**使用Check Function定位出错代码**

|  |  |
| --- | --- |
|  | 作者：李坤峰  职务：助理技术支持工程师  邮箱：support@beckhoff.com.cn  日期：2021-11-9 |
| **摘 要：**  本文介绍了如何使用Check Function检测代码错误并且定位出错代码，并且以CheckBounds为例说明了Check Function的扩展功能。  TwinCAT3 CheckBounds 标越界 除零 子集 | |
| **附 件：**   |  |  |  | | --- | --- | --- | | 序 号 | 文件名 | 备注 | |  |  |  | |  |  |  | |  |  |  | |  |  |  | | |
| **历史版本：**   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | | |
| **免责声明：**  我们已对本文档描述的内容做测试。但是差错在所难免，无法保证绝对正确并完全满足您的使用需求。本文档的内容可能随时更新，如有改动，恕不事先通知，也欢迎您提出改进建议。 | |
| **参考信息：** | |

目 录

[1. 软硬件版本 4](#_Toc88487582)

[1.1. 硬件 4](#_Toc88487583)

[1.2. 控制软件 4](#_Toc88487584)

[2. 准备工作 4](#_Toc88487585)

[3. Bound Checks 4](#_Toc88487586)

[3.1. 如何添加CheckBounds Function 4](#_Toc88487587)

[3.2. CheckBounds使用 5](#_Toc88487588)

[3.3. 定位出错代码 7](#_Toc88487589)

[4. Division Checks 10](#_Toc88487590)

[4.1. 如何添加CheckDivDint Function 10](#_Toc88487591)

[4.2. CheckDivDint的使用 11](#_Toc88487592)

[4.3. 定位错误代码 13](#_Toc88487593)

[5. Range Checks与LRange Checks 13](#_Toc88487594)

[5.1. 如何添加CheckRangeSigned Function 13](#_Toc88487595)

[5.2. CheckRangeSigned的使用 14](#_Toc88487596)

[5.3. 定位出错代码 16](#_Toc88487597)

[6. Pointer Checks 17](#_Toc88487598)

[6.1. 如何添加CheckPointer Function 17](#_Toc88487599)

[6.2. CheckPointer的使用 18](#_Toc88487600)

[6.3. 定位出错代码 20](#_Toc88487601)

[7. 进阶使用 21](#_Toc88487602)

# 软硬件版本

## 硬件

编程电脑一台

## 控制软件

编程电脑是基于TwinCAT 3.1 Build 4024.12版本

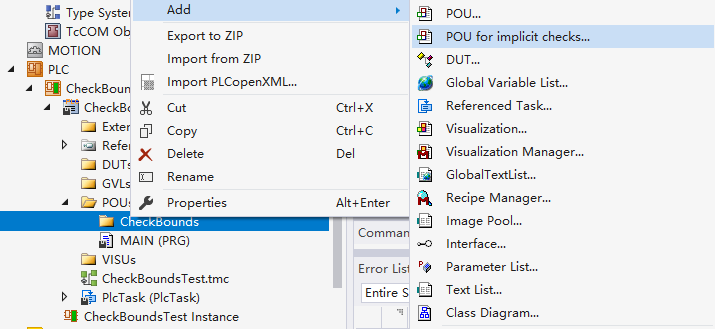
# 准备工作

新建TwinCAT XAE Project 并且在PLC下新建PLC Project。

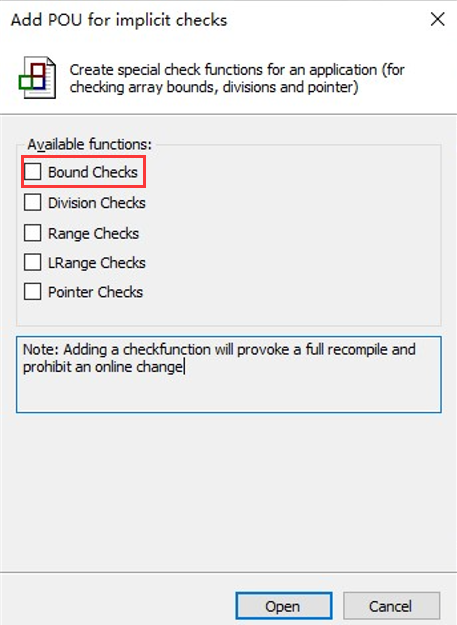
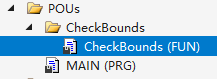
# Bound Checks

## 如何添加CheckBounds Function

我们建立好PLC Project之后，可以在项目下右键ADD，点击POU for implicit checks。进入ADD POU for implicit checks对话框。

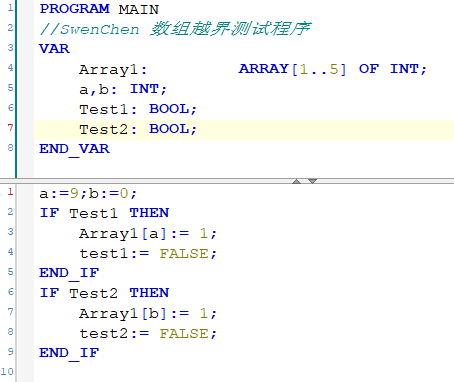


在这个对话框中我们可以选定需要创建的Check Function，本文档就以Bound Checks为例。选中Bound Checks后点击Open。

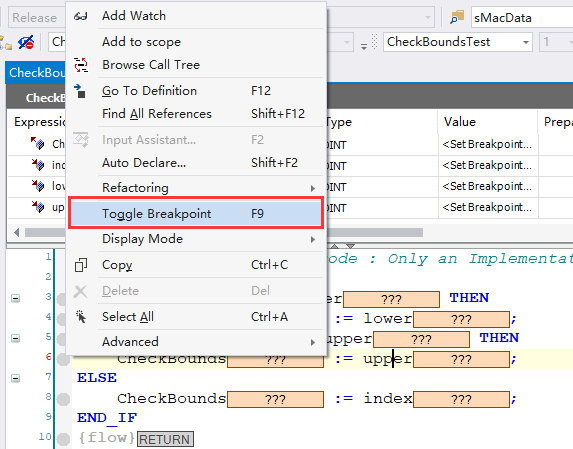
 

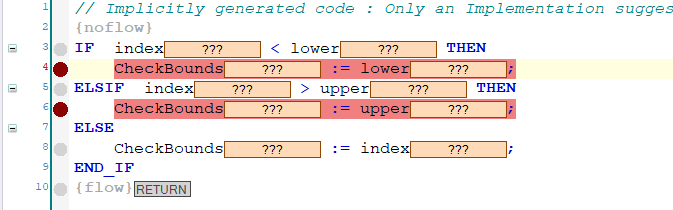
## CheckBounds使用

在创建完CheckBounds Function之后，我们到MAIN函数中编写测试程序，这里以下方的案例程序为例。这个程序意在使用CheckBounds功能检测数组的标越界错误。

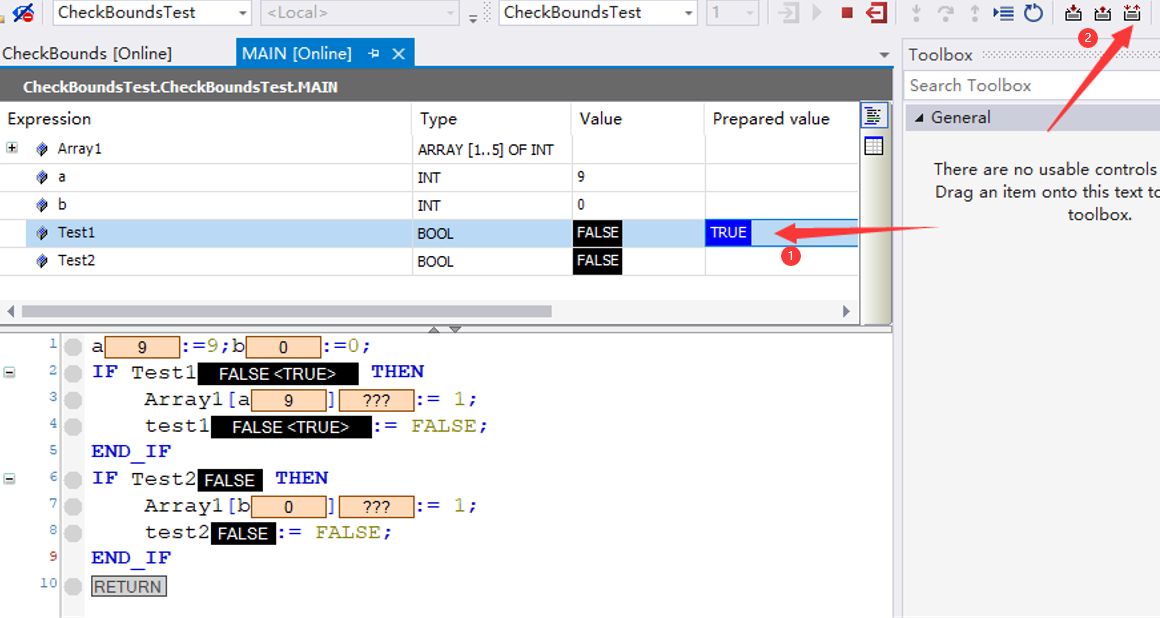


编写完MAIN函数之后，我们运行程序并在线。此时我们需要回到CheckBounds，对函数添加断点，使程序运行到断点停止。

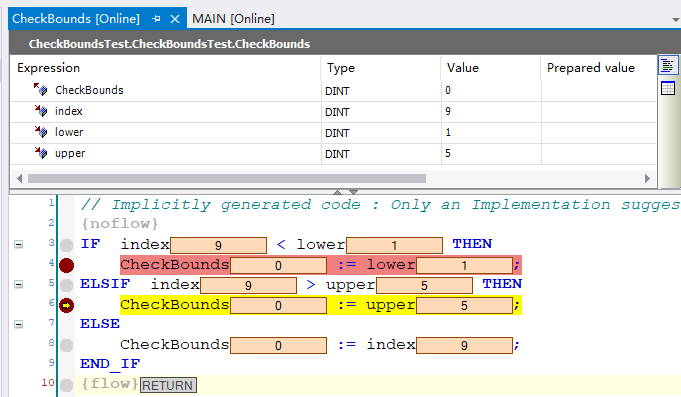




回到MAIN函数对功能进行测试，对Test1或者Test2写入值TRUE。



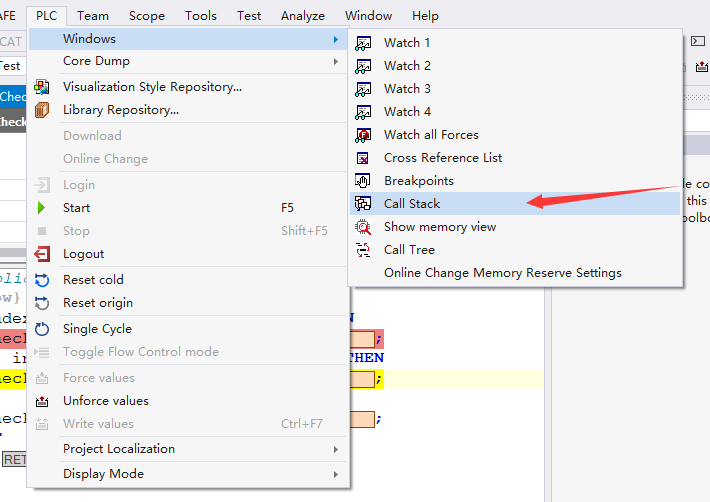
点击写入之后，CheckBounds检测到错误的发生，并且由于断点的存在，会自动回到CheckBounds的界面。通过黄色箭头我们可以看到，程序运行到CheckBounds := upper;处停止，说明写入超过的数组的上限。



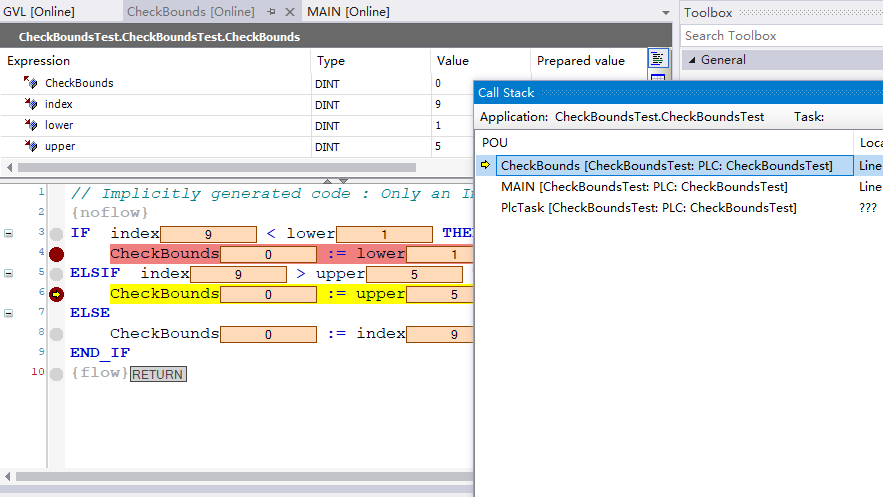
## 定位出错代码

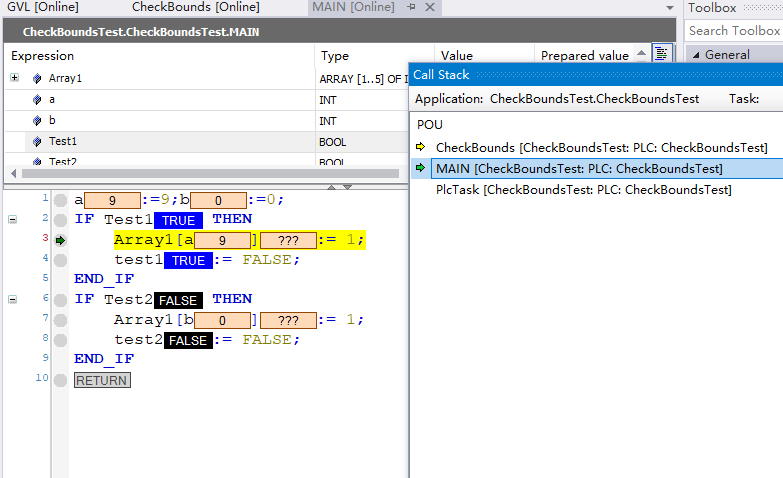
此时我们通过CheckBounds了解到有错误发生，那么我们应该如何精确定位到是哪一段程序发生到错误呢？

我们保持这样的状态之后，在菜单栏处找到PLC，在Windows子菜单中点击Call Stack。

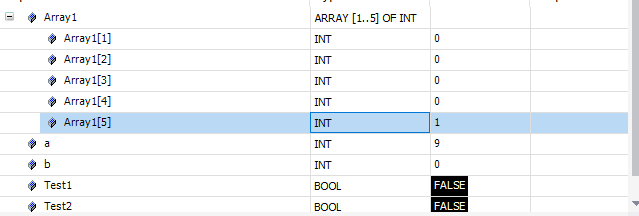


在弹出的窗口中可以看到，第一行就是CheckBounds停止运行的位置，那么我们双击第二行就可以定位到在MAIN函数中对应的出错程序。

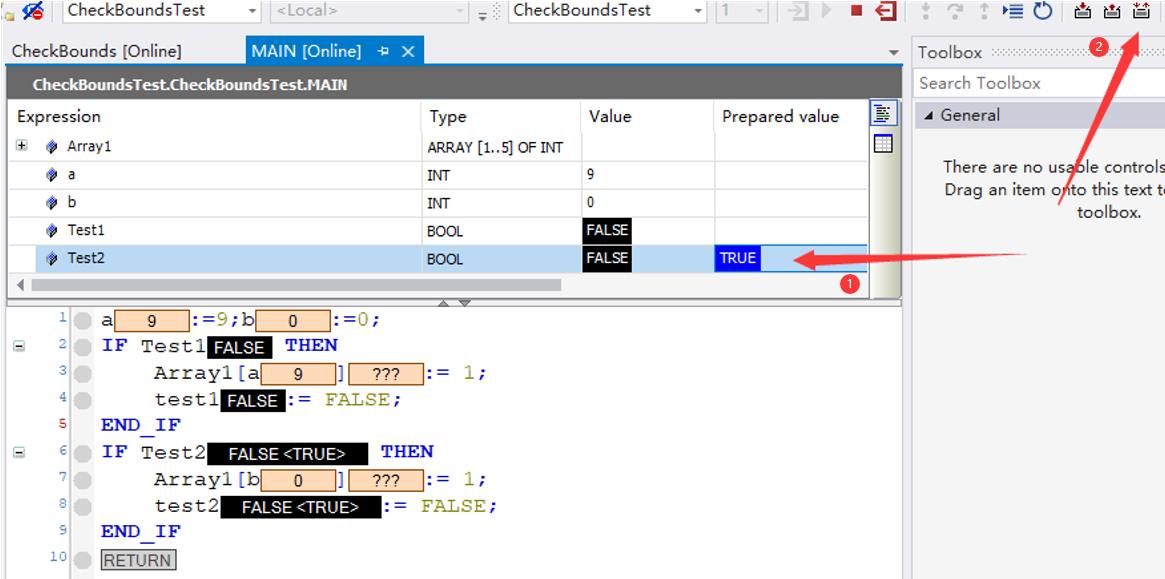


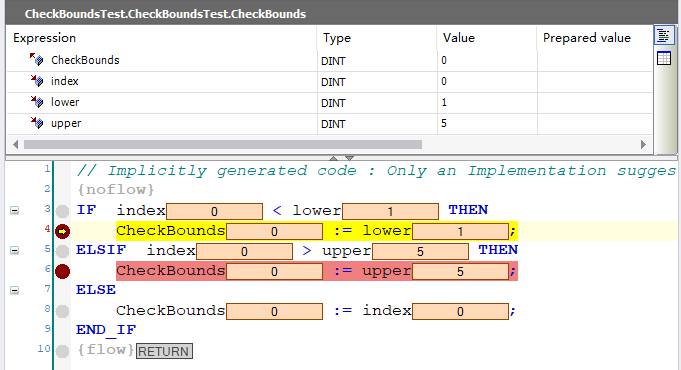


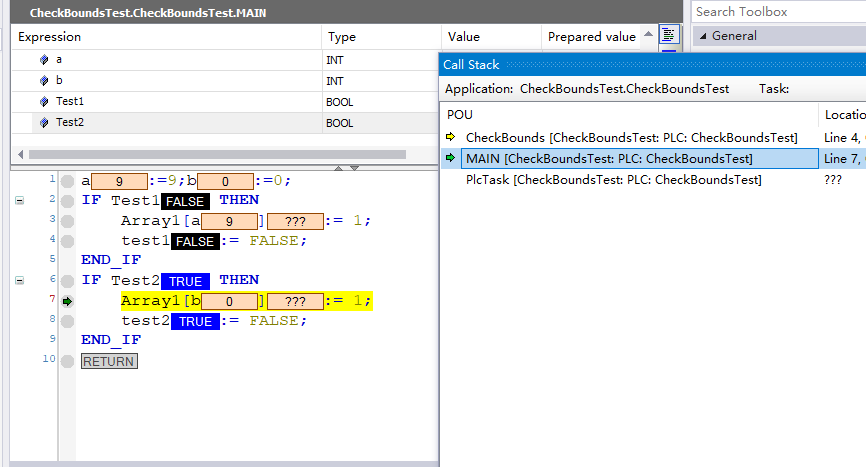
回到MAIN函数，重新点击运行之后我们可以看到Array[5]的值变为了1，说明CheckBounds将值分配给了仍然有效的上边界的Array[5]。

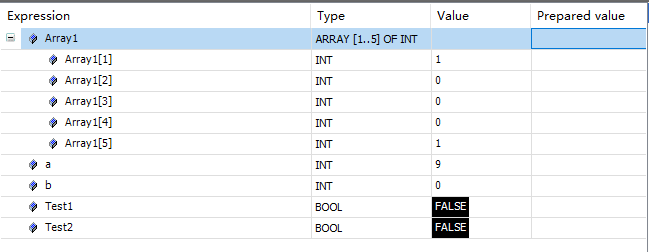


同理我们可以对Test2写入TRUE，测试CheckBounds对下标越界错误的检测。并且定位到错误发生的程序。





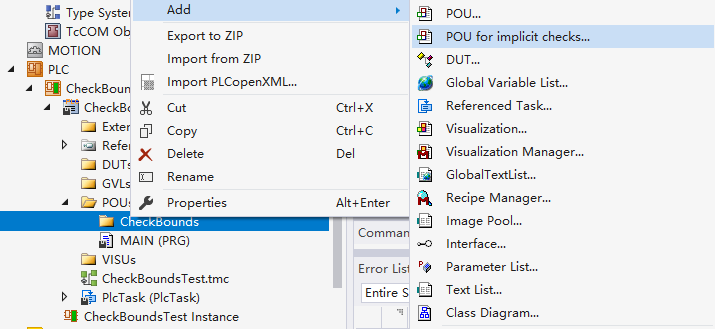




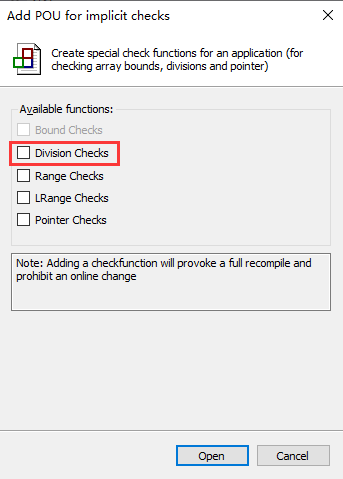
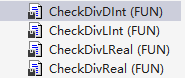
# Division Checks

## 如何添加CheckDivDint Function

添加CheckDivDint Function的方法与CheckBounds Function的添加方式类似。



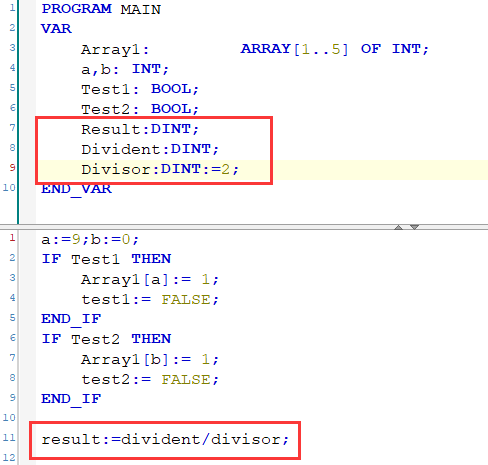
在对话框中选中Division Checks，点击Open后就可以看到添加了四个Function。我们分别查看四个Function的代码部分可以看到他们的代码部分其实都是相同的，他们之间的不同是检测的变量类型各有不同。

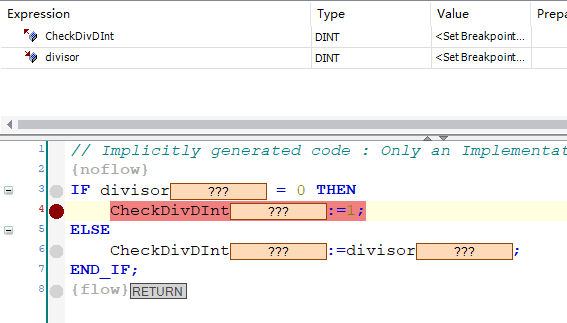
## CheckDivDint的使用

我们以CheckDivDint为例介绍Division Checks的功能和使用。首先我们在MAIN函数中声明并加入程序。由于CheckDivDint检测的是Dint类型的变量，我们声明的变量也应该为Dint类型。

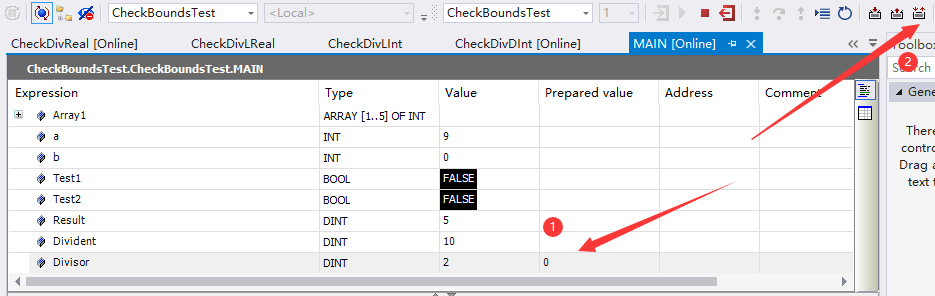
此程序旨在使用CheckDivDint检测0为除数的错误。



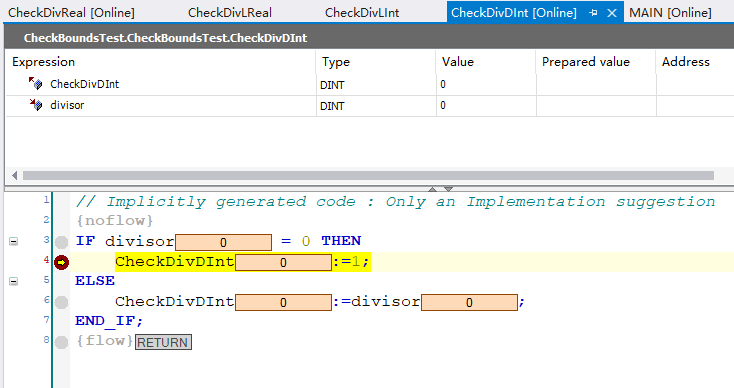
编写完MAIN函数之后，我们运行程序并在线。此时我们需要回到CheckDivDint，对函数添加断点，使程序运行到断点停止。

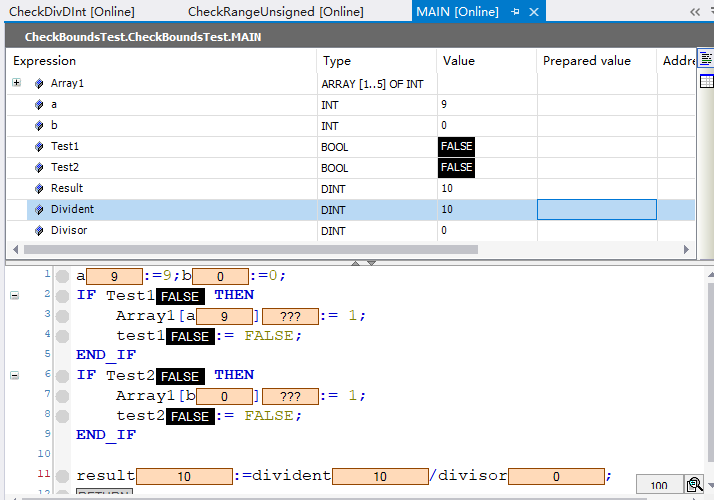


回到MAIN函数对功能进行测试，我们先对Divident写入数值10，在观察到result的值变为5之后，对Divisor写入数值0。



点击写入之后，CheckDivDint检测到错误的发生，由于断点的存在，我们会自动跳回到CheckDivDint的界面。



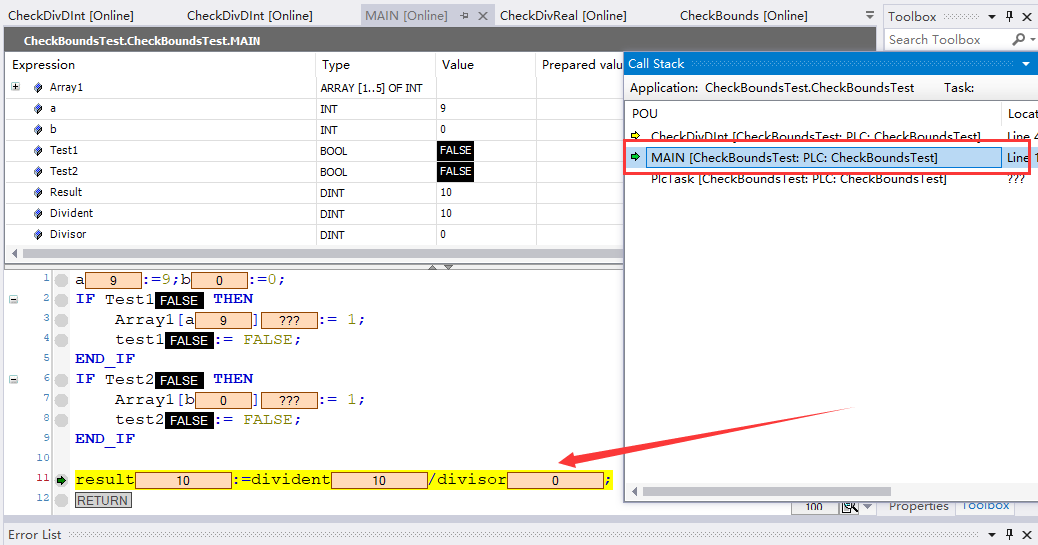


以上就是以CheckDivDint为例的Division Checks功能使用，其他三个Function的使用也与其类似，大家可以自己测试。

## 定位错误代码

在检测到错误的发生之后，我们需要定位出错代码，方法也与CheckBounds部分的定位出错代码的方法相同。

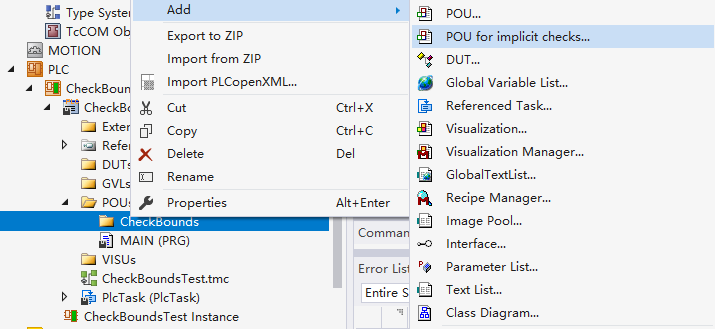
我们保持这样的状态之后，在菜单栏处找到PLC，在Windows子菜单中点击Call Stack。在弹出的窗口中可以看到，第一行就是CheckDivDint停止运行的位置，那么我们双击第二行就可以定位到在MAIN函数中对应的出错程序。



# Range Checks与LRange Checks

## 如何添加CheckRangeSigned Function

添加CheckRangeSigned Function的方法与CheckBounds Function的添加方式类似。



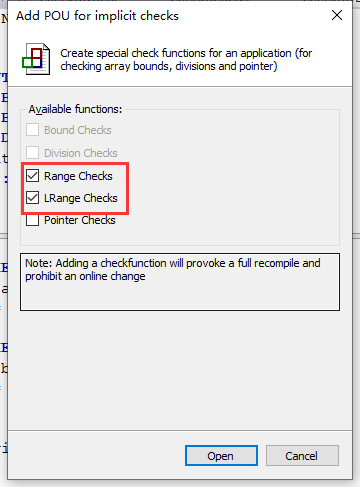
在对话框中选中Range Checks和LRange Checks，点击Open后就可以看到添加了四个Function。我们分别查看四个Function的代码部分可以看到他们的代码部分其实都是相同的，他们之间的不同是检测的变量类型各有不同。

CheckRangeSigned检测的子集范围类型为SINT, INT, DINT。

CheckRangeUnsigned检测的子集范围类型为BYTE, WORD, DWORD, USINT, UINT, UDINT。

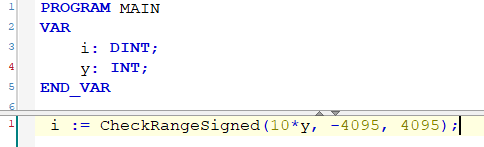
CheckLRangeSigned检测的子集范围类型为LINT。

CheckLRangeUnsigned检测的子集范围类型为LWORD, ULINT。

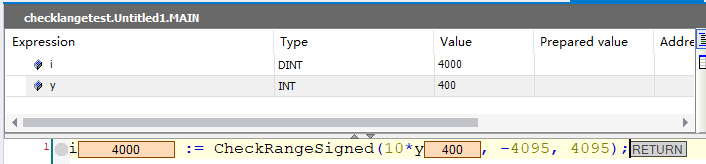
## CheckRangeSigned的使用

我们以CheckRangeSigned为例，介绍Rangle Checks及LRangle Checks功能的使用。首先我们在MAIN函数中声明并添加程序。

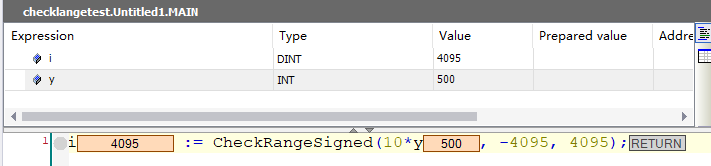


此程序是对i:=10\*y的变形，意在规定i的范围，使它不超过(-4095,4095)。

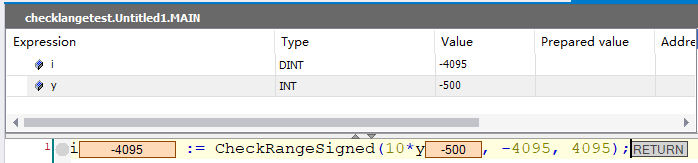
激活配置后运行程序，我们对y进行赋值验证程序



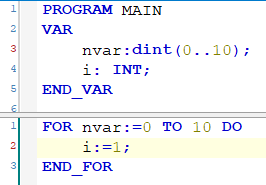
当y等于400时，i等于4000，这时i的值还没有超出范围。



当y等于500时，按照运算关系i的值本应该是5000，但是我们看到此时i的值为4095，也就是我们规定范围的上限，这就是CheckRangeSigned的作用。

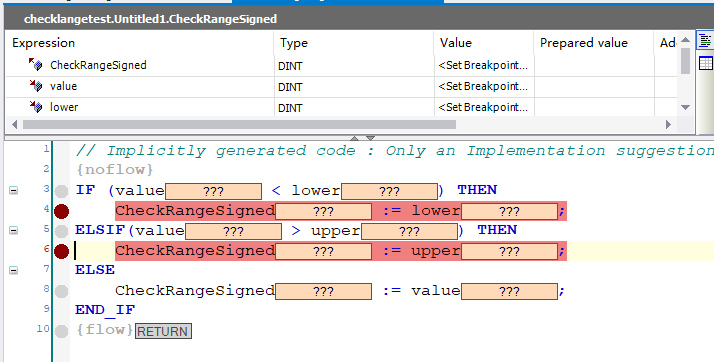


接下来的程序是CheckRangeSigned的另一个使用方法。

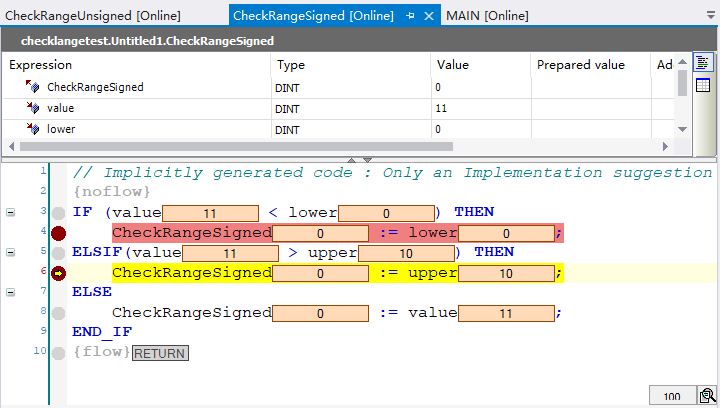


此程序在定义nvar变量时将它范围定义为(0,10)，但是在程序中他的值却会达到11，这会造成循环程序和周期的出错，而使用CheckRangeSigned可以将nVar的值限制在范围之内。

激活配置并运行程序，首先我们需要在CheckRangeSigned的程序中添加断点。

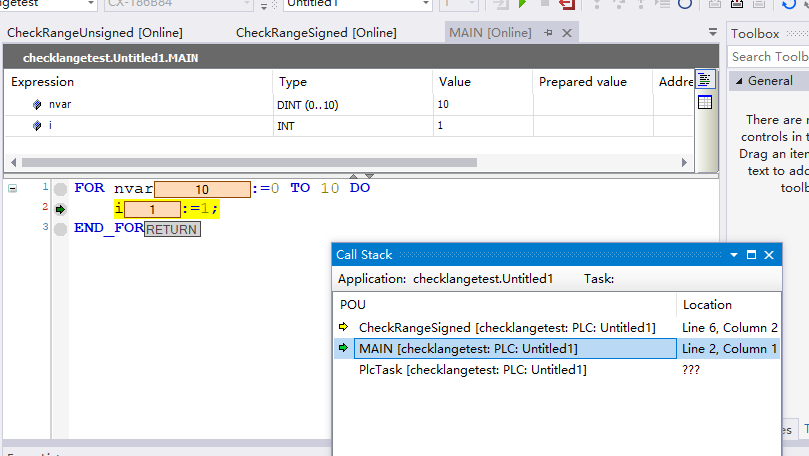


点击运行之后，由于断点的存在程序停止运行，此时程序运行到CheckRangeSigned:=upper处，说明nVar的值超过了上限，回到Main函数界面我们可以看到nVar的值为10。



## 定位出错代码

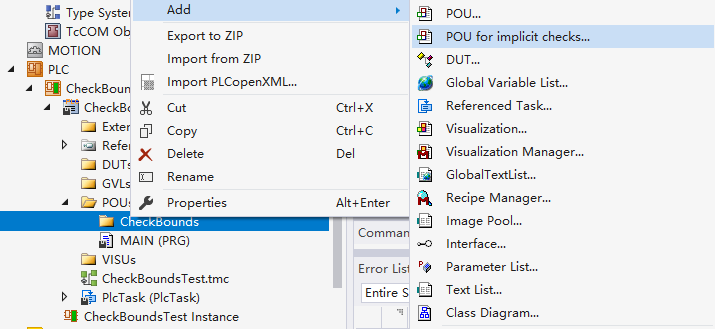
CheckRangeSigned定位出错代码的方法与之前提到的方法一致，在这里就不过多阐述。



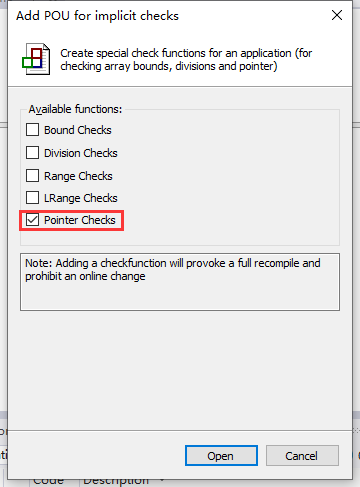
# Pointer Checks

## 如何添加CheckPointer Function

添加CheckPointer Function的方法与CheckBounds Function的添加方式类似。

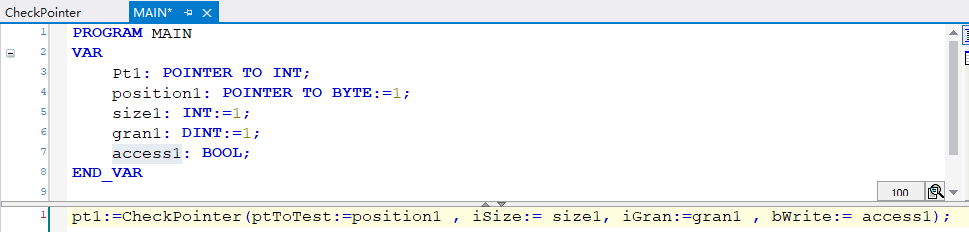


在对话框中选中Pointer Checks点击Open。

## CheckPointer的使用

在创建完CheckPointer Function之后，我们在MAIN函数中编写程序。

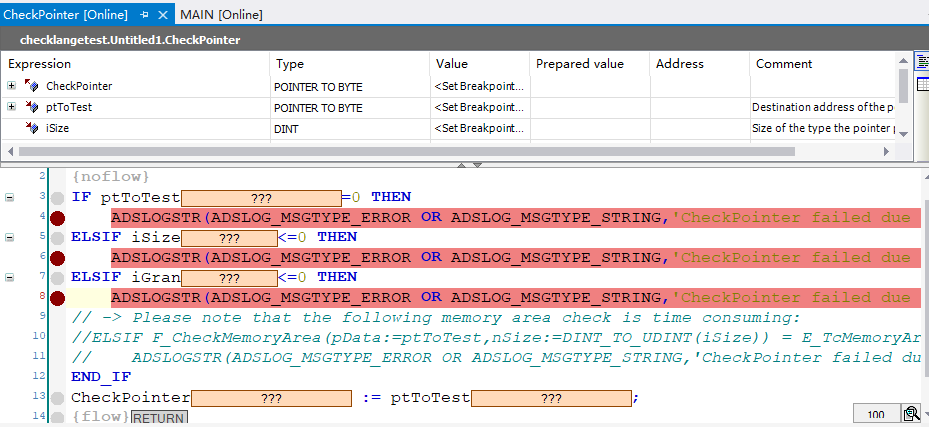


在CheckPointer中我们可以限定指针的目标地址，引用变量的字节大小和粒度以及其访问类型。

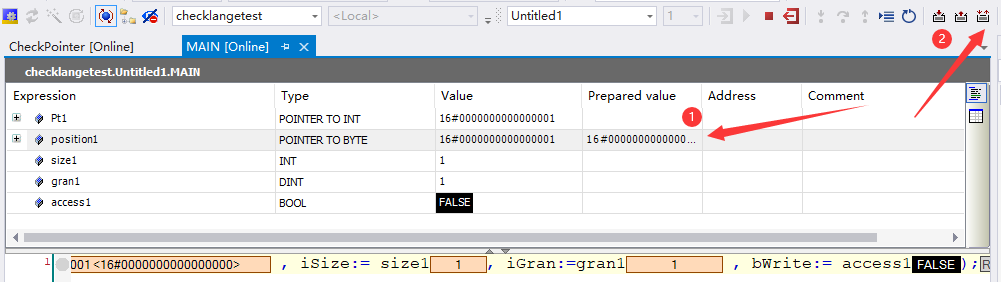
当bWrite为TRUE时为写访问，FALSE为读访问。

本小节以地址错误为例进行演示，更多的错误类型大家可以自行尝试。

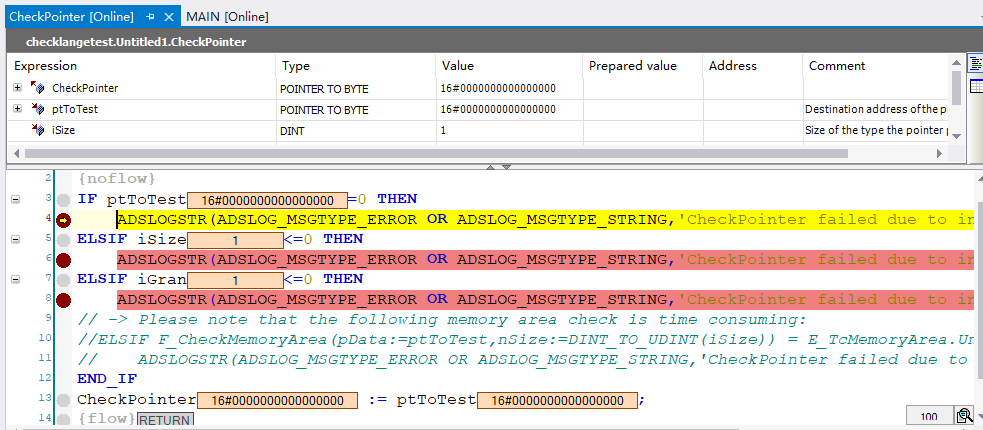
编写好程序之后激活配置运行程序，我们首先需要到CheckPointer中添加断点。



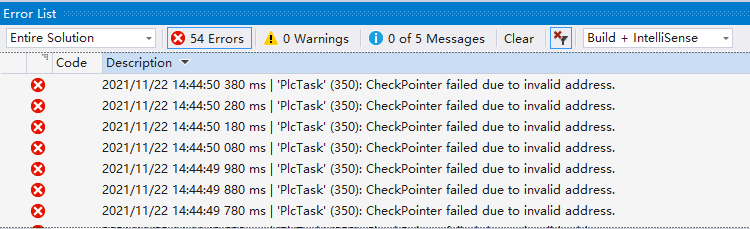
添加完成之后，在MAIN函数的监视窗口中对几个变量的数值进行修改。此处以Position1为例。



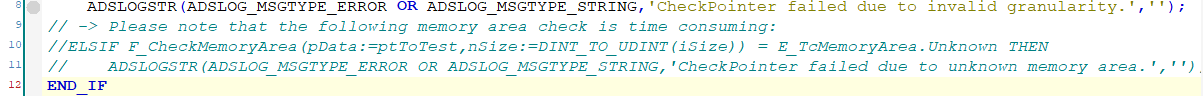
在准备值中填入0，点击写入后，由于断点的存在，程序会自动停止运行，并且界面会回到CheckPointer中。



由此我们可以观察到，程序运行到此处停止，说明出现了无效地址，校验指针失败。并且由于ADSLOGSTR的存在，如果我们再次点击运行程序，就可以在信息栏发现报错信息。或者在没有添加断点的情况下，程序每运行一个周期，信息栏都会显示一条报错信息，经过查阅后，我们也能发现出现了问题和问题类型及原因。

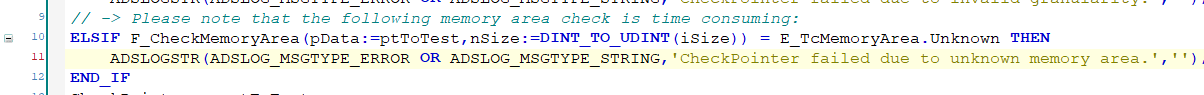


对于无效的大小和无效的粒度的检测方法及现象，都与上述无效地址类似，大家可以自行测试。 我们注意到，在CheckPointer中还有一个隐藏的检测功能，它默认被注释隐藏起来。



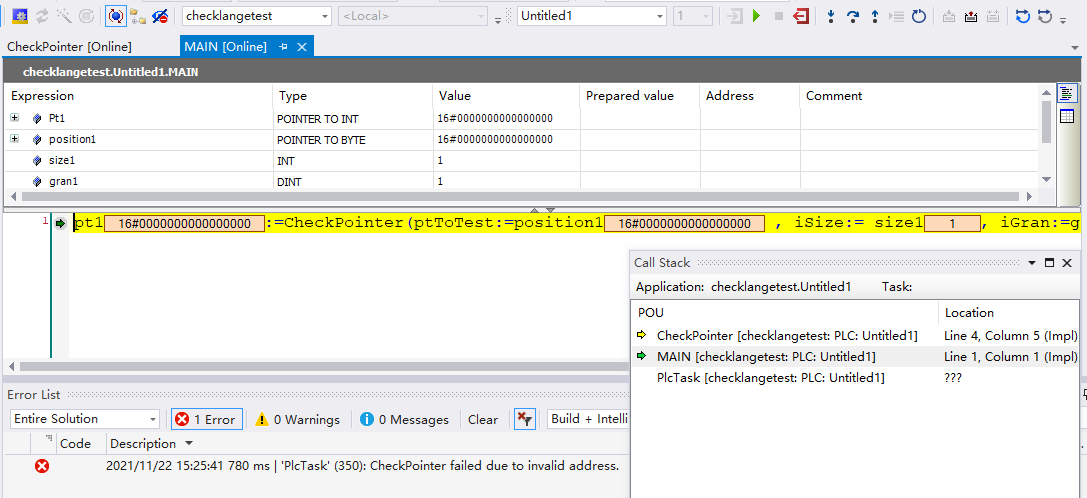
在注释中我们可以看到一则注意事项“请注意，下面的内存区域检查非常耗时”，因此这一功能大家需要谨慎使用。

将两行程序前的//删去，启用这一功能。在这里我就不做演示，大家可以在实际应用中自行启用并测试。



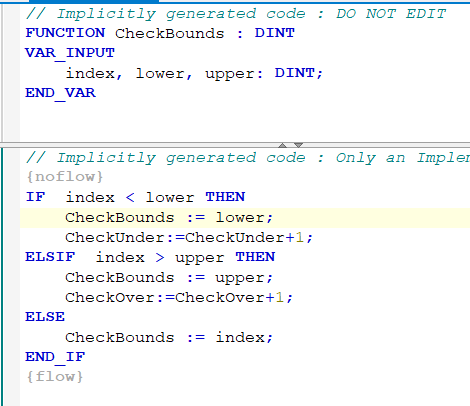
## 定位出错代码

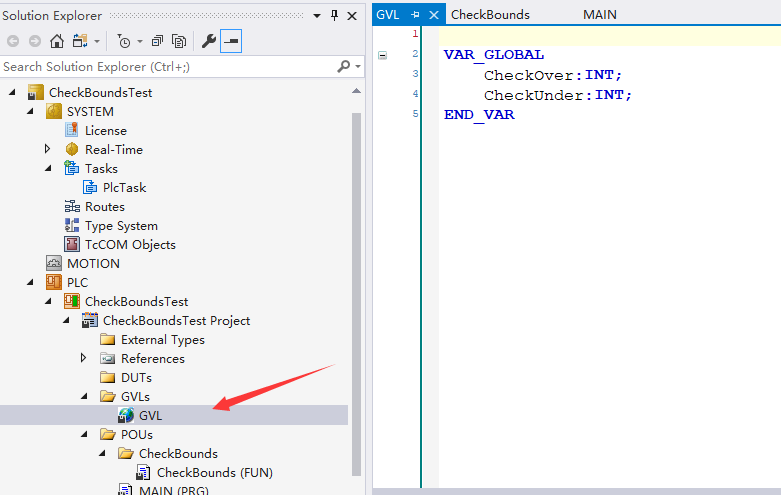
CheckPointer定位出错代码的方法与之前提到的方法一致，在这里就不过多阐述。



# 进阶使用

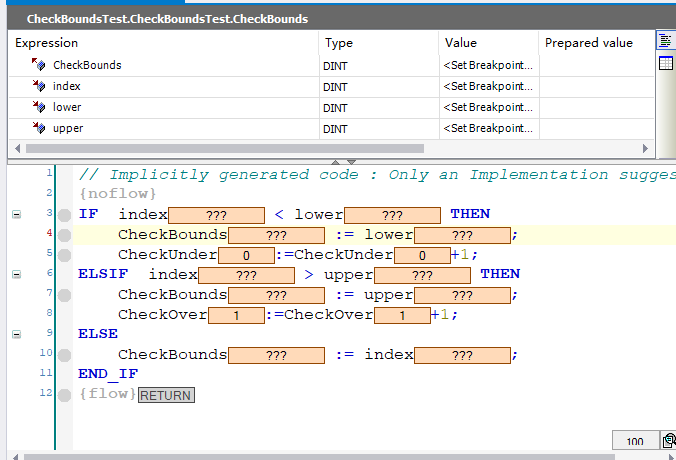
以CheckBounds为例，我们可以对CheckBounds Function中的程序进行更改，来达到一些扩展功能。**注意不要删减原有的程序。**





以上述这个程序为例，我们可以在程序中添加计数的程序，并且将这两个变量声明在GVL 中。这样就可以直接通过变量观察到错误的发生，不再需要添加断点。

我们给Test1写入TRUE之后，就可以直接在CheckBounds中观察到CheckOver数值的变化，判断出出现了上标越界的错误。但是想要定位到出错的程序位置还是需要通过上述的添加断点的方法定位。



更多的功能大家也可以自行编写。

**上海（ 中国区总部）**

中国上海市静安区汶水路 299 弄 9号（市北智汇园）

电话: 021-66312666

**北京分公司**

北京市西城区新街口北大街 3 号新街高和大厦 407 室

电话: 010-82200036 邮箱: beijing@beckhoff.com.cn

**广州分公司**

广州市天河区珠江新城珠江东路16号高德置地G2603 室

电话: 020-38010300/1/2 邮箱: guangzhou@beckhoff.com.cn

**成都分公司**

成都市锦江区东御街18号 百扬大厦2305 房

电话: 028-86202581 邮箱: chengdu@beckhoff.com.cn

|  |  |
| --- | --- |
| 请用微信扫描二维码  通过公众号与技术支持交流 | 倍福官方网站：  https://www.beckhoff.com.cn  在线帮助系统：  https://infosys.beckhoff.com/index\_en.htm |
| 倍福虚拟学院：  https://tr.beckhoff.com.cn/ |
| 招贤纳士：job@beckhoff.com.cn  技术支持：support@beckhoff.com.cn  产品维修：service@beckhoff.com.cn  方案咨询：sales@beckhoff.com.cn |