



重慶大學
CHONGQING UNIVERSITY

数据结构实验

时间：2021年（春）--第二周

QQ群：673749661



重慶大學
CHONGQING UNIVERSITY



CONTENTS

01 | 基础知识

02 | 指针概念

03 | 指针与函数

04 | 指针与数组

05 | 指针运算

06 | 二级指针



重庆大学
CHONGQING UNIVERSITY

1. C语言基础知识



1.1 C 程序主要结构



实例1.1

```
#include <stdio.h>                                     //预处理命令，告诉C编辑器包含stdio.h标准库信息
int main() {                                           //定义为main函数，它不接受参数值
    /* 我的第一个 C 程序 */ //注释
    printf("hello world\n"); //main函数调用库函数printf以显示字符序列
                                //\n表示换行符
    return 0;                                           //return 0; 终止 main() 函数，并返回值 0
}
```



C程序结构

- 预处理器指定
- 函数
- 变量
- 语句&表达式
- 注释



1.2 C 基本语法

● 分号

在C程序中，分号是语句结束符。每个语句必须以分号结束。

● 注释/**/

1、**//**----单行注释，如：“//温度表的下限”

2、**/**/**----可以单行或多行注释。如：

“ /* 当fahr=0, 20, ... , 300 时, 分别
打印华氏温度与摄氏温度对照表 */ ”

实例1.2

```
#include <stdio.h>
/* 当fahr=0, 20, ... , 300 时, 分别
打印华氏温度与摄氏温度对照表 */
int main(){
    int fahr, celsius;
    int lower, upper, step;
    lower = 0; //温度表的下限
    upper = 300; //温度表的上限
    step = 20; //步长
    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr,
celsius);
        fahr = fahr + step;
    }
}
```



1.3 变量及其定义

● 变量

变量的名称可以由**字母**、**数字**和**下划线**字符组成。它必须以字母或下划线开头。大写字母和小写字母是不同的，因为 C 是**大小写敏感**的。

● 变量定义&声明

所有变量都必须先声明后使用，用于说明变量属性，由一个类型名和一个变量表组成：type variable_list; 其中type表示C数据类型；variable_list 可以由一个或多个标识符名称组成，多个标识符之间用逗号分隔。如例1.2中：int fahr, celsius;

● 常用变量类型

int	整数 16位 (-32768~32767)
float	浮点数 32位 至少6位有效数字
char	字符 (一个字节)
short	短整型
long	长整型
double	双精度浮点型



1.4 C 常量及定义

● C 常量

常量是**固定值**，在程序执行期间不会改变。

常量可为任何的基本数据类型，比如整数常量、浮点常量、字符常量，或字符串字面值，也有枚举常量。

常量就像是常规的变量，只不过常量的值在定义后不能进行修改。

● 常量定义

在C 中，有两种简单的定义常量的方式：

- 1、使用 `#define` 预处理器，格式：`#define identifier value`
 - 2、使用 `const` 关键字，格式：`const type variable = value;`
-

1.5 C 运算符

C 运算符		说明
算术运算符	+, -, *, /, %, ++, --	自增（减）运算符，整数值增加 1（减少1）
关系运算符	==, !=, >, <, >=, <=	
逻辑运算符	&&, , !	
位运算符	&, 和 ^	作用于位，逐位执行操作
赋值运算符	=, +=, -=, *=, /=, %= <<=, >>=, &=, ^=, =	



1.6 C 判断语句

语句	描述
if 语句	一个 if 语句 由一个布尔表达式后跟一个或多个语句组成。
if...else 语句	一个 if 语句 后可跟一个可选的 else 语句， else 语句在布尔表达式为假时执行。
switch语句	一个 switch 语句允许测试一个变量等于多个值时的情况。
? : 运算符(三元运算符)	Exp1 ? Exp2 : Exp3; 如果 Exp1 为真，则计算 Exp2 的值，结果即为整个 ? 表达式的值。如果 Exp1 为假，则计算 Exp3 的值，结果即为整个 ? 表达式的值



1.7 while循环语句

● While循环/do...while循环

`while (fahr <= upper) {...}`

首先测试圆括号中的条件，若条件(`fahr <= upper`)为真，则执行循环体，再重复上述步骤；当圆括号中条件测试结果为假，则循环结束，并继续执行跟在while循环语句后的下一条语句。 **do...while**除了它是在循环主体结尾测试条件外，其他与 `while` 语句类似。

实例1.2

```
#include <stdio.h>
/* 当fahr=0, 20, ..., 300 时, 分别
打印华氏温度与摄氏温度对照表 */
main(){
    int fahr, celsius;
    int lower, upper, step;
    lower = 0; /* 温度表的下限 */
    upper = 300; /* 温度表的上限 */
    step = 20; /* 步长 */
    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr,
            celsius);

        fahr = fahr + step;
    }
}
```



1.8 for循环语句

实例1.3

```
#include <stdio.h>
/* 打印华氏温度--摄氏温度对照表 */
main(){
    int fahr;
    for(fahr = 0; fahr <= 300; fahr = fahr + 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

圆括号包含三部分，第一部分 $\text{fahr} = 0$ ，初始化部分，仅在进入循环体前执行一次。第二部分 $\text{fahr} \leq 300$ ，为控制循环的测试或条件部分。若条件为真，则执行循环体。第三部分， $\text{fahr} = \text{fahr} + 20$ 以将循环变量增加一个步长，并再次对条件求值，若条件为假，则循环将终止执行。



1.9 函数—函数定义

```
return_type function_name( parameter list )  
{  
    body of the function  
}
```

- **函数类型：** 一个函数可以返回一个值。return_type 是函数返回的值的数据类型。有些函数执行所需的操作而不返回值，return_type 为void。
- **函数名：** 函数名称（function_name）。函数名和参数列表一起构成了函数签名。
- **参数：** 参数列表（parameter list）包括函数参数的类型、顺序、数量。
- **函数主体：** 函数主体（body of the function）包含一组定义函数执行任务的语句。



1.10 C 结构体

- **关键字struct**: 引入结构声明, 结构声明由包含在花括号{ }内的一系列声明组成。
- **结构体标记tag**: 用于结构体命名, 在定义之后, 结构体标记就代表花括号内的声明, 可以用它作为该声明的简写形式。
- **结构体成员member-list**: 标准的变量定义, 比如 `int i;` 或者 `float f,` 或者其他有效的变量定义。
- **结构变量variable-list**: 定义在结构的末尾, 最后一个分号之前, 可以指定一个或多个结构变量。

```
struct tag {  
    member-list  
    member-list  
    member-list  
    ...  
} variable-list ;
```



1.10 C 结构体声明

- **实例1.4结构体声明：**实例1.4声明了拥有3个成员的结构体，分别为字符数组title，整型的book_id，浮点型price。结构体标记为Books，结构体变量为book1。
- **等价于实例1.5：**此声明声明了拥有3个成员的结构体，分别为字符数组title，整型的book_id，浮点型price。结构体标记为Books。用Books标记结构体，另外声明了变量book2。

实例1.4

```
struct Books{  
    char title[50];  
    int  book_id;  
    float price;  
} book1;
```

实例1.5

```
struct Books{  
    char title[50];  
    int  book_id;  
    float price;  
};  
struct Books book2;
```



1.10 C 结构体初始化

- **实例1.6结构体初始化：** 对结构体变量在定义时指定初始值。

- **访问成员变量：** 成员访问运算符 “.” ， 如可以用下列语句打印book标题：

```
printf("title : %s\n", book.title,);
```

- **实例1.6执行输出结果：**

```
title : C 语言
book_id: 123
price: 45.6
```

实例1.6

```
#include <stdio.h>

struct Books{
    char title[50];
    int  book_id;
    float price;
} book = {"C 语言",123,45.6};

int main(){
    printf("title : %s\nbook_id:
%d\nprice: %f\n", book.title,
book.book_id,book.price);
}
```



1.11 C 联合union

- **联合定义：**有时也被称为联合体或共用体。是一种特殊的数据类型，允许在相同的内存位置存储不同的数据类型。可以定义一个带有多成员的联合，**但是任何时候只能有一个成员带有值**。定义如下：

```
union 共用体名{  
    成员列表  
};
```

实例1.7

```
union u_tag {  
    int ival;  
    float fval;  
    char sval;  
} u;
```

- **联合：**u_tag类型的变量可以存储一个整数、一个浮点数，或者一个字符。可根据需要在联合内使用任何内置的或者自定义数据类型。变量u必须足够大，以保存这三种类型中最大的一种。



1.11 C 访问联合成员

- **联合成员访问：**它与访问结构体的方式相同：联合名.成员。
- **联合实例分析：**其中结构体的定义：

```
union Data{  
    int i;  
    float f;  
    char str[20];  
};
```

输出： data.i : 10
data.f : 220.500000
data.str : C Programming

实例1.8

```
#include <stdio.h>  
#include <string.h>  
  
int main( ){  
    union Data data;  
    data.i = 10;  
    printf( "data.i : %d\n", data.i);  
    data.f = 220.5;  
    printf( "data.f : %f\n", data.f);  
    strcpy( data.str, "C Programming");  
    printf( "data.str : %s\n", data.str);  
    return 0;  
}
```



1.12 C 数组

实例1.9

- **实例1.9功能：** 输出一个 4×4 的整数矩阵。程序运行结果：

0	0	0	0
56	99	20	21
23	25	18	60
34	70	23	30

- **实例分析：** 矩阵16个整数，每个整数定义一个变量，即 16 个变量。为提高开发效率，一般采用数组把每一行的整数放在一个变量里，或把 16 个整数全部都放在一个变量里面。

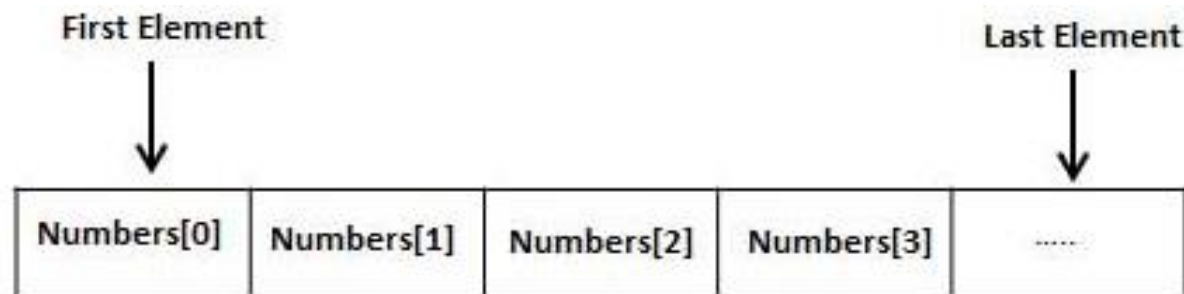
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a1=0, a2=0, a3=0, a4=0;
    int b1=56, b2=99, b3=20, b4=21;
    int c1=23, c2=25, c3=18, c4=60;
    int d1=34, d2=70, d3=23, d4=30;

    printf("%d %d %d %d\n", a1, a2, a3, a4);
    printf("%d %d %d %d\n", b1, b2, b3, b4);
    printf("%d %d %d %d\n", c1, c2, c3, c4);
    printf("%d %d %d %d\n", d1, d2, d3, d4);
    return 0;
}
```



1.12 C 数组声明&定义

数组 (Array) 可以存储一个固定大小的**相同类型**元素的顺序集合。所有的数组都是由连续的内存位置组成。它所包含的每一个数据叫做数组元素 (Element)，所包含的数据的个数称为数组长度 (Length)，最低的地址对应第一个元素，最高的地址对应最后一个元素。



- **声明数组:** `type arrayName[arraySize];` `arraySize` 必须是一个大于零的整数常量, `type` 可以是任意有效的 C 数据类型。如: 声明一个类型为 `int` 的包含 10 个元素的数组 `ndigit`, 声明语句为: `int ndigit[10];`



1.12 C访问数组元素

- **访问数组元素：**C语言中，数组下标一般从0开始，数组元素可以通过数组名称加索引进行访问。访问数组元素，形式为：arrayName [index];
arrayName 为数组名称，index 为下标。
- **实例1.10：**使用循环结构将数据放入数组中（也就是为数组元素逐个赋值），然后再使用循环结构输出（也就是依次读取数组元素的值）。输出结果：100 101 102 103 ... 109。

实例1.10

```
#include <stdio.h>
int main (){
    int n[ 10 ]; /* n 为包含 10 个整数的数组 */
    int i,j;

    /* 初始化数组元素 */
    for ( i = 0; i < 10; i++ ){
        n[ i ] = i + 100; /* 设置元素 i 为 i + 100 */
    }
    /* 输出数组中每个元素的值 */
    for (j = 0; j < 10; j++ ){
        printf("Element[%d] = %d ", j, n[j] );
    }
    return 0;
}
```



1.12 C 数组初始化

- **初始化数组：**在C 中，可以逐个初始化数组，也可以使用一个初始化语句，如：

```
double balance[5] = { 1000.0, 2.0, 3.4, 7.0, 50.0};
```

或

```
double balance[] = { 1000.0, 2.0, 3.4, 7.0, 50.0};
```

 (省略数组的大小)

数组元素的值由{ }包围，各个值之间以 , 分隔。**注：**大括号 { } 之间的值的数量不能大于我们在数组声明时在方括号 [] 中指定的元素数目。若省略数组的大小，数组大小则为初始化时元素的个数，在上例中，两个语句创建的数组。

- **数组元素赋值：**可以只给部分元素赋值，当{ }中值的个数少于元素个数时，表示只给数组前面部分元素赋值。如： `int a[10] = {12, 19, 22 , 993, 344};` 表示只给 `a[0]~a[4]` 5个元素赋值，而后面5个元素自动初始化为0。



1.12 C 数组初始化

- **元素初始化为0：**当赋值的元素少于数组总体元素的时候，剩余元素自动初始化为 0：对于short、int、long，就是整数 0；对于char，就是字符 ‘\0’；对于float、double，就是小数 0.0。如下将数组所有元素初始化为0：

```
int nums[10] = {0};  
char str[10] = {'\0'};  
float scores[10] = {0.0};
```

- **实例1.11输出：**与实例1.9输出相同。

实例1.11

```
#include <stdio.h>  
int main(){  
    int a[4] = {0};  
    int b[4] = {56, 99, 20, 21};  
    int c[4] = {23, 25, 18, 60};  
    int d[4] = {34, 70, 23, 30};  
  
    printf("%d %d %d %d\n", a[0], a[1], a[2], a[3]);  
    printf("%d %d %d %d\n", b[0], b[1], b[2], b[3],);  
    printf("%d %d %d %d\n", c[0], c[1], c[2], c[3]);  
    printf("%d %d %d %d\n", d[0], d[1], d[2], d[3],);  
    return 0;  
}
```



1.13 C 结构体数组

- **结构体数组**：指数组中的每个元素都是一个结构体。
- **结构体数组声明**：在C语言中，定义结构体数组和定义结构体变量的方式类似。如例1.12中，声明了一个结构体stu，并定义了该类型的结构数组class[5]，同时为其分配内存空间。
- **结构体数组初始化**：如：class[5]={...}，当对数组中全部元素赋值时，也可不给出数组长度，如class[]={...}。

实例1.12

```
struct stu{
    char *name; //姓名
    int num; //学号
    int age; //年龄
    char group; //所在小组
    float score; //成绩
}class[5] = {
    {"Li ping", 5, 18, 'C', 145.0},
    {"Zhang ping", 4, 19, 'A', 130.5},
    {"He fang", 1, 18, 'A', 148.5},
    {"Cheng ling", 2, 17, 'F', 139.0},
    {"Wang ming", 3, 17, 'B', 144.5}
};
```



1.14 输入/输出

● **getchar()**函数

从文本流中读入下一个输入字符，并返回结果值。该函数在同一个时间内只会读取一个单一的字符。

● **putchar()**函数

把整型变量c的内容以字符的形式打印出来，显示在屏幕上。该函数在同一个时间内只会输出一个单一的字符。

实例1.13

文件复制

```
#include <stdio.h>
/* copy input to output; 1st version */
main()
{
    int c;
    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
}
```




1.14 输入/输出



gets()函数

`char *gets(char *s)`

函数从 `stdin` 读取一行到 `s` 所指向的缓冲区，直到一个终止符或 EOF



scanf()函数

`int scanf(const char *format, ...)`

函数从标准输入流 `stdin` 读取输入，并根据提供的 `format` 来浏览输入



puts()函数

`int puts(const char *s)`

函数把字符串 `s` 和一个尾随的换行符写入到 `stdout`



printf() 函数

`int printf(const char *format, ...)`

函数把输出写入到标准输出流 `stdout` 并根据提供的格式产生输出



重慶大學
CHONGQING UNIVERSITY

2. 指针基本概念



2.1 指针的概念

每一个变量都有其对应一个内存位置，而每一个内存位置定义了其对应的地址，内存地址可以使用（&）运算符访问。如实例2.1中，访问变量var1，var2地址。

● 实例2.1输出：

var1 变量的地址： 0x7fff5cc109d4

var2 变量的地址： 0x7fff5cc109de

● 指针概念：

指针是保存变量地址的变量。

实例2.1

```
#include <stdio.h>

int main (){
    int var1;
    char var2[10];

    printf("var1 变量的地址： %p\n",
    &var1 );
    printf("var2 变量的地址： %p\n",
    &var2 );

    return 0;
}
```

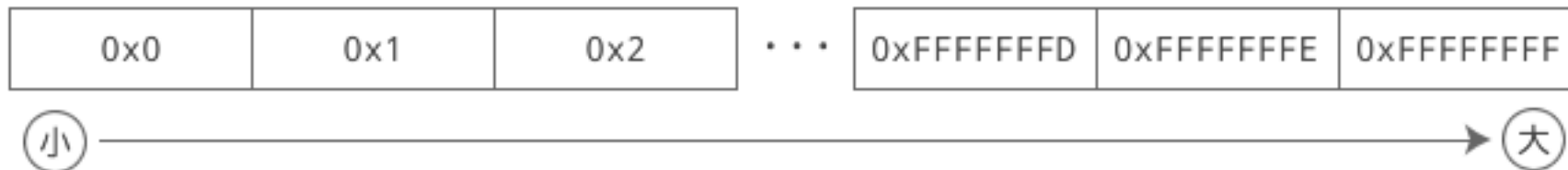
2.2 指针与地址

内存

通常机器有一系列连续编号或编址的存储单元，这些存储单元可以单个或连续成组的方式操纵。在这些存储单元中，不同类型的数据占用的字节数不一样，如 int 占用 4 个字节，char 占用 1 个字节。为了正确地访问内存中数据，必须为每个字节都编上号码，就像身份证号一样，每个字节的编号是唯一的，根据编号可以准确地找到某个字节。一般将内存中字节编号称为**地址**（通常用十六进制表示）。

内存实例

下图是 4G 内存中每个字节的编号（以十六进制表示）：



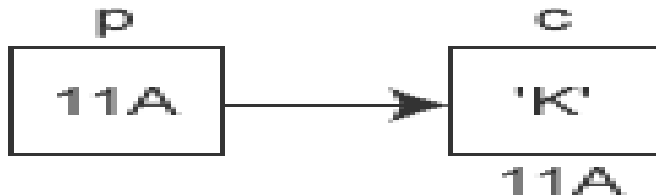
2.3 指针变量

指针变量

指针是能够存放一个地址的一组存储单元（通常是2个或4个字节）。在C语言中，允许用一个变量来存放指针，这种变量称为**指针变量**。指针变量的值就是某个数据的地址，这样的数据可以是数组、字符串、函数，也可以是另外的一个普通变量或指针变量。

实例2.3

若一个 char 类型的变量 c，它存储了字符 'K'（ASCII码为十进制数 75），并占用了地址为 0X11A 的内存。而指针变量 p是指向c的指针，则它的值为 0X11A，如下所示：



2.4 指针定义与使用

指针定义

定义指针变量格式: `datatype *name;` 或 `datatype *name = value;`

`datatype`表示该指针变量所指向的数据的类型。一元运算符 `*` 是间接寻址或间接引用运算符。当它作用于指针时, 将访问指针所指向的对象。

例如: `int *p;` //`p`是指向`int`类型数据的指针。

指针初始化

一元运算符 `&` 可用于取一个对象的地址。地址运算符 `&` 只能应用于内存中的对象, 即变量与数组元素, 它不能作用于表达值或常量。例如: 定义指针变量 `p1`, 并同时
对 `p1` 进行初始化, 将变量 `a` 的地址赋值给 `p1`, 此时 `p1` 指向 `a`:

```
int a = 100;
```

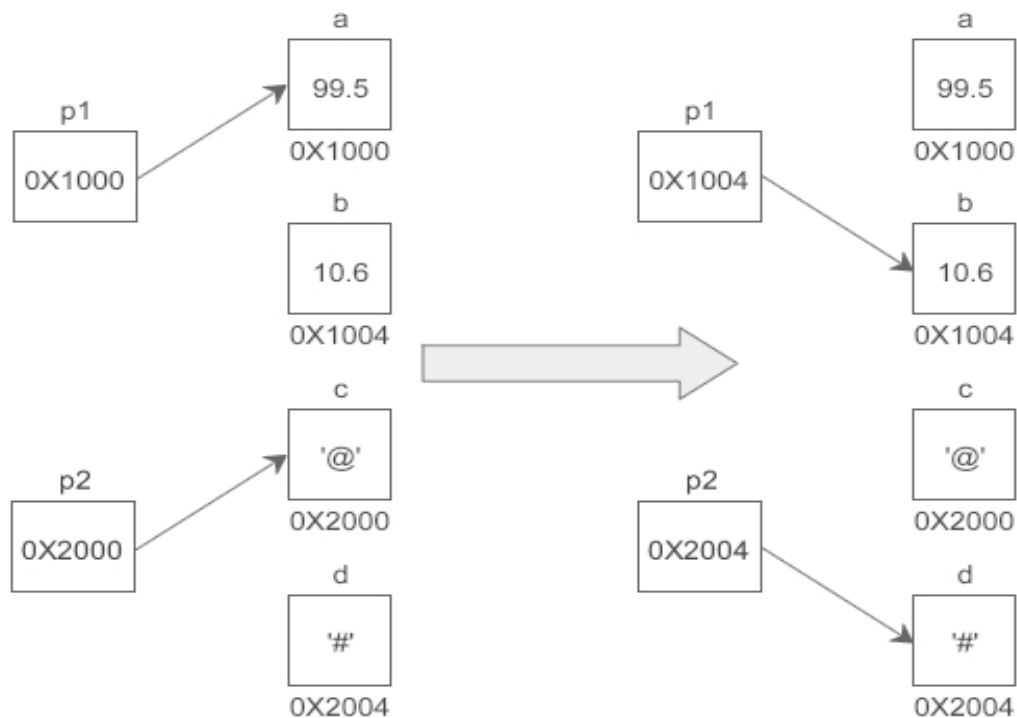
```
int *p1 = &a;
```



2.4 指针定义与使用

实例2.4

实例2.4: 定义指针需使用*, 但指针变量赋值时不能带*. 若变量 a、b、c、d 地址分别为 0X1000、0X1004、0X2000、0X2004, 指针变化:



//定义普通变量

```
float a = 99.5, b = 10.6;
```

```
char c = '@', d = '#';
```

//定义指针变量

// p1 、 p2分别是float* 和char*

```
float *p1 = &a;
```

```
char *p2 = &c;
```

//修改指针变量的值

```
p1 = &b;
```

```
p2 = &d;
```



2.5 通过指针变量访问数据

格式：指针变量存储了数据的地址，通过指针变量能够获得该地址上的数据，格式为：***p;**

实例2.5输出： 15, 15

假设 a 的地址是 0X1000，p 指向 a 后，p 值也会变为 0X1000，*p 表示获取地址 0X1000 上的数据，即变量 a 的值。从运行结果看，*p 和 a 是等价的。若：

*p = *p + 10; // 表示把*p值增加10，即25。

优先级：一元运算符*和&优先级 > 算术运算符

b = *p + 1 ; //把*p指向的对象值取出加1

实例2.5

```
#include <stdio.h>

int main(){
    int a = 15;
    int *p = &a;
    printf("%d, %d\n", a, *p); //两种方式都可以输出a的值
    return 0;
}
```


2.6 NULL 指针

- **NULL指针：**通常在变量声明的时候，如果没有确切的地址可以赋值，为指针变量赋一个 NULL 值是一个良好的编程习惯。赋为 NULL 值的指针被称为**空指针**。NULL 指针是一个定义在标准库中的值为零的常量。

- **判断空指针：**

if(ptr) /* 如果 p 非空，则完成 */

if(!ptr) /* 如果 p 为空，则完成 */

- **实例2.6输出：** ptr 的值为 0x0

实例2.6

```
#include <stdio.h>

int main (){
    int *ptr = NULL;

    printf("ptr 的值为 %p\n", ptr );

    return 0;
}
```



重庆大学
CHONGQING UNIVERSITY

3. 指针与函数



3.1 指针与函数参数

C语言以传值方式将参数值传递给被调用函数。因此，被调用函数不能直接修改主调函数中的变量值。

● 实例3.1输出: a = 66, b = 99

由输出可知，a、b的值没有改变，交换失败。因为参数传递采用传值方式，因此swap()函数内部的 a、b 和 main() 函数内部的 a、b 是不同的变量，占用不同的内存，swap() 交换的是它内部 a、b 的值，不会影响它外部（main() 内部） a、b 的值。

实例3.1

```
#include <stdio.h>
void swap(int a, int b){
    int temp; //临时变量
    temp = a;
    a = b;
    b = temp;
}
int main(){
    int a = 66, b = 99;
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```



3.1 指针与函数参数

可使用主调程序将指向所要交换的变量的指针传递给被调函数，即：swap(&a, &b);

● 实例3.2输出：a = 99, b = 66

由一元运算符&取变量地址，这样&a是指向变量a的指针。swap()参数都声明为指针，并且通过这些指针来间接访问它们指向的操作数。调用 swap() 函数时，将变量 a、b 的地址分别赋值给 p1、p2，这样 *p1、*p2 代表的就是变量 a、b 本身，交换 *p1、*p2 的值也就是交换 a、b 的值。

实例3.2

```
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp; //临时变量
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(){
    int a = 66, b = 99;
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```



3.2 字符指针与函数

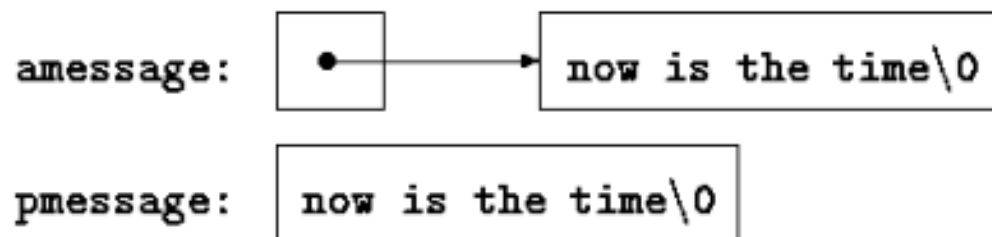


字符指针

```
char amessage[] = "now is the time"; /* 定义一个数组 */
```

```
char *pmessage = "now is the time"; /* 定义一个指针 */
```

上述声明中，amessage是一个仅仅足以存放初始化字符串以及空字符‘\0’的一维数组。数组中单个字符可以修改，但是amessage始终指向同一个存储位置。而pmessage是一个指针，其初值指向一个字符串常量，之后它可以被修改以指向其他地址，但修改不了字符串中的内容。





3.2 字符指针与函数

- **strcpy1()函数**: 通过数组方法实现, 将指针t指向的字符串复制到指针s指向的位置。
- **strcpy2()函数**: 通过指针方法实现。因为参数是通过值传递的, 所以在strcpy()函数可以用任意方式使用参数s和t。其中s和t是进行初始化了的指针, 循环每执行一次, 它们就沿着相应的数组前进一个字符, 直到将t中的结束符' \0' 复制到s为止。

实例3.3

```
/* strcpy: copy t to s; array subscript version */
void strcpy1(char *s, char *t){
    int i;
    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}

/* strcpy: copy t to s; pointer version */
void strcpy2(char *s, char *t){
    int i;
    i = 0;
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```



3.3 指向函数指针

- **定义：** 若把函数首地址赋值给一个**指针**变量，使指针变量指向函数所在的内存区域，然后通过指针变量就可以找到并调用该函数。这种指针就是**函数指针**。

- **定义格式：**

`returnType (*pointerName)(param list);`

其中returnType 为函数返回值类型， pointerName 为指针名称， param list 为函数参数列表。

- **实例3.4执行：** Input two numbers:10 50
Max value: 50

实例3.4

```
#include <stdio.h>
//返回两个数中较大的一个
int max(int a, int b){return a>b ? a : b;}

int main(){
    int x, y, maxval;
    //定义函数指针
    int (*pmax)(int, int) = max;
    printf("Input two numbers:");
    scanf("%d %d", &x, &y);
    maxval = (*pmax)(x, y);
    printf("Max value: %d\n", maxval);
    return 0;}
```



3.4 指针为函数返回值

- **定义：** C语言允许函数的返回值是一个指针，则将这样的函数称为**指针函数**。

- **实例3.5结果：** C Language
c. Language.net
Longer string: c. Language.net

- **注意：**

用指针作为函数返回值时需要注意，函数运行结束后会“销毁”在它内部定义的所有局部数据，包括局部变量、局部数组，函数返回的指针尽量不要指向这些数据。

实例3.5

```
char *strlong(char *str1, char *str2){  
    if(strlen(str1) >= strlen(str2)){  
        return str1;  
    }else{ return str2;}  
}  
int main(){  
    char str1[30], str2[30], *str;  
    gets(str1);  
    gets(str2);  
    str = strlong(str1, str2);  
    printf("Longer string: %s\n", str);  
    return 0;}
```




3.4 指针为函数返回值

- **实例3.6结果:** c.biancheng.net

value = -2

可以看到，现在 p 指向的数据已经不是原来 n 的值了，它变成了一个无意义的值。

- **实例分析:**

“销毁”并非将局部数据所占用的内存全部抹掉，而是程序放弃对它的使用权限，后面的代码可以随意使用这块内存。若func() 运行结束后 n 的内存依然保持原样，值还是 100，则使用也能够得到正确的数据，若有其它函数被调用就会覆盖这块内存，得到的数据就失去了意义。

实例3.6

```
#include <stdio.h>
int *func(){
    int n = 100;
    return &n;
}
int main(){
    int *p = func(), n;
    printf("c.biancheng.net\n");
    n = *p;
    printf("value = %d\n", n);
    return 0;
}
```



重庆大学
CHONGQING UNIVERSITY

4. 指针与数组



4.1 指针与一维数组

- **图4.1.1:** 声明 `int a[10];` 定义一个长度为10的数组a。a[i]表示该数组第i个元素。
- **图4.1.2:** 声明 `int *pa; pa = &a[0];` 表示pa是一个指向整型对象的指针，将指针pa指向数组a的第0个元素，即pa的值为数组元素a[0]的地址。
- **图4.1.3:** pa指向数组中某个元素，则pa+i将指向pa所指向元素之后的第i个元素，pa-i将指向pa所指向元素之前的第i个元素。

图4.1

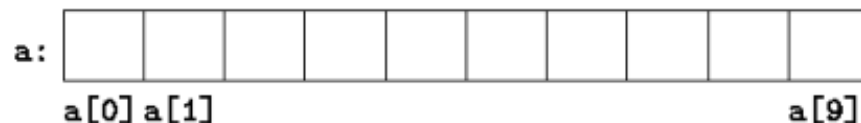


图 4.1.1

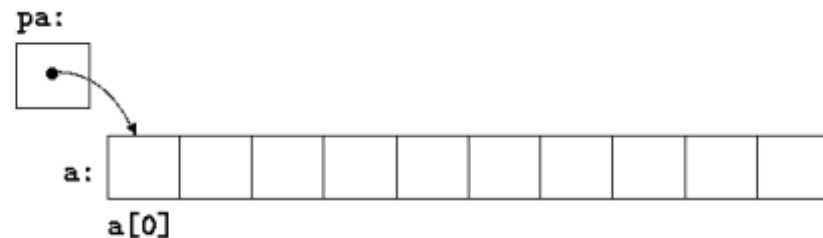


图 4.1.2

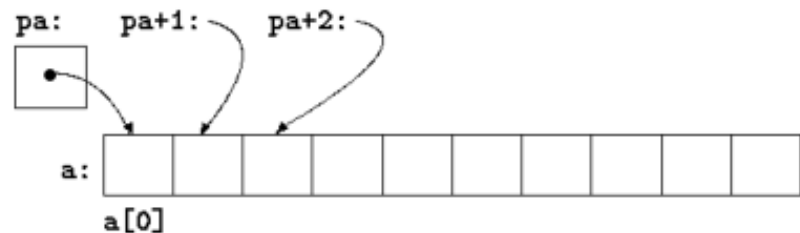


图 4.1.3



4.1 指针与一维数组

- **sizeof()** : sizeof(arr)会获得整个数组所占用的字节数, sizeof(int) 会获得一个数组元素所占用的字节数, 它们相除的结果就是数组包含的元素个数, 也即数组长度。
- ***(p+i)表达式:** arr 是数组名, 指向数组的第 0 个元素, 表数组首地址, 所以int *p = arr;也可写作int *p = &arr[0]; 它们都指向数组第 0 个元素。若 p 指向了数组的第 n 个元素, 那么 p+i 就是指向第 n+i 个元素;
- **实例4.1输出:** 99 15 100 888 252

实例4.1

```
#include <stdio.h>
int main(){
    int arr[] = { 99, 15, 100, 888, 252 };
    //求数组长度
    int len = sizeof(arr) / sizeof(int);
    int i,*p = arr;
    for(i=0; i<len; i++){
        /*(arr+i)等价于arr[i]
        printf("%d ", *(p+i));
    }
    printf("\n");
    return 0;
}
```



4.2 二维数组

- **图4.2.1**: 声明二维数组 `int a[3][4]`; 定义一个 3 行 4 列的二维数组。数组 `a` 为 `int` 类型, 故整个数组共占用 $4 \times (3 \times 4) = 48$ 个字节。
- **图4.2.2**: 二维数组在概念上是二维的, 有行和列, 但在内存中所有的数组元素都是连续排列的。
- **图4.2.3**: 二维数组是按行排列的, 即先存放 `a[0]` 行, 再存放 `a[1]` 行, 最后 `a[2]` 行; 每行中的 4 个元素也依次存放。可把一个二维数组分解成多个一维数组来处理。



图4.2

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

图 4.2.1

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

图 4.2.2

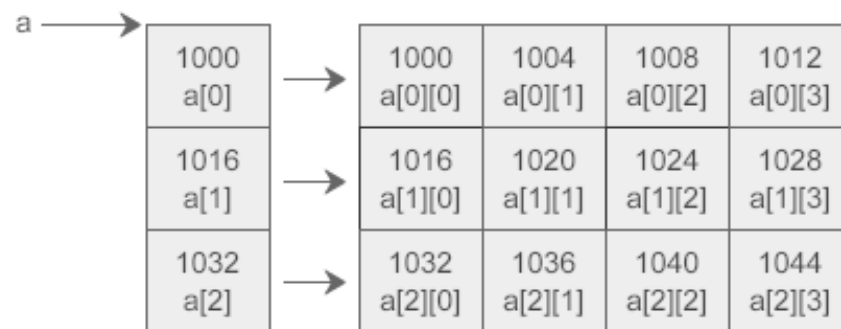


图 4.2.3



4.2 指针与二维数组

- **实例4.2:** 定义一个指向二维数组a 指针变量p, `int (*p)[4] = a;` 括号中的*表明p是一个指针, 它指向一个数组, 数组类型为`int [4]`, 这正是 a 所包含的每个一维数组的类型。

- ***(p+1)表达式:** 指针加法 (减法) 运算时, 它前进 (后退) 步长与它指向的数据类型有关, p 指向的数据类型是`int [4]`, 则p+1就前进 $4 \times 4 = 16$ 个字节, 正好为a 所含的每个一维数组的长度。

- **实例4.2输出:** 16

实例4.2

```
#include <stdio.h>
int main(){
    int a[3][4] = { {0, 1, 2, 3},
                    {4, 5, 6, 7},
                    {8, 9, 10, 11} };
    int (*p)[4] = a;
    printf("%d\n", sizeof(*(p+1)));

    return 0;
}
```



4.2 指针与二维数组

- ***(p+1)表达式:** *(p+1)单独使用时表示的是第 1 行数据, 在表达式中会被转换为第 1 行数据的首地址, 也就是第 1 行第 0 个元素的地址。

- ***(*(p+1)+1)表达式:** *(*(p+1)+1)表示第 1 行第 1 个元素。等价关系: $a+i == p+i$

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

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

- **实例4.3输出:**

0	1	2	3
4	5	6	7
8	9	10	11

实例4.3

```
#include <stdio.h>
int main(){
    int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
    int(*p)[4];
    int i,j;
    p=a;
    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            printf("%2d",*(*(p+i)+j));
            printf("\n");
        }
    }
    return 0;
}
```



重庆大学
CHONGQING UNIVERSITY

5. 指针算术运算



5.1 指针算术运算

- **比较运算：**在某些情况下对指针可以进行比较运算。如指针p和q指向同一个数组的成员，则可以进行==、!=、<、>=的关系比较运算。如p指向数组元素位置在q指向数组元素位置之前，则 $p < q$ 。
- **加减运算：**指针可以和整数进行相加或相减运算。如 $p + n$ 表示指针p当前指向的对象之后第n个对象的地址。指针加减运算跟数据类型长度有关。
- **实例5.1：**指针p被初始化指向s，指向字符串第一个字符。由于p是字符型指针，故执行一次 $p++$ ，p就将指向下一个字符的地址， $p-s$ 则表示字符串长度。

实例5.1

```
/* strlen: return length of string s */
int strlen(char *s)
{
    char *p = s;
    while (*p != '\0')
        p++;
    return p - s;
}
```



5.2 指针递增运算

● 实例 5.2 输出结果:

Address of var[0] = bf882b30

Value of var[0] = 10

Address of var[1] = bf882b34

Value of var[1] = 100

Address of var[2] = bf882b38

Value of var[2] = 200

由结果可知: ptr 每增加一次, 它都将指向下一个整数位置, 即当前位置往后移 4 个字节, 正好是 int 类型的长度。这个运算会在不影响内存位置中实际值的情况下, 移动指针到下一个内存位置。

实例 5.2

```
#include <stdio.h>
const int MAX = 3;

int main (){
    int var[] = {10, 100, 200};
    int i, *ptr;
    /* 指针中的数组地址 */
    ptr = var;
    for ( i = 0; i < MAX; i++){
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        /* 移动到下一个位置 */
        ptr++;
    }
    return 0;
}
```



5.3 指针递减运算

● 实例 5.3 输出结果:

Address of var[3] = bfeedbcd8

Value of var[3] = 200

Address of var[2] = bfeedbcd4

Value of var[2] = 100

Address of var[1] = bfeedbcd0

Value of var[1] = 10

由结果可知: ptr 每减少一次, 它都将指向前一个整数位置, 即当前位置后退 4 个字节, 正好是 int 类型的长度。对指针进行递减运算, 即把值减去其数据类型的字节数。

实例5.3

```
#include <stdio.h>
const int MAX = 3;

int main (){
    int var[] = {10, 100, 200};
    int i, *ptr;
    /* 指针中最后一个元素的地址 */
    ptr = &var[MAX-1];
    for ( i = MAX; i > 0; i--){
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        /* 移动到上一个位置 */
        ptr--;
    }
    return 0;
}
```



5.4 指针比较运算

● 实例 5.4 输出结果:

Address of var[0] = bfdbcb20

Value of var[0] = 10

Address of var[1] = bfdbcb24

Value of var[1] = 100

Address of var[2] = bfdbcb28

Value of var[2] = 200

比较运算: 若 p1 和 p2 指向两个相关的变量, 如同一个数组中的不同元素, 则可对 p1 和 p2 进行大小比较。实例5.4中变量指针所指向的地址小于或等于数组的最后一个元素的地址 &var[MAX - 1], 则把变量指针进行递增。

实例5.4

```
#include <stdio.h>
const int MAX = 3;
int main (){
    int var[] = {10, 100, 200};
    int i, *ptr;
    /* 指针中第0个元素的地址 */
    ptr = var;
    i = 0;
    while ( ptr <= &var[MAX - 1] ){
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );
        /* 指向下一个位置 */
        ptr++;
        i++;
    }
    return 0;
}
```



5.5 指针算术运算

● 实例 5.5 输出结果:

&a=0X28FF44, &b=0X28FF30, &c=0X28FF2B

pa=0X28FF44, pb=0X28FF30, pc=0X28FF2B

pa=0X28FF48, pb=0X28FF38, pc=0X28FF2C

pa=0X28FF40, pb=0X28FF28, pc=0X28FF2A

2686784

由结果可知: pa、pb、pc 每次加 1, 它们的地址分别增加 4、8、1, 正好是 int、double、char 类型的长度; 减 2 时, 地址分别减少 8、16、2, 正好是 int、double、char 类型长度的 2 倍。

实例 5.5

```
main(){
    int  a = 10, *pa = &a, *paa = &a;
    double b = 99.9, *pb = &b;
    char  c = '@', *pc = &c;
    printf("&a=%#X, &b=%#X, &c=%#X\n",
           &a, &b, &c);

    printf("pa=%#X, pb=%#X, pc=%#X\n",
           pa, pb, pc);

    pa++; pb++; pc++;           //加法运算
    printf("pa=%#X, pb=%#X, pc=%#X\n",
           pa, pb, pc);

    pa -= 2; pb -= 2; pc -= 2;   //减法运算
    printf("pa=%#X, pb=%#X, pc=%#X\n",
           pa, pb, pc);

    if(pa == paa){               //比较运算
        printf("%d\n", *paa);
    }else{ printf("%d\n", *pa); }}
```



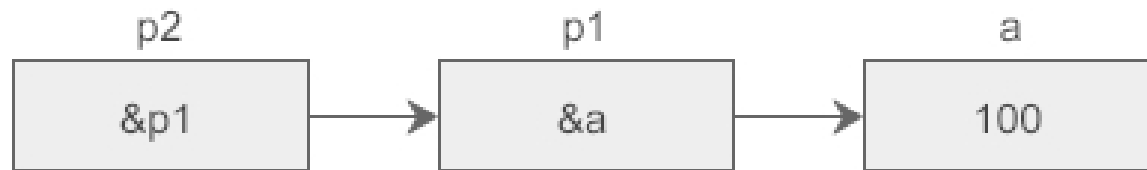
重庆大学
CHONGQING UNIVERSITY

6. 二级指针与指针数组

6.1 二级指针

定义

一个指针指向的是另外一个指针，我们就称它为二级指针，或者**指向指针的指针**。假设有一个 int 类型的变量 a，p1是指向 a 的指针变量，p2 又是指向 p1 的指针变量，它们的关系如下图所示：



对应C代码：

```
int a = 100;  
int *p1 = &a;  
int **p2 = &p1;
```

指针变量也是一种变量，也会占用存储空间，也可以使用&获取它的地址。C语言不限制指针的级数，每增加一级指针，在定义指针变量时就得增加一个星号*。



6.1 二级指针

● 实例6.1输出结果：

100, 100, 100, 100

&p2 = 0X28FF3C, p3 = 0X28FF3C

&p1 = 0X28FF40, p2 = 0X28FF40, *p3 = 0X28FF40

&a = 0X28FF44, p1 = 0X28FF44, *p2 = 0X28FF44,
**p3 = 0X28FF44

以三级指针 p3 为例来分析实例6.1。***p3 等价于*(**p3)。*p3 得到的是 p2 的值，也即 p1 的地址；*(**p3) 得到的是 p1 的值，也即 a 的地址；经过三次“取值”操作后，*(**(*p3)) 得到的才是 a 的值。

实例6.1

```
#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;
}
```




6.2 指针数组

- **指针数组定义：** 如果一个数组中的所有元素保存的都是指针，则称它为**指针数组**。指针定义形式为：`dataType *arrayName[length]`；由于`[]`的优先级高于`*`，括号里面说明`arrayName`是一个数组，包含了`length`个元素，括号外面说明每个元素的类型为`dataType *`。
- **实例6.2输出结果：**
16, 932, 100
16, 932, 100
- **实例分析：** `arr` 是一个指针数组，它包含了3个元素，每个元素都是一个指针。`parr` 是指向数组 `arr` 的指针。

实例6.2

```
#include <stdio.h>

int main(){
    int a = 16, b = 932, c = 100;
    //定义一个指针数组
    int *arr[3] = {&a, &b, &c};
    //定义一个指向指针数组的指针
    int **parr = arr;
    printf("%d, %d, %d\n", *arr[0], *arr[1],
                                   *arr[2]);
    printf("%d, %d, %d\n", **(parr+0),
                                   **(parr+1), **(parr+2));

    return 0;
}
```

6.3 常见指针变量定义

定 义	含 义
<code>int *p;</code>	p 可以指向 int 类型的数据，也可以指向类似 <code>int arr[n]</code> 的数组。
<code>int **p;</code>	p 为二级指针，指向 <code>int *</code> 类型的数据。
<code>int *p[n];</code>	p 为指针数组。[] 的优先级高于 *，所以应该理解为 <code>int *(p[n]);</code>
<code>int (*p)[n];</code>	p 为二维数组指针。
<code>int *p();</code>	p 是一个函数，它的返回值类型为 <code>int *</code> 。
<code>int (*p)();</code>	p 是一个函数指针，指向原型为 <code>int func()</code> 的函数。

7. 练习平台

PTA

Step 1: 打开网页www.pintia.cn

Step 2: 点击右上角注册，通常用邮箱注册

Step 3: 注册完成后登录。

 PTA | 程序设计类实验辅助教学平台



只需一个账号
即可访问百腾教育旗下所有产品

PAT PTA PDS OMS

登录 | 注册 | 考试登录 | 旧版用户迁移 | 微信登录






邮箱地址
请填写邮箱地址

密码
请填写密码

☐ 记住我
重发激活邮件

忘记密码?

登录

固定题目集	我的题目集
 浙大版《C语言程序设计（第3版）》题目集	一直可用
分数: 0	刷题 刷题A 用户 admin
 浙大版《C语言程序设计实验与习题指导（第3版）》题目集	一直可用
分数: 0	刷题 刷题A 用户 admin
 基础编程题目集	一直可用
分数: 0	刷题 刷题A 用户 admin
 数据结构与算法题目集（中文）	一直可用
分数: 0	刷题 刷题A 用户 admin
 Data Structures and Algorithms (English)	一直可用
分数: --	刷题 刷题A 用户 admin