知识点总结

·在C语言中，指针数组是数组的一种，其数组元素均为指针类型。

·定义指针数组的语法格式：数据类型* 数组名[数组长度];

其中，“数据类型”指的是指针所指向的数据类型。

“*”是指针声明符，表明这个数组里的每个元素都是指针。

例：`int *ptr_arr[3];`

定义了一个名为ptr_arr的指针数组，它有3个元素，每个元素都是指向int类型数据的指针。

·在初始化指针数组时，需要将地址赋给数组里各元素。

例：定义了3个int型变量，并初始化一个指针数组，数组元素分别指向这3个变量。

1）定义指针数组的同时，进行初始化。

```
int num1 = 10;
int num2 = 20;
int num3 = 30;
int *ptr_arr[3] = {&num1, &num2, &num3};
```

2）先定义指针数组，之后再逐个给数组元素赋值。

```
int *ptr_arr[3];
int num1 = 10, num2 = 20, num3 = 30;
ptr_arr[0] = &num1;
ptr_arr[1] = &num2;
ptr_arr[2] = &num3;
```

·可以通过遍历指针数组来访问这些指针所指向的变量值。

例：通过for循环遍历指针数组ptr_arr，并打印出这些指针指向的变量值。

```
int num1 = 10;
int num2 = 20;
int num3 = 30;
int *ptr_arr[3] = {&num1, &num2, &num3};

for (int i = 0; i < 3; i++) {
    // []优先级高于*
    // ptr_arr[i]获取的是当前的指针元素
    // 通过解引用运算符*获取当前指针指向的变量值
    printf("%d\n", *ptr_arr[i]);
}
```

·在C语言中，数组指针是一个指针，它指向一整个数组。

定义数组指针的语法格式：

1）指向一维数组： 数据类型 (*指针名)[数组长度];

其中，“数据类型”是指针指向的数组元素的类型。

“数组长度”是指针指向的数组的元素个数。

如果数组指针指向的是二维数组的一行，数组长度就等于二维数组的列数。

例：`int (*ptr)[5];` 定义了一个名为ptr的数组指针，它指向一个包含5个int类型元素的一维数组。

2）指向二维数组： 数据类型 (*指针名)[行数][列数];

例：`int (*ptr)[2][3];` 定义了一个名为ptr的数组指针，它指向一个包含2行3列的int类型数组。

·初始化数组指针时，是将整个数组的地址赋给指针。

1）指向一整个数组的指针，要赋值为：&数组名

例1：指针ptr_1指向一维数组arr_1。

```
int arr_1[5] = {1, 2, 3, 4, 5};
int (*ptr_1)[5] = &arr_1;
```

例2：指针ptr_2指向二维数组arr_2。

```
int arr_2[3][5] = {
    {1, 2, 3, 4, 5},
    {6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15}
};
int (*ptr_2)[3][5] = &arr_2;
```

·2）指向二维数组第i行的指针，赋值为：&数组名[i]

例：指针ptr_2指向二维数组的第i行。

```
int arr_2[3][5] = {
    {1, 2, 3, 4, 5},
    {6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15}
};
int (*ptr_2)[5] = &arr_2[i];
```

注：二维数组名代表二维数组里首行的地址，arr_2等价于&arr_2[0]。

·通过指针访问数组：指针可以进行算术运算，比如，给指针加或减一个整数。

·数组的元素在内存里是按顺序连续存储的，借助指针的算术运算，可以便捷地访问数组中的任意元素。

·在数组里，指针的算术运算步长由指向的数据类型的大小决定。

比如：

当一个指针指向的是数组里的一个int类型元素时，它的步长就是一个int所占的大小，通常为4字节。

当一个指针指向的是数组里一个char类型元素时，它的步长就以一个char所占的大小，也就是1字节。

当一个指针指向的是整个一维数组时，它的步长就以整个数组的大小为单位。

例1：ptr_1指向一维数组arr_1里的首元素，它的步长为一个元素

```
int arr_1[5] = {1, 2, 3, 4, 5};
// 指针ptr_1指向arr_1[0]
int* ptr_1 = arr_1;
// 此时，指针ptr_1会向后移动两个元素的距离，也就是指向arr[2]
ptr_1 = ptr_1 + 2;
// 此时，指针ptr_1会向前移动一个元素的距离，也就是指向arr[1]
ptr_1 = ptr_1 - 1;
```

例2：ptr_1指向一维数组arr_1，它的步长为整个arr_1数组的大小

```
int arr_1[5] = {1, 2, 3, 4, 5};
// 指针ptr_1指向整个arr_1数组
int* ptr_1 = &arr_1;
// 此时，指针ptr_1会向后移动一整个arr_1数组的距离，超出了数组arr_1的范围
// 得到的是在数组arr_1之外的一个地址。
ptr_1 = ptr_1 + 1;
```

例3：ptr_2指向二维数组arr_2里的首行，它的步长为一行数组

```
int arr_2[3][5] = {
    {1, 2, 3, 4, 5},
    {6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15}
};
// 指针ptr_2指向arr_2[0]这行
int (*ptr_2)[5] = arr_2;
// 此时，指针ptr_2会往后移动一行，也就是指向arr_2[1]这行
ptr_2 = ptr_2 + 1;
```

·使用数组指针访问数组里的元素：

例1：利用指向整个一维数组arr_1的指针ptr_1，打印出arr_1数组里的所有元素值。

```
int arr_1[5] = {1, 2, 3, 4, 5};
int (*ptr_1)[5] = &arr_1;

for (int i = 0; i < 5; i++) {
    printf("%d ", (*ptr_1)[i]);
}
```

例2：利用指向整个二维数组arr_2的指针ptr_2，打印出arr_2数组里的所有元素值。

```
int arr_2[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
int (*ptr_2)[2][3] = &arr_2;
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d ", (*ptr_2)[i][j]);
    }
    printf("\n");
}
```

例3：利用指向二维数组arr_2首行的指针ptr_3，打印出arr_2数组里的所有元素值。

```
int arr_2[3][5] = {
    {1, 2, 3, 4, 5},
    {6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15}
};
int (*ptr_3)[5] = arr_2;

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 5; j++) {
        printf("%d ", *(*(ptr_3 + i) + j)); arr_2[i][j]
    }
    printf("\n");
}
```

数组组索引运算符[]的优先级高于指针声明符*

题目1、若有定义语句：int *ptr[10];

以下叙述正确的是（A）

A．ptr是一个具有10个指针元素的一维数组，每个元素都只能指向整型变量

B．ptr是指向整型变量的指针

C．ptr是一个指向具有10个整型元素的一维数组的指针

D．ptr是一个指针，可以保存整型变量的地址

题目2、设有如下定义语句：int m[ ] = {2, 4, 6, 8}, *k = m;

以下选项中，表达式的值为6的是（A）

A. *(k + 2)

B. k + 2

C. *k + 2

D. *k += 2

题目3、若有定义语句：int a[4][10], *p, *q[4];
且0≤i<4，则错误的赋值是（A）

A. p = a

B. q[i] = a[i]

C. p = a[i]

D. p = &a[2][1]

优先级：() > [] > *

解引用只能对指针使用

数组索引运算符[]不能对整数使用

题目4、设有以下语句：int a[3][4], (* p)[4]; p = a;

则与表达式*(*p + 2)等价的选项是（A）

A．a[0][2]

B．*(a + 2)[0]

C．(*a + 2)[0]

D．a[2][0]