C语言指针

> 说明: 参考《Linux C 一站式编程》、菜鸟教程、等

1、指针的基本操作

1.1 什么是指针

> 把一个变量所在的内存单元的地址保存在另外一个内存单元中，保存地址的这个内存单元称为指针;
>
> 通过指针和间接寻址访问变量，这种指针在C语言中可以用一个指针类型的变量表示.

例如:

```
int i = 5;
int *pi = &i;
```

- 5这个数字需要用int类型的空间存储；“名字”为 i 类型为 int 的空间存储了 5 。
- &i ---- 取存储空间 i 的地址；
- &i 需要用 类型为 int *的空间存储，“名字”为 pi 、类型为 int * 的空间存储了 &i
- pi 是 指针变量； int * 是 指针变量的类型；
- 那块 存储 &i（地址）的内存单元 就是指针。

在32位操作系统上，指针保存的是32位的虚拟地址；64位操作系统，保存的是64位虚拟地址，8个字节。

1.2 指针的值、指针的类型

这里简单说明

```
char *p = (char *)0x4124;//像这样的就是野指针（十六进制数是随便输入的）（这个也可能不少野指针）
```

- “野指针”：**指向不确定**地址的指针称为“野指针”;(常常会导致段错误) （个人认为这是个主观概念）
- 空指针：指向NULL的指针
- & ： 取地址运算符
- * ： 指针间接寻址运算符

关于 指针类型的含义 将在后面解析

1.3 地址与整型的关系

指针和整型的数据之间可以强制转换。有许多的地方用到，比如：NULL的定义
#define NULL ((void *)0)

```
char *p = (char *) 0x1212;
P =  (void *) 2323323;
//也就是将存储空间中地址为0x1212的这个地址保存到了指针p中；
//你可以这样转换，但是一般不可以 *p这样会有段错误，（地址访问越界）


int i = 9;
printf("%p\n", &i);
printf("%p\n", i);
long p = (long)&i;
printf("%ld\n", p);


运行结果：
0x7ffeefb7cb2c
0x9
140732920220460
```

- 指针占8个字节大小，int 4个字节大小，long 8个字节大小
- &i 和 i 转换成 %p 格式之后，一个是i的地址，一个是按十六进制输出了i里面的内容，也就是0X9。（这里都是16进制）

2、指针类型的参数和返回值

2.1 传值、传指针的区别

这一块的主要重点就是传值、传指针的区别

用一段代码去说明：

```
#include <stdio.h>

void add1(int a, int b);
void add2(int *a, int *b);

void add1(int a, int b)
{
    a++;
    b++;
}

void add2(int *a, int *b)
{
    (*a)++;
    (*b)++;
}

int main(int argc, char *argv[]) {
    int a1 = 4;
    int b1 = 5;

    printf("a1 = %d, b1 = %d\n", a1, b1);
    add1(a1, b1);

    printf("a1 = %d, b1 = %d\n", a1, b1);

    add2(&a1, &b1);
    printf("a1 = %d, b1 = %d\n", a1, b1);

    return 0;
}
```

运行结果：

```
a1 = 4, b1 = 5
a1 = 4, b1 = 5
a1 = 5, b1 = 6
```

2.2 为什么指针能作为返回值、参数?

add1(a1, b1);相当于

```
int *a = &a1;
int *b = &b1;
```

因为这个等式成立，可以赋值，才可以作为参数和返回值。

反例：数组不可以作为参数。

3、指针与数组

3.1 指针与数组

```
int a[5] = {0};
//int b[5] = {1};//只有第一个元素赋值为1，其余都是0；
int *p = &a[0];
//int *p = a;
p++;



void func(int a[])
{
    ...
}

void func(int *a)
```

- 后缀运算符的优先级高于单目运算符（&），所以是取a[0]的地址，而不是取a的地址
- 一个字节大小的内存空间就有一个地址；int占4个字节，用第一个地址（首地址）标记这个int数据
- a[2]之所以能取数组的第2个元素，是因为它等价于*(a+2)
- 数组名作右值时，自动转换成指向首元素的指针
- 数组的下标允许是负数，（在数组或存储空间可控制的地方才有意义）

- 参数写成指针形式还是数组形式对编译器来说没区别，都表示这个参数是指针，之所以规定两种形式是为了给读代码的人提供有用的信息，如果这个参数指向一个元素，通常写成指针的形式，如果这个参数指向一串元素中的首元素，则经常写成数组的形式。

3.2 指针的类型

指针的类型：

> 指针存储的地址 所指向的那块存储空间的 数据类型

- 指针不仅仅有值，而且是有类型的。

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int  pi[5] = {0};

    printf("pi[0] = %p\n", (pi + 0));
    printf("pi[1] = %p\n", (pi + 1));
    printf("pc[0] = %p\n", ((char *)pi + 4));

    return 0;
}
```

运行结果：

> pi[0] = 0x7ffee466cb40
> pi[1] = 0x7ffee466cb44
> pc[0] = 0x7ffee466cb44

- 可以看到，int * 类型的指针转换成为 char * 后，+4才是pi[1]的地址；
- int 占4个字节，char 占一个字节；
- int *类型 +1 就是一个int的空间大小

3.3 二维数组指针的类型

每个数据都有值和类型；当然指针、数组的每个成员也不例外； 但是数组比较复杂一点，总结如下：

- 数组名做右值会自动转换为 指向数组 首元素 的指针。（也就是a[0]）

```
int a[10] = {0};

1、当数组是一维的时候

 这个时候数组名作右值，数组名转换成-数组的首地址，指针的类型是 int *
 此时void *parr = arr；（可以）
 如果这个时候 &arr， 类型为int（*）[10]

2、当数组是二维的时候，int arr[4][5]

  这个时候数组名作右值，数组名也是转换成数组的首地址；
  但是这个时候类型不是int *了，而是指向数组的指针，也就是int（*p）[5];
  此时void * parr = arr；（警告）


    int a[10] = {0};
    // int *p = a;
    // int *p = &a[0];
    // int *p = &a;
    int arr[4][5] = {0};
    // int (*p)[5] = arr;
    // int (*p)[4][5] = &arr;
    // int (*p)[5] = &arr[1];
    // int *p = &arr[2][3];
```

一维数组 int a[10];

	数值	类型
a作右值	数组a的首地址	int *
&a[0]	数组a的首地址	int *
&a	数组a的首地址	int * [10]

二维数组 int arr[4][5];

	数值	类型
arr作右值	数组arr的首地址	int (* p) [5]
&arr[0]	数组arr的首地址	int (*) [5]
&arr	数组arr的首地址	int * [4][5]
&arr[2][3]	(arr + 2 * 3)的地址	int *

- 重点：数组中指针的类型； 多维数组存储的物理模型，逻辑模型（见后面）

4、指向指针的指针与指针数组

4.1

```
int i = 1;
int *pi = &i;//指针
int ** ppi = &pi;//指向指针的指针


int *p[5];//指针数组
ppi = &p[1];
```

4.2 int main（int argc, char *argv[]）

> 我们知道main函数的标准原型应该是int main(int argc, char * argv[]);。
> argc是命令行参数的个数。
> 而argv是一个指向指针的指针，为什么不是指针数组呢?
> 因为前面讲过，函数原型中的[]表示指针而不表示数组，等价于char ** argv。那为什么要写成char * argv[]而不写成char ** argv呢?
> 这样写给读代码的人提供了有用信息，argv不是指向单个指针，而是指向一个指针 数组的首元素。数组中每个元素都是char *指针，指向一个命令行参数字符串。

打印命令行参数：

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    // for (int i = 0; i < argc; i++)
    for (int i = 0; argv[i] != NULL; i++)
    {
        printf("argv[%d] = %s\n", i, argv[i]);
    }


    return 0;
}
```

运行结果：

> bao:day1111 bao$ ./test5 a b c d
> argv[0] = ./test5
> argv[1] = a
> argv[2] = b

> argv[3] = c
> argv[4] = d

5、指向数组的指针与多维数组

略，见3.3

6、指针与结构体

重点：结构体、结构体成员的赋值、定义的一些宏（略）

6.1 访问结构体成员

```
struct student
{
    char name[20];
    int age;
};

struct student wang = {"wang", 20};
struct student *p = &wang;

wang.name
p->name
```

结构体变量wang 和 结构体指针p 访问结构体的成员的方式是不一样的。

- 要通过指针p访问结构体成员可以写成(* p).c和(*p).num，为了书写方便，C语言提供了 -> 运算符，也可以写成p->name和p->age。

6.2 结构体成员的赋值

关于C语言的变量、数组、结构体的赋值

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

struct student
{
    char name[20];
    int age;
};

int main(int argc, char *argv[]) {
    //第一种赋值方法
    struct student wang = {"wang", 20};
    printf("name = %s, age = %d\n", wang.name, wang.age);
    //这种方式是错误的,
    // wang.name = "Misswang";
    wang.age = 21; //这样可以
    // 第二种赋值方法
    strcpy(wang.name, "Misswang1");
    printf("name = %s, age = %d\n", wang.name, wang.age);
    strncpy(wang.name, "Misswang2", 20);
    printf("name = %s, age = %d\n", wang.name, wang.age);

    //第三种赋值方法
    wang.name[0] = 'w';
    wang.name[1] = 'a';
    wang.name[2] = 'n';
    wang.name[3] = 'g';
    wang.name[4] = '\0';//这条语句很重要!
    printf("name = %s, age = %d\n", wang.name, wang.age);

    //
    scanf("%s", wang.name);
    printf("name = %s, age = %d\n", wang.name, wang.age);

    return 0;


<!--没有  wang.name[4] = '\0'; 条语句运行结果为:
name = wang, age = 20
name = Misswang1, age = 21
name = Misswang2, age = 21
name = wangwang2, age = 21
wang1
name = wang1, age = 21
-->
}
```

运行结果:

> name = wang, age = 20
> name = Misswang1, age = 21
> name = Misswang2, age = 21
> name = wang, age = 21
> wang1
> name = wang1, age = 21

为什么数组不能像整型数据那样直接赋值?
1、关于C编译器的解释

> 在初始化字符数组时，编译器会给字符数组首元素赋予初始地址。而后再另外给字符数组赋值字符串，此时字符数组已经具有地址，编译器就会以为是你要给字符数组某个元素赋值。所以说此时是可以给单个字符数组元素赋值单个字符的，但是不可以赋值字符串

2、C语言语法方面

> 数组名不可以作左值，因为它是一个常量，不是一个存储空间。

7、指针与const限定符

8、函数类型和函数指针类型

8.1 函数类型

重点：函数指针类型、回调函数

> 在C语言中，函数也是一种类型，我们可以定义指向函数的指针。指针变量的内存单元存放一个地址值，而函数指针的存放的就是函数入口地址值。

```
#include <stdio.h>

int func(int a, char *p)
{
    printf("func\n");
}

int main(int argc, char *argv[]) {
    int (*p)(int, char *) = func;
    //int (*p)(int, char *) = &func;
    p(5, (char *)3);
    return 0;
}
```

运行结果：

```
baos:day1111 bao$ ./test3
func
```

- 运算符优先级；int (*p)(int, char *) * 先和p结合，这是一个指针，返回值是int，参数是(int, char *)的函数的指针
- 函数名作右值时，自动转换指向函数首地址的指针
- 通过函数指针调用函数：func(5, (char *)3) 或 （* p）(5, (char *)3)则是把函数类型自动转换成 函数指针然后做函数调用。