2020/04/21_第15课_指针函数和函数指针(指针2)

笔记本:	C
创建时间:	2020/4/21 星期二 15:06
作者:	ileemi
标签:	函数指针, 指针函数

指针二

指针函数

就是一个返回值为指针的函数，其本质还是一个函数。
声明：返回值类型 *函数名(参数表)

例如：int *Fun(int a, int b);
返回值是一个int类型的指针，是一个地址

隐藏错误：

1、不应该返回参数的地址

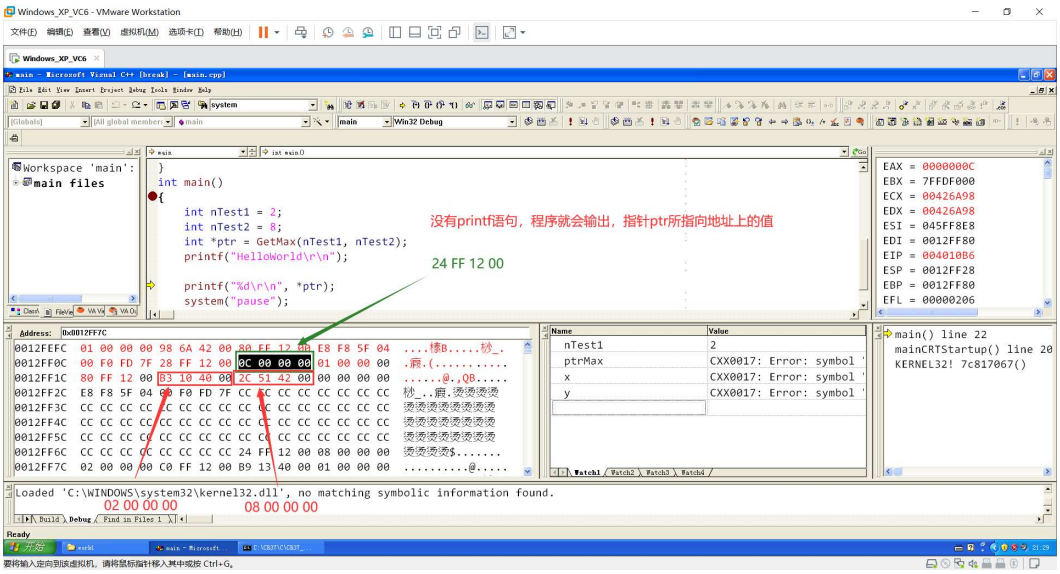
- 当该函数调用完成后，参数被释放。
- 返回的指针引用了一个栈外的地址，这个地址会被新的函数占用并覆盖修改。

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

int *GetMax(int x, int y)
{
    int *ptrMax = &x;
    if (y > x)
    {
        ptrMax = &y;
    }

    return ptrMax;
}
int main()
{
```

```
    int nTest1 = 2;
    int nTest2 = 8;
    int *ptr = GetMax(nTest1, nTest2);
    //printf("HelloWorld\r\n");
    /*
    没有上一句printf，程序就会输出8，如果有上面的printf语句，程序输出的值非8，
    在本地测试时数值：4346156
    */
    printf("%d\r\n", *ptr);
    system("pause");
    return 0;
}
```



将上面的程序放到VS2019上会有以下提示信息：

2、不应该返回局部变量的地址

- 当该函数调用完成后，局部变量被释放。
- 返回的指针引用了一个栈外的地址，这个地址会被新的函数占用并覆盖修改。

返回地址在main栈范围内就安全，

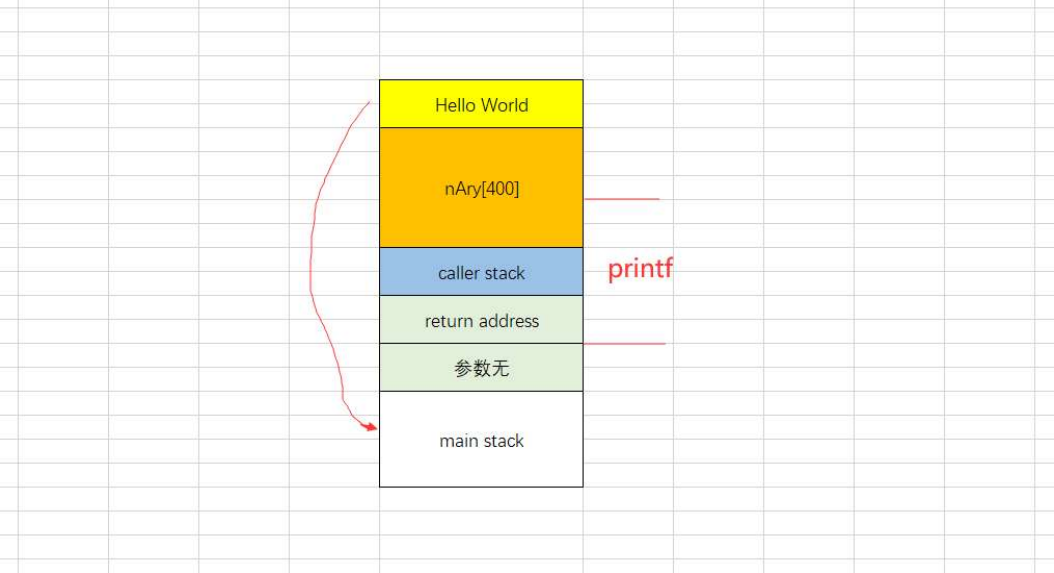
被传递的值是main函数中的局部变量的地址，就是安全得

返回的指针地址 只要是 在caller 的栈段 。不会有被覆盖的风险，如果再caller的栈外又不属于全局作用域的就有风险了

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

```
char *foo()
{
    //int nAry[400] = { 0 };
    char szBuf[] = "Hello World";
    return szBuf;
}
int main()
{
    /*
    Debug下
    指针函数foo内部初始化nAry数组后，main()函数的两条显示语句将会正常显示Hello World
    如果指针函数foo内部未初始化nAry数组，显示ASCII码😀
    */
    printf("%s\r\n", foo());
    puts(foo());
    system("pause");
    return 0;
}
```

nAry数组给的足够大时，printf产生的操作就不会覆盖到指针函数的返回值



函数指针

其本质是一个指针变量，该指针指向这个函数，函数指针就是一个指向函数的指针。
声明：函数返回值类型 (*指针变量名)(函数参数列表)

示例：int (*pFunTest)(int nTest1, int nTest2);

函数指针需要将一个函数的地址赋值给它，有两写法：
1、FunTest = &Funtion;
2、FunTest = Funtion;

&(取地址符)可以不写，因为函数名和数组名就表示其首地址

函数指针调用函数

```
int FunTest(int x,...); //声明一个函数FunTest

int (*pFunTest) (int x,...); //定义一个函数指针

pFunTest = FunTest; //将FunTest函数的首地址赋给指针变量pFunTest
```

赋值时函数 FunTest 不用带括号，也不用带参数。因为函数名 FunTest 代表了改函数的首地址，所以经过赋值以后，指针变量 pFunTest 就指向函数 FunTest() 代码的首地址了。

函数指针学习的三个方面：
1、熟悉基本语法
2、掌握内存原理
3、了解设计原理

typedef

//取别名，以进一步说明其类型意义时使用
宏是编译前，编译器对源码文本做的查找并替换的工作,不增加新类型
typedef不同，他是让编译器实实在在地认为有了这个新类型（增加新类型）
typedef int INT; 加分号，取别名
typedef float HEIGHT;

//标准
typedef unsigned int size_t;
typedef unsigned int wchar_t;

函数的第一条被执行的指令地址，称为函数的首地址
函数名，就表示函数首地址的常量

同参数个数，同参数类型顺序，同返回值，同调用约定
这样的函数指针，才是同类型的函数指针
在设计中，函数指针常用于接口设计

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

```
//比较x, y的数值，返回最大值得地址
int *GetMax(int x, int y)
{
    int *ptrMax = &x;
    if (y > x)
    {
        ptrMax = &y;

    }
    return ptrMax;
}

// char *foo()
// {
//     //int nAry[400];
//     char szBuf[] = "Hello World";
//     return szBuf;
// }

void FunA()
{
    puts("采购");
}

void FunB()
{
    puts("加工");
}

void FunC()
{
    puts("库存");
}

void FunD()
{
    puts("销售");
}

int main()
{
//     FunA();
//     FunB();
//     FunC();
//     FunD();
    //void (*pfn)() = FunA();
```

```
    //自定义流程，本例使用简单的线性流程
    void (*pfn[])() = {
        FunA, FunB, FunC, FunD
    };

    for (int i = 0; i < sizeof(pfn) / sizeof(pfn[0]); i++)
    {
        pfn[i]();
    }

//     printf("%s\r\n", foo());
//     puts(foo());


//     int nTest1 = 2;
//     int nTest2 = 8;
//     int *ptr = GetMax(nTest1, nTest2);
//     printf("HelloWorld\r\n");
//
//     printf("%d\r\n", *ptr);
    system("pause");
    return 0;
}
```

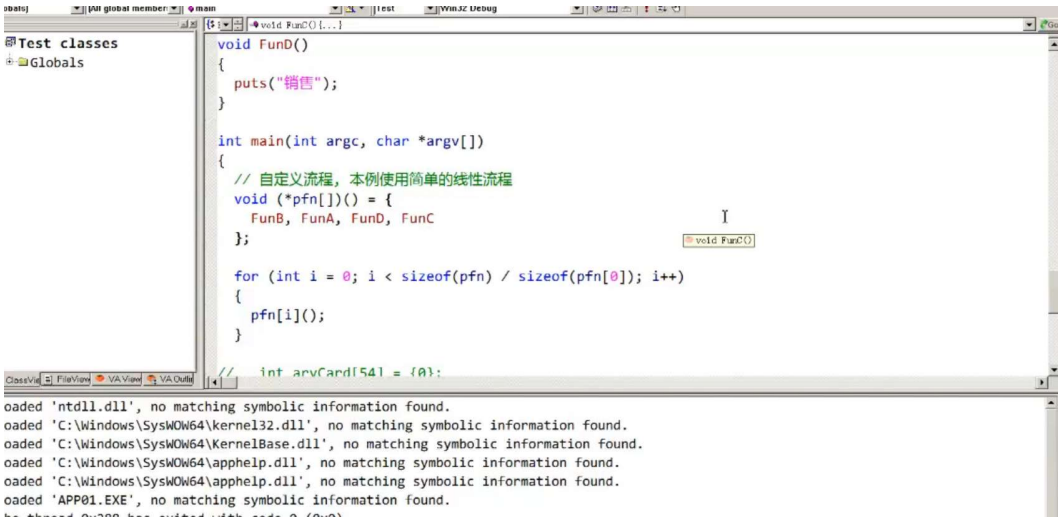
热更新
热补丁
适合规模小的程序
规模大的程序就需要重启

运行前无法预知
函数指针运行时可以修改程序的行为

运行时修改行为可以用函数指针

业务(行业逻辑)和界面分离
常用算法和业务分离

参数化行为



函数指针在程序运行的时候可以根据具体情况更新程序的具体值

不要为了达到某个设计模式而设计
为了解决需求而设计
设计模式是为了解决问题的，如果没有非用不可的必要，就别用