2020/04/20 第14课 指针的使用(指针1)

笔记本: C

创建时间: 2020/4/20 星期— 16:22

作者: ileemi

- 指针
- 指针安全规范
- 下标运算
- 对指针做加法运算
- <u>const的修饰</u>
- 指针的相减运算
- 指针和字符串

指针

- 1、语法层面
- 2、对内存结构熟悉(指针玩的好)

下标访问优于指针间接访问

了解原理,了解状态

指针安全规范

1、任何时候指针只能有两种状态,有效(正常)状态,0 (NULL)状态

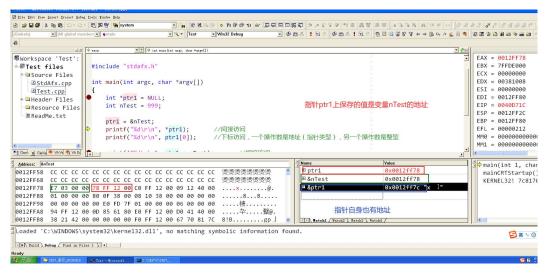
int *p = NULL; //任何指针都要赋初值,不允许出现状态不明

int(解释方式) *(身份标识)p(标识符) = NULL(赋初值);

2、仅仅给出地址,计算机无法正确有效的读取数据

指针和地址的关键区别:指针是一个具有解释方式信息的地址

```
int *p1, p2;
//p1是一个指针
int* p1, p2;
//p1是一个指针,*号跟类型,这类规范同时要求一行语句只定义一个变量
//*号跟着变量走,会更加贴近编译器的思维方式,当编译器扫描的时候,*号跟
右边最近的标识符相结合(修饰最近的标识符)
```



由于指针简单粗暴,所以C语言规定对指针进行强类型检查

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
   int *ptr1 = NULL;
   unsigned char *ptr2 = NULL;
   int *ptr3 = NULL;
   short int *ptr4 = NULL;
   float fTest = 3.14f;
   int nTest = 999;
   int nAry[3] = \{ 6, 7, 8 \};
   ptr1 = &nTest;
   ptr2 = (unsigned char *)&nTest; //由于指针简单粗暴,所以C语言规定对
指针进行强类型检查
   ptr3 = (int *) \& fTest;
   printf("%x\r\n", *ptr3);
   ptr4 = (short int *)0x00400000;
   printf("%x", *ptr4); //间接访问,对其指向的地址上的值以short int
解释方式
   //直接访问,编译器将变量nTest转换成其地址,之后编译器直接产生访问
nTest地址的代码
   printf("%d\r\n", nTest);
   间接访问,第一次访问ptr1上保存的值(变量nTest的地址),
```

```
*/
printf("%d\r\n", *ptrl); //间接访问, 取内容, 运算法则: 按照指针的解释方式(这里 int) 对变量ptrl中所存放的值做间接访问
printf("%02x\r\n", *ptr2); //按照unsigned char 解释方式访问ptr2
指向地址上的值, 输出e7

printf("%d\r\n", ptrl[0]); //下标访问, 一个操作数是地址(指针类型),另一个操作数是整型
return 0;
}
```

直接访问:按照变量自身的类型,访问效率比间接访问高

间接访问:按指针定义时的类型

使用指针需要注意:

1、在哪里

2、是什么

间接运算(取内容)

下标运算

```
公式:

type *ptr = ...;

int n = ...;

对指针做下标运算,得到指针定义时的类型变量

ptr[n] == *(type *)((int)ptr + sizeof(type) * n) //完全等价
```

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

int main(int argc, char *argv[])
{
    int *ptrl = NULL;
    int nTest = 999;

    /*
    type *ptr = ...
    int n = ...
    //对指针做下标运算,得到指针定义时的类型变量
    ptr[n] == *(type *)((int)ptr + sizeof(type) * n)
    */
    printf("%d\r\n", ptrl[nTest]); //下标访问,一个操作数是地址(指
针类型),另一个操作数是整型
```

```
printf("%d\r\n", *(int *)((int)ptrl + sizeof(int) * nTest));
        system("pause");
       return 0;
          printf("%d\r\n", ptr1[nTest]);
 40:
                                           //下标访问,一个操作数是地址(指针类型),另一个操作数是整型
2004010E0
                      eax,dword ptr [ebp-18
ecx,dword ptr [ebp-4]
 004010E3
 004010F6
           mov
                      edx,dword ptr [ecx+eax*4]
          push
 004010E9
                      edx
          push
                      offset string "%d\r\n" (0042603c)
 004010EA
                      printf (00401170)
 004010EF call
 004010F4 add
                      esp,8
         printf("%d\r\n", *(int *)((int)ptr1 + sizeof(int) * nTest));
 41:
 004010F7 mov
                      eax, dword ptr
                      ecx,dword ptr [ebp-4]
 004010FA mov
 004010FD
                      edx,dword ptr [ecx+eax*4]
          mov
 00401100
          push
                      edx
                      offset string "%d\r\n" (0042603c)
 00401101
          push
 00401106
          call
                      printf (00401170)
0040110B
          add
                      esp.8
42:
```

数组名是数组第0个元素类型的指针常量

```
int *ptr = NULL;
int nTest = 999;
int nAry[3] = \{6, 7, 8\};
数组名是数组第0个元素(第0个元素为 int)类型的指针常量
数组名是 int 类型的指针常量
ptr = nAry; //int类型的常量可以给int类型的变量赋值
直接访问,nAry是一个数组名,在编译器眼里数组名就是一个常量
*(int *) 解释为整型
sizeof得到的都是常量,编译的时候产生的常量值
printf("%d\r\n", nAry[nTest]);
间接访问
2、将ptr地址上的值(所指向的数组首地址: 0x0012ff68)
printf("%d\r\n", ptr[nTest]);
```

sizeof得到的都是常量,编译的时候产生的常量值

从反汇编窗口可以观察到,在编译器没有优化之前(按照语法)**使用数组名进行下标访问比使用指针进行下标访问的效率要高**,编译器优化后都一样。

```
43:
             printf("%d\r\n", nAry[nTest]);
→0041267E
  00412681
                              ecx,dword ptr [ebp+eax*4-24h]
  00412685
               push
                        ecx
offset string "%d\r\n" (0042603c)
  00412686 push
  0041268B call printf (00401170)
  00412690
               add
                              esp,8
             printf("%d\r\n", ptr1[nTest]);
  44:
  00412693
  00412696 mov
                            eax,dword ptr [ebp-4]
 00412699 mov ecx,dword ptr [eax+edx*4]
0041269C push ecx
0041269D push offset string "%d\r\n" (0042603c)
004126A2 call printf (00401170)
004126A7 add esp,8
 00412699 mov
0041269C push
```

对指针做加法运算

```
type *ptr = ...;
int n = ...;
```

对指针做加法运算,必须是整型系类型(short int、int、unsigned int),得到同类型的指针常量

```
ptr + n = (type *const)((int)ptr + sizeof(type) * n)
```

const的修饰

```
int nAry[3] = {1, 2, 3};
int nTest = 999;
int *ptr = nAry;
/*
int const* ptr = nAry; const在 * 左边的,间接访问的目标是常量,不能间接写入,所以这里会编译错误
int *const ptr = nAry; const在 * 右边的,指针自己是常量,指针变量
不能修改
const int *const ptr = nAry; 间接访问的目标是常量,指针自己是常量
*/
ptr = &nTest; /常量不能进行修改
*ptr = 0; //间接写入,指针不是常量,指向的目标是常量,常量不能修
改
```

指针的相减运算

```
type *ptr1 = ...;
type *ptr2 = ...;
```

```
int n = ...;
减法运算的规则:
同类型的指针才能相减,得到整型常量
ptr2 - ptr1 = ((int)ptr2 - (int)ptr1) / sizeof(type);
```

对两个指针做整型值相减再除以 sizeof(type) 得到整型常量

```
#include <stdlib.h>
int main(int argc, char *argv[])
   int *ptr1 = NULL;
   int nTest = 999;
   int nAry[3] = \{ 6, 7, 8 \};
   ptr1 = &nTest;
   3 (常量)
   这样做可以求出数组中有多少个元素
      2、不利于维护(变量nTest需要在数组前)
   printf("%d\r\n", ptr1 - nAry); //间接访问
   printf("%d\r\n", sizeof(nAry) / sizeof(nAry[0]));
   system("pause");
```

```
int *ptr = NULL;
ptr++; //ptr = ptr + 1
```

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

int main(int argc, char *argv[])
{
   int *ptr = NULL;

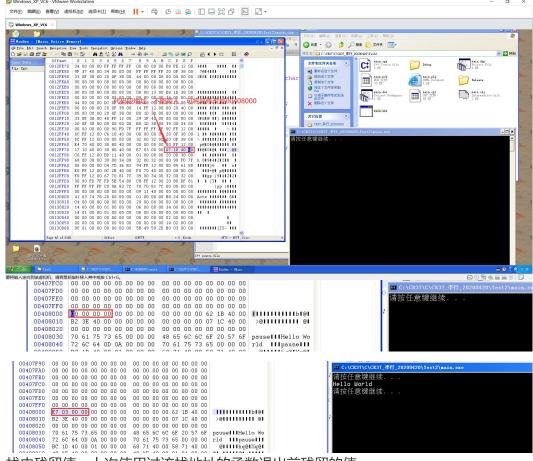
   int nTest = 999;
   int nAry[3] = { 6, 7, 8 };
   ptr = nAry;
```

```
// ++ 与指针ptr 先结合,得到同类型指针,由于后置++,表达式完成后才
                                *(ptr++) = 3:
                                 // *号与指针ptr先结合,得到整型,然后对整型自加一
                                  //(*ptr)++ = 3; 编译错误,后置++得到常量,前置++得到变量
                                  (*ptr)++;
                                printf("%0x\r\n", nAry[0]);
                                 printf("\%0x\r\n", ptr);
                                system("pause");
                                return 0;
 P File Edit View Insert Project Bebug Tools Mindow Hel
 (All global members • • main
                                                                                                                                                                                                                              ☑ 食服火 : 可② 食服火 : 时④ ❷ $ $ $ $ 4 → $ 6 4 4 ₹ 8 4
  ®Workspace 'Test': ●{
                                                                              int *ptr = NULL;
   ■ #Test files
       ⊜ aSource Files
                                                                              int nTest = 999;

    StdAfx.cpp

               ∄Test.cpp
                                                                              int nAry[3] = { 6, 7 , 8 };
ptr = nAry;
       ⊕ ≅Header Files
          ■ Resource Files
                                                                              // ++ 与指针ptr 先结合,得到同类型指针,由于后置++,表达式完成后才加一(类型)
          ■ ReadMe.txt
                                                                              *ptr++ = 3;
                                                                              //(*ptr)++
printf("%d\r\n", nAry[0]);
                                                            system("nause").
 Classic FileVie VA Vie 🐴 VA Ou
     Address: 0x0012FF7C
  nTest
                                                                                                                                                                                                                                                   ⊞ nAry
                                                                                                                                                                                                                                                                                                                    0x0012ff6c
                                                                                                                                                                                                                                                   ⊞ ptr
                                                                                                                                                                                                                                                                                                                        0x0012ff70
  0012FF9C 00 30 FD 7F 01 00 00 00 06 00 00 09 4F F1 2 00 .0?.....................
0012FFAC 0D 85 61 80 E0 FF 12 00 C0 4B 40 00 50 61 42 00 .卆.....繩@.PaB.
  0012FFBC 00 00 00 00 F0 FF 12 00 67 70 81 7C 00 00 00 00 .....gp ]....
AA12FFCC 0A 0A 0A 0A 0A 0A 0A FD 7F FD 5R 54 8A CR FF 12 0A A2的T
                                                                                                                                                                                                                                                ◀ ▶ Natch1 ( Watch2 ) Watch3 ) Watch4 /
 Loaded 'C:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.
 #### | Torrord Vania (*** (Dreak) = [### Case op s]

| Dight for the foot place flow foot place flow flow
| Dight for the foot place flow flow
| Dight for the foot place flow flow
| Dight for the foot place flow flow
| Dight foot flow flow
| Dight flow flow flow flow flow
| Dight flow flow flow
| Dight flow
| Dig
                                                                                                                                                                                                                                                                                                                                                 // ++ 与指针ptr 先结合,得到同类型指针,由于后置++,表达式完成后才加一(类型)*ptr++ = 3;
                                                                   // 毛与指针ptr先结合 . 得到整型 . 然后对整型自加一
//(*ptr)++ = 3; 编译错误 . 后置++得到常量 , 前置++得到变量
(*ptr)++;
printf(*%8x)*/h", nAry[0]);
printf(*%8x)*/n", ptr);
  | Address: | ModRIFFEC | ModR
                                                                                                                                                                                                                                                                                                                                                  ⇒main(int 1, char
mainCRTStartup()
KERNEL32! 7c8170€
                                                                                                                                                                                                                                                                               0x0012ff7c "p ]"
                                                                                                                                                                                                                       nTest
                                                                                                                                                                                                                                                                               0x0012ff6
                                                                                                                                                                                                                    ⊞ nArv
                                                                                                                                                                                                                ⊞ptr
                                                                                                                                                                                                                ▼ ▶ Vatch1 / Vatch2 \ Vatch3 \ Vatch4 /
  Loaded 'C:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.
   Duild Debug Find in Files 1
                                                                                                                                                                                                                                                                                                                                                                Ln 13, Col 16
```



栈内残留值,上次使用过该栈地址的函数退出前残留的值

指针和字符串

局部指针初值为一个字符串常量的首地址(字符串常量属于全局区), 0x0042.... 数据区已初始化 只读,不能写

栈区数据可修改

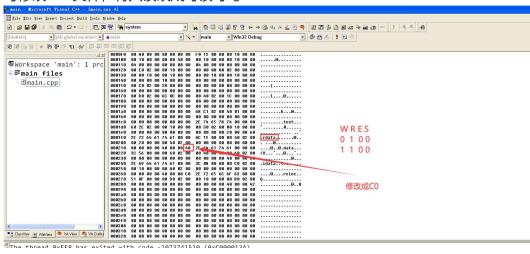
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char szBuf[] = "Hello Wolrd";
    char *psz = "Hello World";

    szBuf[0] = 'h';
    psz[0] = 'h';

    puts(szBuf);
    puts(psz);
    system("pause");
    return 0;
}
```

可修改.exe文件,将只读改成可读可写



The thread GVEFR has evited with code -10737/1510 (GVCGG