**一、数组的定义、初始化**

1. 判断题
   1. 数组的概念可以理解为一系列相同类型的数据元素的有序集合。
   2. 数组的大小必须是一个常量表达式，也就是说，它的值在编译时就必须确定。
   3. 我们没有明确指定数组的大小，编译器会根据初始化列表自动计算
   4. 当我们在初始化时提供了所有元素的值，可以省略数组大小，编译器会自动计算
   5. C语言支持数组大小没有指定，且也没有初始化列表，如, int a[ ]
   6. 如果初始化值的数量少于数组大小，剩余的元素会被自动初始化为 0（对于数值类型）或 '\0'（对于字符类型）。
   7. int numbers[] = {1, 2, 3, 4, 5}; 则 编译器会自动计算数组大小为5
   8. int partial[5] = {1, 2}; // 剩余元素会被自动初始化为 0
   9. int g[5]; //所有的元素会初始化为 0
   10. char partial[5] = {‘a’, ‘b’}; // 剩余元素会被自动初始化为 ‘\0’
   11. char g[5]; //所有的元素会初始化为 ‘\0’
   12. float e[5]={0.1, 0.2}; //剩余元素会被自动初始化为 0
   13. float e[5]; //所有的元素会初始化为 0
   14. char c[ ]="Hello"; //合法赋值，编译器会自动计算数组大小为5
   15. char c[10]="Hello"; //合法赋值，c[4]为 ‘o’, c[5]为’\0’
2. 填空

int numbers[] = {1, 2, 3, 4, 5}; 数组大小为\_\_\_\_\_\_\_

char vowels[5] = {'a', 'e', 'i'}; prinft(“%c”,vowels[1]) , prinft(“%c”,vowels[3]) 输出分别为\_\_\_\_、\_\_\_\_

int partial[5] = {1, 2}; prinft(“%c”, partial [1]) , prinft(“%c”, partial [3]) 输出分别为\_\_\_\_、\_\_\_\_

char c[10]="Hello";

**二、数组的内存布局**

1. 判断题
   1. C语言中，一维数组在内存中是以连续的方式存储的，这意味着数组的元素在内存中彼此相邻，没有间隔。
   2. 数组名实际上是一个指向数组第一个元素的指针。
   3. 当我们使用索引访问数组元素时，编译器会计算元素的确切内存地址。

例如， int age[5]={18,19,20,21,22};

当要访问 age[2]，编译器会计算：**基地址（age 的地址）+ 2 \* sizeof(int)**。

* 1. sizeof(数组名)，sizeof中单独放数组名，这⾥的**数组名表⽰整个数组，计算的是整个数组的⼤⼩， 单位是字节**

1. 填空：
   1. C语言中，int 类型占用 \_\_\_ 个字节, sizeof(int) = \_\_\_\_\_\_
   2. C语言中，short int 类型占用 \_\_\_ 个字节, sizeof(short int) = \_\_\_\_\_\_
   3. C语言中，char 类型占用 \_\_\_ 个字节, sizeof(chat) = \_\_\_\_\_\_
   4. C代码：

char arr[20]={‘a’,b’};

printf(“%d，”, sizeof(arr));

printf(“%d ”, sizeof(arr[0]));

输出：

——————， ————

* 1. 假设我们有一个整型数组 int arr[3]，每个整数占用 4 个字节。请填内存地址：

|  |  |
| --- | --- |
| 内存地址 | 值 |
| 0x1000 | arr[0] |
|  | arr[1] |
|  | arr[2] |

* 1. C语言代码如下：

int numbers[5] = {10, 20, 30, 40, 50};

int \*ptr = numbers; // ptr 指向数组的第一个元素

printf("第一个元素：%d\n", \*ptr); // 输出 ————

printf("第二个元素：%d\n", \*(ptr + 1)); // 输出————

printf("第三个元素：%d\n", \*(ptr + 2)); // 输出 ————

* 1. C语言代码如下：

int numbers[5] = {10, 20, 30, 40, 50};

int \*ptr = numbers; // ptr 指向数组的第一个元素

printf("%d\n", numbers); // 输出 **0x5ffdb0**

printf("%d\n", ptr); // 输出 \_\_\_\_\_\_\_\_\_\_\_

printf("%d\n", ptr + 1); // 输出 \_\_\_\_\_\_\_\_\_\_\_

printf("%d\n", ptr + 2 ); // 输出 \_\_\_\_\_\_\_\_\_\_

* 1. 使用 sizeof 运算符来查看数组及其元素的内存占用情况,请完成代码

#include <stdio.h>

int main() {

int numbers[5] = {10, 20, 30, 40, 50};

printf("整个数组占用的内存大小：%u 字节\n",\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

printf("每个数组元素占用的内存大小：%u 字节\n", \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

printf("数组元素的个数：%u\n", \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

return 0;

}

输出结果：

整个数组占用的内存大小：\_\_\_\_\_ 字节

每个数组元素占用的内存大小：\_\_\_\_\_\_ 字节

数组元素的个数：\_\_\_\_\_\_

* 1. 以下C语言代码中，使用指针来给数组赋值，请完成填空:

int numbers[5];

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

for (int i = 0; i < 5; i++) {

\*ptr = i \* 10;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

三、数组赋值给另一个数组

1. 判断题
   1. 数组名本质上是指向数组第一个元素的指针这一特性。通过移动指针，我们可以顺序访问和修改数组的每个元素
   2. 在C语言中，我们能将一个数组直接赋值给另一个数组，也能用 == 运算符直接比较两个数组。
   3. 在C语言中，将一个数组赋值给另一个数组，并不像简单的变量赋值那样直接，这是因为数组名本质上是一个指向数组第一个元素的指针，而不是一个可以直接赋值的变量。因此，我们需要采用其他方法来实现数组之间的赋值。
   4. 以下代码：

int a[]={1,2,3};

int b[3];

a=b;

以上可以将数组a直接赋值给数组b，数组b值为: {1,2,3};

* 1. 以下代码：

int a[]={1,2,3};

int b[3]={1,2,3}

if(a==b)

printf(“true/n”);

else

printf(“false/n”);

以上输出结果为 true

* 1. 一个数组赋值给另外一个数组，最直接的方法是使用循环来逐个复制数组元素。这种方法适用于所有类型的数组，无论是整型、浮点型还是字符型。

这种方法的优点是简单直观，适用于各种情况。但是，对于大型数组，这种方法可能会比较耗时。

* 1. 对于需要复制大量数据的情况，使用 memcpy() 函数可能会更高效。这个函数来自 <string.h> 库，可以快速复制内存块。
  2. 若处理的是字符数组（字符串），可以使用 strcpy() 函数。这个函数专门用于复制字符串，会自动处理字符串的结束符 '\0'
  3. strcpy() 函数仅适用于字符数组

二、填空

1.完成以下c语言代码，用memcpy实现数组复制：

int source[5] = {1, 2, 3, 4, 5};

int destination[5];

memcpy(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

printf("Copied array: ");

1.完成以下c语言代码，用strcpy实现数组复制：

char source[20] =” Hello, World!”;

char destination[20];

strcpy (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

printf("Copied array: ");

三、指针数组

1. 判断题
   1. 整型数组，是存放整型数据类型的数组，字符数组是存放字符类型的数组。 同理，指针数组 是存放指针类型的数组.
   2. 数组的概念可以理解为一系列相同类型的数据元素的有序集合。所以，指针数组, 就是一组指针的集合。
   3. 指针数组中的所有元素保存的都是[指针](https://c.biancheng.net/c/80/)
2. 填空

#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};//也可以不指定长度，直接写作 int \*arr[]      
    //定义一个指向指针数组的指针      
    int \*\*parr = arr;  
    printf("%d, %d, %d\n", &a, &b, &c);   
    printf("%d, %d, %d\n", \*arr[0], \*arr[1], \*arr[2]);      
    printf("%d, %d, %d\n", \*\*(parr+0), \*\*(parr+1), \*\*(parr+2));     
    return 0;  
}

运行结果：

16, 932, 100

\_\_\_\_\_\_, \_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_

\_\_\_\_\_\_, \_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_