#define \_CRT\_SECURE\_NO\_WARNINGS 1

//调试技巧2：实践篇

//#include <stdio.h>

//int main()

//{

// int arr[10] = { 0 };

// int sz = sizeof(arr) / sizeof(arr[0]);

// int i = 0;

// for (i = 0; i < sz; i++)

// {

// arr[i] = i + 1;//条件断点 i==5。

// }

// for (i = 0; i < sz; i++)

// {

// printf("%d\n", arr[i]);

// }

// return 0;

//}

//实例2：

//实现代码：求1!+2!+3!...+n!; 不考虑溢出

//调试前

//#include <stdio.h>

//int main()

//{

// int i = 0;

// int sum = 0;

// int n = 0;

// int ret = 1;

// scanf("%d", &n);//3

// for (i = 1; i <= n; i++)

// {

// int j = 0;

// for (j = 1; j <= i; j++)

// {

// ret \*= j;

// }

// sum += ret;

// }

// printf("%d\n",sum);

// return 0;

//}

//调试后

//#include <stdio.h>

//int main()

//{

// int i = 0;

// int sum = 0;

// int n = 0;

//

// scanf("%d", &n);//3

// for (i = 1; i <= n; i++)

// {

// int j = 0;

// int ret = 1;

// for (j = 1; j <= i; j++)

// {

// ret \*= j;

// }

// sum += ret;

// }

// printf("%d\n", sum);

// return 0;

//}

//#include <stdio.h>

//int main()

//{

// int i = 0;

// int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };

// //0-9

// for (i = 0; i <= 12; i++)

// {

// arr[i] = 0;

// printf("hehe\n");

// }

// //代码运行的结果是什么？

// //无线循环打印hehe

//

//}

//i和arr是局部变量

//局部变量是放在栈区上的

//栈区内存的使用习惯是：先使用高地址空间，再使用低地址空间

//栈区规定上面是高地址

//注意：栈区和数据结构栈没有关系，不要搞混

//越界访问程序会报错。

//c陷阱与缺陷

/\*优秀的代码

1.代码运行正常

2.bug很少

3.效率高

4.可读性高

5.可维护性高

6.注释清晰

7.文档齐全

\*/

/\*

常见的coding技巧

1.使用assert

2.尽量使用const

3.养成良好的编码风格

4.添加必要的注释

5.避免编码的陷阱

\*/

//预防错误发生

//通过编码技巧，减少错误

//示范：

//模拟实现库函数strcpy

#include <stdio.h>

#include <string.h>

//写的不好

//void my\_strcpy(char \*Destination,char \*source)

//{

// while (\*source!='\0')

// {

// \*Destination = \*source;

// Destination++;

// source++;

// }

// \*Destination = \*source;

//}

//修改版本1

//void my\_strcpy(char \*Destination,char \*source)

//{

// while (\*source!='\0')

// {

// \*Destination++ = \*source++;

// }

// \*Destination = \*source;

//}

//修改版本2

//void my\_strcpy(char \*Destination, char \*source)//

//{

// while (\*Destination++ = \*source++)//注：'\0'本意（ASCII的值为0）就是0，当0赋值给Destination后，整个表达式的值就是0，条件为假

// {

//

// }

//}

//修改版本3-最优版本

//#include <assert.h>

//void my\_strcpy(char \*Destination, const char \*source)//假设source和Destination写反了 加一个const \*source就是不可变的

//{

// assert(source != NULL);//断言 表达式为真断言成功 为假断言失败会进行报错，提示错误出现在哪里

// //不希望source为空指针，否则会提示问题出现在哪里

// while (\*Destination++ = \*source++)//如果source接受到的是空指针，无法解引用会引起程序崩溃

// {

//

// }

//}

//int main()

//{

// char arr1[20] = "xxxxxxxxxxx";

// char arr2[ ] = "hello";

// my\_strcpy(arr1,arr2);//假设起始源地址传递的空指针

// //strcpy(arr1, arr2);

// printf("%s\n", arr1);

// return 0;

//}

//const分析 const修饰变量 ---常变量 ：本质上是变量但是不能修改

int main()

{

int num = 10;

int n = 1;

// const int \*p = &num;//const如果放在\*左边修饰的\*p，表示指针所指向的内容是不能通过指针来改变的

////但是指针变量本身是可以修改的

// \*p = 20;//不能改变

// p = &n;

//printf("%d\n");

int \* const p = &num;//const如果放在\*右边边修饰的p，表示指针变量是不能改变的

//但是指针所指向的内容是可以修改的

\*p = 20;

p = &n;//不能改变

printf("%d\n");

return 0;

}