#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <stdio.h>

//结构体初阶

//数组：一组相同类型的元素的集合

//结构体：也是一些值的集合，但是值的类型可以不同!

//例如：描述一个人，人是一个复杂对象，不能用C语言提供的基础类型进行单一描述

//这时就需要自定义一个类型，即使用结构体自定义类型

//自定义类型Stu

struct B

{

char c;

short s;

double d;

};

struct Stu

{

//成员变量

struct B mb;

char name[20];//名字

int age;//年龄

char id[20];//学号

}s1,s2;//此处s1和s2也是结构体变量，不过是全局变量 （变量列表）

//int main()

//{

// struct Stu s = {"李明",24,"E20230321"};//Stu是类型，s是对象，属于局部变量

// struct Stu \*p = &s;

// printf("%s\n", p->name);

// return 0;

//}

//结构体成员的类型可以是标量、数组、指针、甚至是其他结构体。

//结构体的初始化使用{}，初始化每一个成员可以在里面再放{}

//例如：struct Stu s={{'w',20,3.14},"张三",30,"ID20230322"};

//struct Point

//{

// int x;

// int y;

//};

//

//struct Node

//{

// int data;

// struct Point p;

// struct Node\* next;

//

//}n1 = { 10,{5,6},NULL };//结构体嵌套初始化

//

//struct Node n2 = { 20,{5,6},NULL };//结构体嵌套初始化

//

//int main()

//{

// struct Node\* pnode = &n2;

// printf("%d %d,%d ,%p",n2.data,pnode->p.x,n2.p.y,n2.next);

// return 0;

//}

/\*结构体传参\*/

void print1(struct Stu st)

{

printf("%c--%d--%.2f\n", st.mb.c, st.mb.s, st.mb.d);

printf("%s\n", st.name);

printf("%d\n", st.age);

printf("%s\n", st.id);

}//值传参

void print2(struct Stu\* st)

{

printf("%c--%d--%.2f\n", st->mb.c, st->mb.s, st->mb.d);

printf("%s\n", st->name);

printf("%d\n", st->age);

printf("%s\n", st->id);

}//址传参

int main()

{

struct Stu s = { {'w',20,3.14},"张三",30,"ID20230322" };

print1(s);//传值调用

printf("\n");

print2(&s);//传址调用

return 0;

}

//函数传参的时候，参数是需要压栈的，如果传递一个结构体对象的时候，结构体过大

//参数压栈的系统开销就比较大，所以会导致性能的下降。

//结论：结构体传参优先使用传址调用，可节约空间、时间，传参效率更高。

//函数调用的参数压栈：

//栈是一种数据结构：先进的后出，后进的先出

int Add(int x, int y)

{

int z = 0;

z = x + y;

return z;

}

int main()

{

int a = 3;

int b = 5;

int c = 0;

c = Add(a, b);//传参顺序，从右到左，先b后a

return 0;

}

/\*

上面的程序执行时，在函数调用之前，会将地址压入栈区，

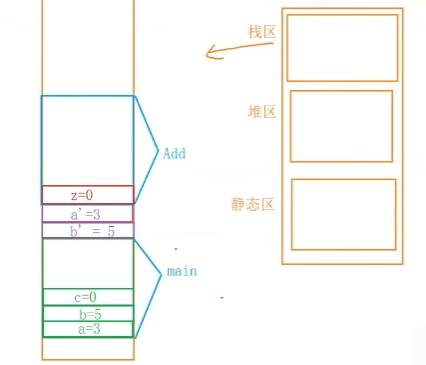
再把你调用的那个函数里的一些局部变量啊，形参啊等等压入栈中

你函数调用执行完毕。栈就会把你调用的这个函数之前压入栈的变量

和形参全部清除出栈，之后根据下一条代码的地址，接着执行程序，

以后的程序也都是这么执行。

\*/



注：感兴趣可以学习“函数栈帧的创建和销毁”