← C 数组 C 函数指针与回调函数 →

C 指针

学习 C 语言的指针既简单又有趣。通过指针，可以简化一些 C 编程任务的执行，还有一些任务，如动态内存分配，没有指针是无法执行的。所以，想要成为一名优秀的 C 程序员，学习指针是很有必要的。

正如您所知道的，每一个变量都有一个内存位置，每一个内存位置都定义了可使用连字号（&）运算符访问的地址，它表示了在内存中的一个地址。请看下面的实例，它将输出定义的变量地址：

实例

```
#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;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
var1 变量的地址： 0x7fff5cc109d4
var2 变量的地址： 0x7fff5cc109de
```

通过上面的实例，我们了解了什么是内存地址以及如何访问它。接下来让我们看看什么是指针。

什么是指针?

指针是一个变量，其值为另一个变量的地址，即，内存位置的直接地址。就像其他变量或常量一样，您必须在使用指针存储其他变量地址之前，对其进行声明。指针变量声明的一般形式为：

```
type *var-name;
```

在这里，**type** 是指针的基类型，它必须是一个有效的 C 数据类型，**var-name** 是指针变量的名称。用来声明指针的星号 * 与乘法中使用的星号是相同的。但是，在这个语

句中，星号是用来指定一个变量是指针。以下是有效的指针声明：

```
int    *ip;    /* 一个整型的指针 */
double *dp;    /* 一个 double 型的指针 */
float  *fp;    /* 一个浮点型的指针 */
char   *ch;     /* 一个字符型的指针 */
```

所有指针的值的实际数据类型，不管是整型、浮点型、字符型，还是其他的数据类型，都是一样的，都是一个代表内存地址的长的十六进制数。不同数据类型的指针之间唯一的不同是，指针所指向的变量或常量的数据类型不同。

如何使用指针?

使用指针时会频繁进行以下几个操作：定义一个指针变量、把变量地址赋值给指针、访问指针变量中可用地址的值。这些是通过使用一元运算符 * 来返回位于操作数所指定地址的变量的值。下面的实例涉及到了这些操作：

实例

```
#include <stdio.h>

int main ()
{
   int  var = 20;   /* 实际变量的声明 */
   int  *ip;        /* 指针变量的声明 */

   ip = &var;  /* 在指针变量中存储 var 的地址 */

   printf("Address of var variable: %p\n", &var  );

   /* 在指针变量中存储的地址 */
   printf("Address stored in ip variable: %p\n", ip );

   /* 使用指针访问值 */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

C 中的 NULL 指针

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。请看下面的程序：

实例

```
#include <stdio.h>

int main ()
{
   int  *ptr = NULL;
```

```
   printf("ptr 的地址是 %p\n", ptr  );

   return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
ptr 的地址是 0x0
```

在大多数的操作系统上，程序不允许访问地址为 0 的内存，因为该内存是操作系统保留的。然而，内存地址 0 有特别重要的意义，它表明该指针不指向一个可访问的内存位置。但按照惯例，如果指针包含空值（零值），则假定它不指向任何东西。

如需检查一个空指针，您可以使用 if 语句，如下所示：

```
if(ptr)     /* 如果 p 非空，则完成 */
if(!ptr)    /* 如果 p 为空，则完成 */
```

C 指针详解

在 C 中，有很多指针相关的概念，这些概念都很简单，但是都很重要。下面列出了 C 程序员必须清楚的一些与指针相关的重要概念：

概念	描述
指针的算术运算	可以对指针进行四种算术运算：++、--、+、-
指针数组	可以定义用来存储指针的数组。
指向指针的指针	C 允许指向指针的指针。
传递指针给函数	通过引用或地址传递参数，使传递的参数在调用函数中被改变。
从函数返回指针	C 允许函数返回指针到局部变量、静态变量和动态内存分配。

← C 数组　　C 函数指针与回调函数 →

6 篇笔记

指针是一个变量，其值为另一个变量的地址，即，内存位置的直接地址。就像其他变量或常量一样，您必须在使用指针存储其他变量地址之前，对其进行声明。

要理解指针就要先理解计算机的内存。计算机内存会被划分为按顺序编号的内存单元。每个变量都是存储在内存单元中的，称之为地址。

```
#include <stdio.h>

int main ()
```

```
{
    int var = 20;   /* 实际变量的声明 此时的 VAR 这个变量是存在某个地址的，地址对应某个内存单元，该单元中存储了数据20 */
    int *ip;        /* 指针变量的声明 定义了一个指针 即一个内存单元的地址变量 */

    ip = &var;      /* 在指针变量中存储 var 的地址 就是将地址值赋值给指针这个变量*/

    /* 在指针变量中存储的地址 利用&符号直接输出了var所存储的数据的内存单元的地址*/
    printf("Address of var variable: %p\n", &var );

    /* 在指针变量中存储的地址 ip代表的是这个赋值到的地址的值 所以输出的是地址值 */
    printf("Address stored in ip variable: %p\n", ip );

    /* 使用指针访问值 *ip代表的是定义到这个内存单元之后，内存单元中所存储的数据的值也就是将20赋值给var中20这个值 */
    printf("Value of *ip variable: %d\n", *ip );

    return 0;
}
```

STM小菜鸟 1年前 (2017-03-24)

指针是一个变量，所以可以使用任何合法的变量名。在大多数的操作系统上，程序不允许访问地址为 0 的内存，因为该内存是操作系统保留的。

然而，内存地址 0 有特别重要的意义，它表明该指针不指向一个可访问的内存位置。

但按照惯例，如果指针包含空值（零值），则假定它不指向任何东西。

所有指针在创建时都要初始化，如果不知道他指向什么就将 0 赋值给他。必须初始化指针，没有被初始化的指针被称为失控指针(野指针）。

```
#include <stdio.h>

int main ()
{
    int *p = 0;
    int a ;
    p = &a;
    printf ("输入一个数字\n");
    scanf ("%d",p);
    printf("%d\n",*p);
}
```

实例定义了变量 a 和指针变量 p。**p = &a;**表示指针变量指向了变量 a，p 中存放的地址为 a 的地址 &a，*p 所指的是 p 中存放的地址 a 内存单元中的值。

反馈/建议

STM小菜鸟 1年前 (2017-03-24)

```
/*按照偏移值访问函数形参内容实验*/
//二级指针
void Pros(char* a,int b,int e,char et)
{
    char **p=&a;
    //a==*p
    printf("%p %p %p %p \n%p\n",&a,p,a,*p,&b);
    printf("%s\n",*p);
    p++;
    printf("%d\n",*p);
    p++;
    printf("%d\n",*p);
    p++;
    printf("%c\n",*p);
    return;
}

//一级指针访问
void Test(char* a,int b)
{
    char *p=(char*)&a;
    //a!=*p;
    //printf("%p %p %p %p\n",&a,p,a,*p);
    //printf("%p\n",&b);
    //得出结果一级指针自加+1 二级指针自按照元素内容大小自加
    //printf("%d %p\n",*(++p),p);
    //printf("%d %p\n",*(p+8),p+8);
    //a=a[0]一个printf函数以'\0'结束
    //此时p=&a把元素首地址给了p或者说a只记录一个元素首地址的地址
    //同等汇编语句 a: db 'Hello' b:db '16'
    //所以 p=&a != p=a ;
    /*
    char *a="Hello";
    char *b=(char*)&a;
    printf("%p %p %p %p",&a,b,a,&(a[0]));
    */
    //printf("%c %p %p\n",*a,a,&(a[0]));
    //printf("%c %p %p\n",*(a+1),a+1,&(a[1]));
    printf("%c\n",*(*(char**)p));
    //if p=a; *p=a;
    p=a;
    printf("%s",p);
    return;
}

int main()
{
    //Pros("Hello",5,66666,'a');
    Test("Hello",16);
    //指针转换问题
    /*
```

反馈/建议

```
    char *a="Hello";//&a变量里面存储着a所指向的变量地址
    //char **p=&a;
    char *b=(char*)&a;
    char **p=&a;
    printf("%p %p %p %p\n",&a,b,a,*b);
    printf("%p %c\n",&(*a),*(&(*(a+1))));
    printf("%p %c\n",a,*a);//此时a->H，*a=H;
    printf("%p %c\n",(*p),*(*p));
    //p=&a,*p=a所指向的第一个元素的地址还需要一解才能访问正确数据
    //所以1级指针需要解2次 所以进行强制转换
    printf("%c \n",*(*(char**)b));
    //原试解 现在b=&a,*b= &a->a所以如果此时想正确访问H必须在解
    */
    return 1;
}
```

HBR1 1年前 (2017-05-01)

指针的一些复杂说明：

- **int p;** -- 这是一个普通的整型变量
- **int *p;** -- 首先从 p 处开始,先与*结合,所以说明 p 是一个指针, 然后再与 int 结合, 说明指针所指向的内容的类型为 int 型。所以 p 是一个返回整型数据的指针。
- **int p[3]** -- 首先从 p 处开始,先与[] 结合,说明 p 是一个数组, 然后与 int 结合, 说明数组里的元素是整型的, 所以 p 是一个由整型数据组成的数组。
- **int *p[3];** -- 首先从 p 处开始, 先与 [] 结合, 因为其优先级比 * 高,所以 p 是一个数组, 然后再与 * 结合, 说明数组里的元素是指针类型, 然后再与 int 结合, 说明指针所指向的内容的类型是整型的, 所以 p 是一个由返回整型数据的指针所组成的数组。
- **int (*p)[3];** -- 首先从 p 处开始, 先与 * 结合,说明 p 是一个指针然后再与 [] 结合(与"()"这步可以忽略,只是为了改变优先级), 说明指针所指向的内容是一个数组, 然后再与int 结合, 说明数组里的元素是整型的。所以 p 是一个指向由整型数据组成的数组的指针。
- **int **p;** -- 首先从 p 开始, 先与 * 结合, 说是 p 是一个指针, 然后再与 * 结合, 说明指针所指向的元素是指针, 然后再与 int 结合, 说明该指针所指向的元素是整型数据。由于二级指针以及更高级的指针极少用在复杂的类型中, 所以后面更复杂的类型我们就不考虑多级指针了, 最多只考虑一级指针。
- **int p(int);** -- 从 p 处起,先与 () 结合, 说明 p 是一个函数, 然后进入 () 里分析, 说明该函数有一个整型变量的参数, 然后再与外面的 int 结合, 说明函数的返回值是一个整型数据。
- **int (*p)(int);** -- 从 p 处开始, 先与指针结合, 说明 p 是一个指针, 然后与()结合, 说明指针指向的是一个函数, 然后再与()里的 int 结合, 说明函数有一个int 型的参数, 再与最外层的 int 结合, 说明函数的返回类型是整型, 所以 p 是一个指向有一个整型参数且返回类型为整型的函数的指针。

反馈/建议

- `int *(*p(int))[3];` -- 可以先跳过，不看这个类型，过于复杂从 p 开始,先与 () 结合，说明 p 是一个函数，然后进入 () 里面，与 int 结合，说明函数有一个整型变量参数，然后再与外面的 * 结合，说明函数返回的是一个指针，然后到最外面一层，先与[]结合，说明返回的指针指向的是一个数组，然后再与 * 结合，说明数组里的元素是指针，然后再与 int 结合，说明指针指向的内容是整型数据。所以 p 是一个参数为一个整数据且返回一个指向由整型指针变量组成的数组的指针变量的函数。

更多内容参考： C 指针详解

humen_robot 7个月前 (12-26)

指针实例说明：

```
int board[8][8];    /* int 数组的数组 */
int ** ptr;         /* 指向 int 指针的指针 */
int * risks[10];    /* 具有 10 个元素的数组，每个元素是一个指向 int 的指针 */
int (* rusks) [10];  /* 一个指针，指向具有 10 个元素的 int 数组 */
int * oof[3][4];    /* 一个 3 x 4 的数组，每个元素是一个指向 int 的指针 */
int (* uuf) [3][4]; /* 一个指针，指向 3 X 4 的 int 数组 */
int (* uof[3]) [4]; /* 一个具有 3 个元素的数组，每个元素是一个指向具有 4 个元素的int 数组的指针 */
```

狂吠的小疯狗 2个月前 (05-21)

指向函数的指针

代码和数据是一样的，都需要占据一定内存，那当然也会有一个基地址，所以我们可以定义一个指针来指向这个基地址，这就是所谓的函数指针。

假设有函数：

```
double func(int a,char c);
double (*p)(int a,char c);
p=&func;
```

即可以定义一个函数指针。

调用函数

```
double s1=func(100,'x');
double s2=(*p)(100,'x');
```

上面两个语句是等价的。

玲珑争 1周前 (07-06)

点我分享笔记

笔记需要是本篇文章的内容扩展！

反馈/建议