修饰\*p

int const \*p; //修饰\*p

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a = 10;  int b = 5;  const int \*p = &a; //修饰\*p  // \*p = 10;//此时不能对指针指向对象空间进行写操作  p = &b; //指针的指向可以改变  return 0;  } |

1. int \* const p; //修饰p

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a = 10;  int b = 5;  int \* const p = &a; //修饰p  \*p = 20;//可以对指针指向对象空间进行写操作  // p = &b; //指针的指向不可以改变  return 0;  } |

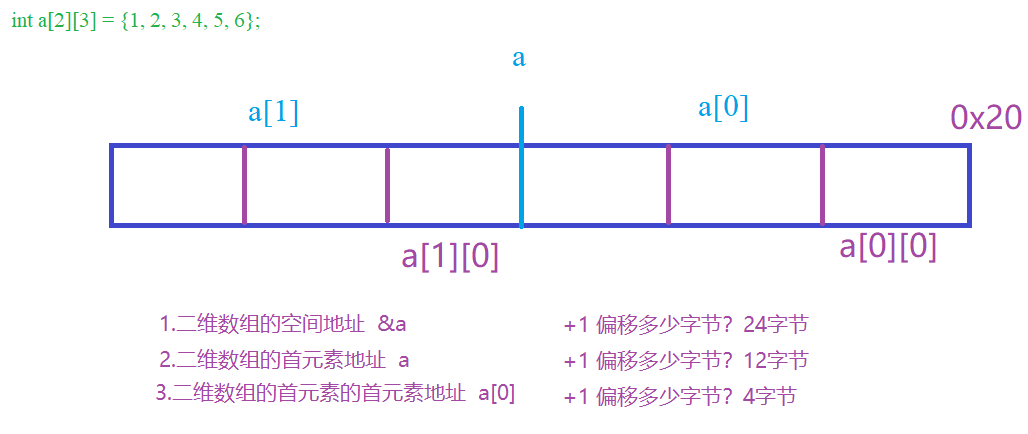
1. const int \* const p; //修饰\*p、p

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a = 10;  int b = 5;  const int \* const p = &a; //修饰\*p、p  // \*p = 20;//不可以对指针指向对象空间进行写操作  // p = &b; //指针的指向不可以改变  return 0;  } |

## 指针指向二维数组

二维数组的三地址 数组数组

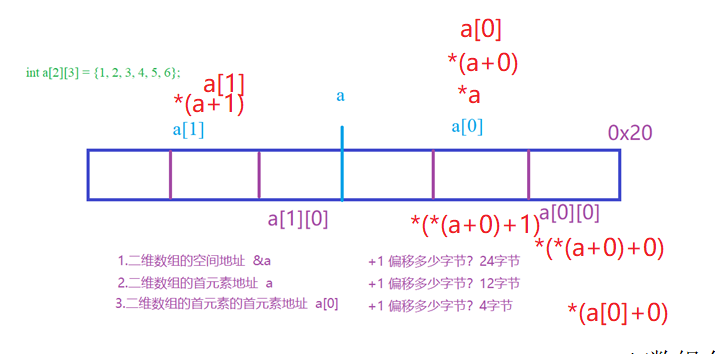
int a[2][3] = {1, 2, 3, 4, 5, 6}; 二维数组a有2个元素a[0] 、a[1] ，在a[0]、a[1]元素里面又有3个元素

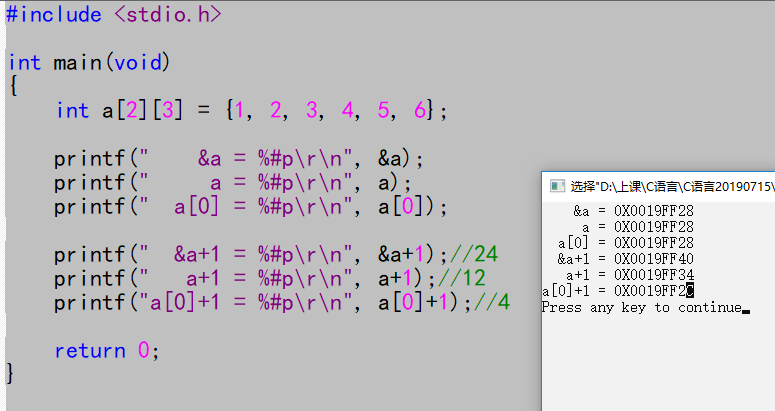


数组名[行下标][列下标]；

\*(数组名[行下标]+ 列下标)；

\*(\*(数组名+行下标)+列下标)；





将主函数的二维数组传递到子函数时

|  |
| --- |
| #include <stdio.h>  void My\_Printf(int a[][3], int h, int l);  int main(void)  {  int a[2][3] = {1, 2, 3, 4, 5, 6};  My\_Printf(a, 2, 3);  return 0;  }  void My\_Printf(int a[][3], int h, int l)//int a[][3] int \*a int (\*a)[3]  {  int i, j;    for(i = 0; i < h; i++)  {  for(j = 0; j < l; j++)  {  printf("%d ", a[i][j]);  }  printf("\r\n");  }  } |

## 数组指针

1. 指针的三要素

int a = 10;

int \*p;

p = &a;

1. 指针类型：int \* 决定访问一次的字节数 去掉指针名，剩下就是指针类型
2. 指针指向的对象：int 表示指针指向的对象空间的类型int类型 去掉\*指针名，剩下的就是指针指向的对象类型
3. 指针变量的值：p 存另一个内存的地址

在定义一个变量的时候，同时出现\*、()、[]，标识符会根据运算符的优先级和结合性来决定跟谁结合

int (\*p)[5]：标识符和\*结合，数组指针：本质指针 指向对象的空间类型是数组类型的指针

int \* p[5]：标识符和[]结合，指针数组：数组 元素类型是指针类型的数组

int a[5]; 定义了元素个数为5，元素类型是int类型的空间 数据类型：元素类型

int \*p[5];定义了元素个数为5，元素类型是int \*类型的空间

1. 数组指针：偏移量自己决定

本质：指针 指向对象的空间类型是数组类型的指针

格式：数据类型 [长度] (\*指针名)；int [5] (\*p) p+1:20字节

长度：这个表示指向的数组类型里面的元素个数

指向二维数组时，这个长度跟列长度 🡪 行指针

int a[3][4];

int (\*p)[4];

1. 当数组指针指向一维数组时，会把一维数组当成二维数组来进访问，指针名就是数组名

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};  int (\*p)[3] = a;    printf("a = %#p\r\n", a);  printf("p = %#p\r\n", p);  printf("a+1 = %#p\r\n", a+1);  printf("p+1 = %#p\r\n", p+1);//12  printf("a[0] = %d\r\n", a[0]);  printf("p[0][0] = %d\r\n", p[0][0]);  printf("a[1] = %d\r\n", a[1]);  printf("p[1][0] = %d\r\n", p[1][0]);  return 0;  } |

1. 数组指针指向二维数组时，把指针名当成数组名来使用(数组指针里面的长度要和二维数组的列长度一样)

示例：

|  |
| --- |
| int main(void)  {  int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};  int (\*p)[3];  p = a;  printf(" \*\*p = %d\r\n", \*\*p);  printf(" \*p[0] = %d\r\n", \*p[0]);  printf(" (\*p)[1] = %d\r\n", (\*p)[1]);  printf(" \*p[1] = %d\r\n", \*p[1]);  printf("(\*++p)[0] = %d\r\n", (\*++p)[0]);  printf("\*(\*++p) = %d\r\n", \*(\*++p));  printf("\*((\*p)+2) = %d\r\n", \*((\*p)+2));  return 0;  } |

## 指针数组

int a[5];//定义了一个元素个数为5，元素类型为int类型的空间

int \*a[5];//定义了一个元素个数为5，元素类型为int \*类型的空间

本质：数组 元素类型是指针类型 多个指针变量组成

格式：数据类型 \*数组名[长度]；

元素类型：数据类型 \* 🡪 指针类型

长度：表示里面元素的个数，有多少个指针变量

大小：长度\*4

应用：指向字符串

示例：

|  |
| --- |
| #include <stdio.h>  int main(void)  {  /\* int a = 5, b = 10, c = 20;  int \*p[3] = {&a, &b, &c};  printf("a = %d\r\n", \*p[0]);  printf("b = %d\r\n", \*p[1]);  printf("c = %d\r\n", \*p[2]);\*/  char a[3][20] = {"dsfdsf", "1234556", "jklkjhk"};  char \*p[3] = {a[0], a[1], a[2]};  int i;  for(i = 0; i < 3; i++)  {  puts(p[i]);  }  return 0;  } |

## 函数指针

本质：指针 指向的对象的空间类型是函数类型的指针

格式：返回值类型 (\*指针名)(形参列表)；int (\*p)(int, int);

形参列表：形参列表可以为空，也可以不为空，如果不为空，写形参类型即可，不写形参名

int (\*p)(int, int); //函数指针 可以更加灵活的访问函数

使用方式：

指针名(实参)；//函数名的用法一样

(\*指针名)(实参)；

示例：

|  |
| --- |
| #include <stdio.h>  int My\_Mul(int a, int b);  int My\_Sub(int a, int b);  int My\_Add(int a, int b);  int My\_Bb(int a);  int main(void)  {  //函数指针  int (\*p)();//不写形参类型个数，使用更加通用 写了形参类型个数，方便查看传入的形参个数  // p = My\_Add;  // p = My\_Sub;  p = My\_Bb;  printf("%d\r\n", p(10));  return 0;  }  int My\_Add(int a, int b)  {  return a+b;  }  int My\_Sub(int a, int b)  {  return a-b;  }  int My\_Mul(int a, int b)  {  return a\*b;  }  int My\_Bb(int a)  {  return a\*a;  } |

回调函数：当函数指针作为函数的形参时，这个函数就叫回调函数

作用：使这个函数的功能多样化，这个回调函数的功能取决于传入的函数名

|  |
| --- |
| #include <stdio.h>  int My\_Mul(int a, int b);  int My\_Sub(int a, int b);  int My\_Add(int a, int b);  int My\_Bb(int a);  int Hd\_Hs(int(\*p)(), int a, int b);  int main(void)  {  //函数指针  int (\*p)();  // p = My\_Add;  // p = My\_Sub;  p = My\_Bb;  printf("%d\r\n", Hd\_Hs(My\_Bb, 10, 5));  return 0;  }  int Hd\_Hs(int(\*p)(), int a, int b)//回调函数  {  return p(a, b);  }  int My\_Add(int a, int b)  {  return a+b;  }  int My\_Sub(int a, int b)  {  return a-b;  }  int My\_Mul(int a, int b)  {  return a\*b;  }  int My\_Bb(int a)  {  return a\*a;  } |

## 指针函数

本质：函数 返回值类型是指针类型的函数

格式：返回值类型 \*函数名(形参列表)

{

函数体；

}

注意：不能返回局部变量的地址(局部变量在函数结束后空间就会释放) 静态局部变量(static)

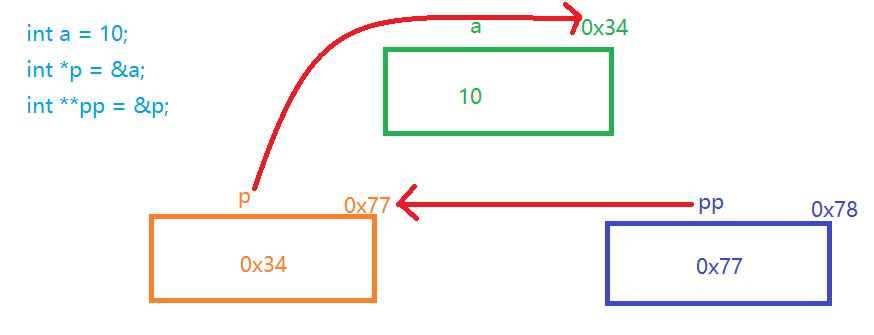
|  |
| --- |
| #include <stdio.h>  int \*Cmp(int a, int b);  //int \*Cmp(void);  /\*  int a = 10;  int b = 20;\*/  int main(void)  {  int \*p = Cmp(10, 5);  printf("%d\r\n", \*p);  return 0;  }  int \*Cmp(int a, int b)  {  if(a > b)  {  return &a;  }else  {  return &b;  }  }  /\*  int \*Cmp(void)  {  if(a > b)  {  return &a;  }else  {  return &b;  }  }\*/ |

## 双重指针

指针的指针 本质：指针 存储地址

格式：数据类型 \*\*指针名；

初始化：数据类型 \*\*指针名 = NULL;



示例：

|  |
| --- |
| #include <stdio.h>  int main(void)  {  int a = 10;  int \*p = &a;  int \*\*pp = &p;  printf("a = %#p\r\n", &a);  printf("p = %#p\r\n", p);  printf("&p = %#p\r\n", &p);  printf("pp = %#p\r\n", pp);  printf("a = %d\r\n", a);  printf("\*p = %d\r\n", \*p);  printf("\*\*pp = %d\r\n", \*\*pp);  return 0;  } |

练习：

将二维数组以双重指针的方式输出