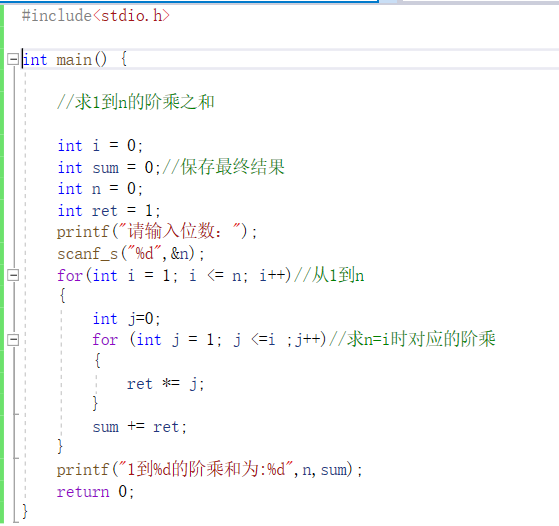
2021/9/8

**1.初识代码调试：**

原代码1：

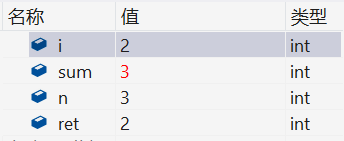


发现结果不对，开始调试（假设n=3）：

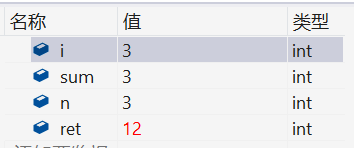
当i=1时，可以看到运算结果是对的



继续i=2：



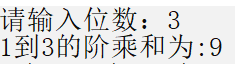
i=3时：





可以看到当n=3时，则出了问题；

我们可以看到，当i=3循环结束后，ret的值等于12，说明ret的那条语句出现了错误，而我们又可以很快地观察到在i=1和i=2的循环开始之前，ret=1，所以结果正确;而在i=3的循环开始之前，ret=2，导致了结果偏大，所以根据调试结果可以知道，在每次内循环结束后，将ret的值重新赋为1，即可得到正确结果。



**2.字符串的拷贝：**

#include<stdio.h>

//将arr2中的字符串拷贝给arr1

void my\_copy(char\* dest,char\* src) {

while (\*src!='\0') {

\*dest = \*src;

src++;//指针地址++，向后移一位

dest++;

}

\*dest = \*src; //拷贝arr2中的’\0’

}

int main() {

char arr1[] = "##########";

char arr2[] = "hello bit";

my\_copy(arr1,arr2);//分别将arr1和arr2的首地址传进去

printf("%s",arr1);

return 0;

}

发现while循环体内可以优化：

while (\*src!='\0') {

\*dest++ = \*src++;//++放在后面，代替了之前的三行代码

}

\*dest = \*src; //拷贝arr2中的’\0’

但是发现拷贝字符串和\0，我写了两条语句（都是拷贝，怎么就不能写在一起呢）

于是又进行了优化：

void my\_copy(char\* dest,char\* src) {

while (\*dest++ = \*src++)//括号内的值为每次拷贝进去的内容，依次为：'h','e','l','l',

'o',' ','b','i','t',在拷贝完't'之后，指针++，src指向arr2[]的'\0',再次进入循环，将\0拷贝给arr1[]，表达式的值为'\0'，跳出循环。

{

;

}

}

但是如果不小心传错了，传进去的是空指针呢，发现并不会进行拷贝，也不会报错，引入assert函数，当传进去的是空指针的话，报错。

void my\_copy(char\* dest,char\* src) {

assert(dest!=NULL);

assert(src != NULL);

while (\*dest++ = \*src++)

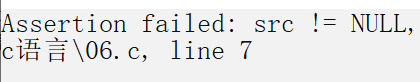
{

;

}

}

当传入空指针，报错：



**3.Const的用法：**

// const int\* p = &num;// const放在指针变量\*的左边时，修饰的是\*p，就不能通过p来改变\*p的值

int\* const p = &num; //const放在指针变量\*的右边时，修饰的是p，能通过p来改变\*p的值

\*p = 20; //报错。

**4．实现strlen函数的功能：**

#include<stdio.h>

#include<assert.h>

int my\_strlen(const char\* str) {//用const修饰\*str，即arr[]的具体内容，使得其不会改变

assert(str!=NULL);//保证指针有效，不为空

int count=0;

while (\*str != '\0') {

count++;

str++;

}

return count;

}

int main() {

char arr[] = "abcdef";

printf("%d", my\_strlen(arr));

return 0;

}