C语言关于指针与指针变量的总结

> 链接：https://www.jianshu.com/p/65d16ffd83a6

指针

1. 一个变量的内存地址称为该变量的“**指针**”。(指针 即 地址)
2. 存放指针的变量称为"**指针变量**"。（指针变量 即 保存地址的变量）
3. 指针变量的作用原理就是我知道这个变量住在哪(地址)，我就能直接找到你并且操作你。 **注意区分指针和指针变量的区别** 就我个人理解（可能不太严谨，但可以帮助理解）： **指针变量是用来存放指针的变量，是一个变量。而指针即地址，分配好空间后地址就不变了，是一个常量**

指针变量和指针运算

——指针变量

1. 指针变量的定义

```
类型说明符 *变量名1, *变量名2, ...;

int *p, *q;
```

——指针运算

1. 取地址运算和赋值运算

```
& 变量  //表示取该变量的地址
int *pa, *qa, a=10;
pa = &a;  // 表示将变量a的地址赋值给指针变量pa，即指针变量pa保存了变量a的地址
qa = pa;  // 表示把指针变量pa保存的地址赋值给指针变量qa，即指针变量qa和pa都保存了变量a的地址
```

2. 取内容运算符

```
* 指针变量名
int *pa, a=10;
pa = &a;  // 指针变量pa保存变量a的地址
*pa = 100;  // 这句话相当于 a = 100
printf("%d", a);  // 100
```

补充：printf中%p以地址方式输出，地址的格式是8位十六进制数(不同计算机会有差异)

```
int *pa, a=10;
pa = &a;
printf("%p", a);  // 0000000A，以地址格式输出变量a的取值，即是8位十六进制的10
printf("%p", &a);  // 0060FEF8，这是变量a所在内存地址
printf("%d", a);  // 10，以十进制格式输出变量a的取值
printf("%p", pa);  // 0060FEF8，以地址格式输出指针变量pa的取值，该取值就是变量a所在内存地址
printf("%p", &pa);  // 0060FEFC，这是指针变量所在内存地址。指针变量也是变量，也要有专门储存它的内存地址。
printf("%d", pa);  // 6356728，以十进制格式输出指针变量pa的取值
```

3. 指针表达式与整数进行加减运算

```
p + n  // p + n的值 = p的值 + p指向类型的字节数 * n
p - n  // p - n的值 = p的值 - p指向类型的字节数*n
```

4. 相同类型指针的减法运算

```
p - q;  // 相减的结果是两个地址(指针)之间间隔的数据个数
```

5. 指针的关系运算

```
// 判断两个指针是否保存同一个地址
p == q;  p != q;

// 判断两个指针保存的地址的先后顺序(即地址的大小)
p > q;  p >= q;  p < q;  p <= q;
```

6. 指针类型的强制类型转换

```
int *px;
(float *)px;  // int型指针px被强制转换为float型指针
```

7. 空指针

```
int *p = 0;
int *q = NULL;
```

指针与数组

——数组元素的指针

```
int a[5] = {2, 4, 6, 8, 10};
int *p, *q;
p = &a[3];  // p保存第四个数组元素的地址
q = a;  // 数组名即数组首地址，相当于q = &a[0];
```

——通过指针引用数组元素

```
int a[5] = {1, 2, 3, 4, 5};
int *p = a;
a[i]  // 数组下标法
*(a+i)  // 数组名即数组首地址，(a+i)得到数组第i个元素得地址，再*(a+i)间接取值
p[i]  // 指针变量下标法
*(p+i)  // 与*(a+i)用法相同
```

——字符指针与字符串

1. 字符指针访问字符串时，字符串的**第一个字符的地址**存放到字符指针变量中。
2. 字符指针指向的均是字符串常量，不能对字符串进行修改。若要修改，需要把字符串存放到字符数组中。

```
char *str = "Welcome to C";
// 或者
char *str;
str = "Welcome to C";
```

将字符指针指向的字符串放入字符数组中以便修改字符串

```
char *s1 = "Welcome to C";
char s2[80];
strcpy(s2, s1);
```

——指针与多维数组

1. 数组名代表数组的首地址，是一个**地址常量。**
2. 数组名是一个**地址常量**，并不是一个指针，指针变量可以**保存地址常量**，指针变量本身也有地址。
3. 而数组名就是一个**地址常量**，一个**地址常量**，一个**地址常量。**
4. 补充，c编译器认为，地址常量取地址符运算&后，依然是这个地址常量

```
int a[3];
printf("%p\n", a);  // 0060FEF4
printf("%p\n", &a);  // 0060FEF4
// 以二维数组a为例
int a[3][4];
```

1. a代表第一个“**一维数组元素**”a[0]的地址，相当于&a[0]
2. a+1代表第二个“**一维数组元素**”a[1]的地址，相当于&a[1]
3. a[1]代表第二个“**一维数组元素**”的第一个“**int型元素**”a[1][0]的地址，相当于&a[1][0]

指针变量p指向二维数组元素的方式

```
int a[3][4];
int *p;
p = &a[i][j];  // p指向元素a[i][j]
p = a[i];    // p指向元素a[i][0]
p = a[i] + j;    // p指向元素a[i][j]
p = *(a+i) + j;    //p指向元素a[i][j]
```

注意以下是错误的：错误原因，*p声明的是指向一个**int型的变量**，而a和&a[i]表示**int型的一维数组**，指针变量指向的变量类型不符合。

```
p = a;
p = &a[i];
```

——指向一维数组的指针变量

这里区分一下，上面讨论的是数组本身的地址操作，而这里讨论的是指向数组的**指针变量**的操作，弄清楚**指针变量**与**地址常量**的关系，指针变量可以**保存地址常量**。

```
类型说明符 (*指针变量名)[数组长度];
int (*p)[5];
int arr[3][5];
p = a;  // 指向二维数组首行a[0]
p = &a[0];  // 指向二维数组首行a[0]
p = a + 2;  // 指向二维数组首行a[2]
```

——指针数组(元素是指针的数组)

```
类型说明符 *数组名[数组长度];

char *lang[5] = {"C", "C++", "JAVA", "JavaScript", "Python"};
```

指针与函数

——指针作为函数的参数

```
#include <stdio.h>
void swap(int *p, int *q)
{
    int temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
int main()
{
    int a=10, b=100;
    int *pa = &a, *pb = &b;
```

```
    swap(pa, pb);
    return 0;
}
```

——指针作为函数的返回值

```
类型说明符 *函数名(形参列表)
{
    // 函数体其它语句
    return 指针变量名;
}
```

——指向函数的指针变量

1. 系统为函数分配一段存储空间，其起始地址称为函数的指针
2. 存放函数的起始地址的变量称为指针变量

1. 函数指针变量的定义

```
类型说明符 (*指针变量名)(函数的形参列表);

int (*p1)(int a, int b);
```

2. 函数指针变量的赋值

```
函数指针变量 = 函数名;

p1 = func;
```

3. 通过函数指针变量调用函数

```
(*函数指针变量)(实参列表);

int c = (*p1)(100, 10);  // 此时p1指针变量指向的func函数被传入实参(100, 10)，并将返回值赋值给整型变量c
```

多级指针

```
int **p, *q, d=10;
q = &d;  //q是一级指针变量
p = &q;  //p是二级指针变量
printf("一级指针变量q保存的地址：%p", q);  // 0060FEF4
printf("一级指针变量q自身的地址：%p", &q);  // 0060FEF8
printf("二级指针变量p保存的地址：%p", p);  // 0060FEF8
printf("二级指针变量p自身的地址：%p", &p); // 0060FEFC
```

动态内存空间分配

给指针变量指向的地址分配特定内存空间

1. malloc函数，返回一个内存空间为size的起始地址

```
void *malloc(unsigned int size);
```

1. calloc函数，返回n个内存空间为size的起始地址

```
void *calloc(unsigned int n, unsigned int size);
```

1. free函数，释放指针变量p指向的内存空间

```
void free(void *p);
```

1. realloc函数，重新分配指针变量p指向的内存空间为size

```
void realloc(void *p, unsigned int size);
```

补充：void类型指针在赋值时会自动适应变量类型

补充：字符串常量可以作为函数的实际参数。只需要将该函数的形式参数在声明函数时定义为字符型指针即可。

```
#include <stdio.h>
#include <stdlib.h>

void printStr(char *str){
    printf("%s", str);
}

int main(){
    printStr("Hello world");
    return 0;
}
```