



LDRA Testbed

入门使用指南

上海创景计算机系统有限公司

www.visionmc.com

LDRA Testbed.....	1
一. 源代码文件 (Source Files)	1
1.1 概述.....	1
1.1.1 基于单个文件的分析.....	1
1.1.2 基于多个文件的分析.....	1
1.2 对源代码的要求.....	1
二. Testbed图形用户界面概述	2
2.1 图形用户界面.....	2
2.2 Testbed菜单	3
2.3 快捷键.....	3
三. Testbed和编译器设置	4
3.1 以命令行使用方式安装编译器.....	4
3.1.1 在Windows9x上安装VC++	4
3.1.2 在WindowsNT, 2000 和XP上安装VC++.....	4
3.2 测试编译器安装是否正确.....	5
3.3 Borland公司的编译器.....	6
3.3.1 太多的警告产生的错误.....	6
3.3.2 DOS命名问题.....	6
3.3.3 输出名设置.....	7
四. 分析单个文件 (GUI)	8
4.1 运行Testbed	8
4.2 选择源文件testrian.c/cpp	8
4.3 删除源文件已经存在的工作文件.....	9
4.4 Testbed向导对话框	9
4.5 检查分析范围设置.....	10
4.6 质量模型设置.....	11
4.7 选择分析选项.....	12
4.8 清除当前的选择.....	12
五. 主要静态分析.....	13
5.1 运行静态分析并察看结果.....	13
5.1.1 图形化显示分析结果.....	13
5.1.2 文本显示分析结果.....	15
六. 复杂度分析.....	18
6.1 运行复杂度分析并察看结果.....	18
6.1.1 图形化显示分析结果.....	18
6.1.2 文本显示分析结果.....	20
七. 静态数据流, 交叉索引, 信息流和数据对象分析.....	22
7.1 运行各项分析.....	22
7.2 察看分析结果.....	23
7.2.1 察看静态数据流分析结果.....	23
7.2.2 察看交叉索引的结果.....	24
7.2.3 察看信息流分析结果.....	24
7.2.4 察看数据对象分析结果.....	25

7.2.5 察看质量报告.....	25
7.2.6 察看其他分析结果.....	26
八. 动态分析.....	28
8.1 进行动态分析.....	28
8.2 选择执行插装程序命令.....	29
8.3 选择动态覆盖率分析选项.....	30
8.4 执行分析.....	31
8.5 执行插装程序.....	31
九. 深层次的动态分析.....	34
9.1 再次执行插装后的程序.....	34
十. 以集 (set) 的方式进行分析	37
10.1 设置集属性.....	37
10.2 往集里添加文件.....	38
10.3 集的分析及结果察看.....	39
十一. 附注：数据流分析.....	40
十二. 附注：信息流分析.....	42
十三. 分析自己的代码.....	44
13.1 概述.....	44
13.2 基本规则.....	44
13.3 分析范围.....	44
13.4 编译插装后的代码.....	46
13.4.1 概述.....	46
13.4.2 初步.....	46
13.4.3 自动过程.....	46
13.4.4 进一步.....	47

一. 源代码文件 (Source Files)

1.1 概述

LDRA Testbed 既能分析单个的文件也能通过“集”(set)的方式同时分析多个文件。

1.1.1 基于单个文件的分析

本使用指南将以如何使用 Testbed 来分析示例程序 `testrian.c/cpp` 为中心，来介绍 Testbed 的基本功能。这个示例程序很简单，因此很容易对其进行分析，作为示范，它能很快的让用户对 Testbed 的广泛的功能有一个直观的了解。

`testrian.c/cpp` 这个程序的功能是，让用户输入三个整数，然后判断以这三个整数为三边能否构成一个三角形；这个程序中存在问题缺陷，这些在 Testbed 的分析结果中能反映出来。这样初级用户就能够通过对这个例子的分析对软件的使用和作用有一个快速的了解和掌握。

1.1.2 基于多个文件的分析

除了能分析单个源文件外，Testbed 还能以“集”(set)的方式对多个文件同时进行分析。一个 set 可以有下面两种模式：

- Group(default)
- System

Group 这种模式下，Testbed 对一个 set 中的文件进行相互独立的分析，这样对于一批文件就可以一次分析完，而不需要一个个的导入工具来进行分析。

System 这种模式下，Testbed 将这个 set 中的所有文件是作为一个整体来分析的，会给出这些文件内函数相互间的调用关系，变量引用等等结果，也就是将这些文件作为一个工程来分析。

1.2 对源代码的要求

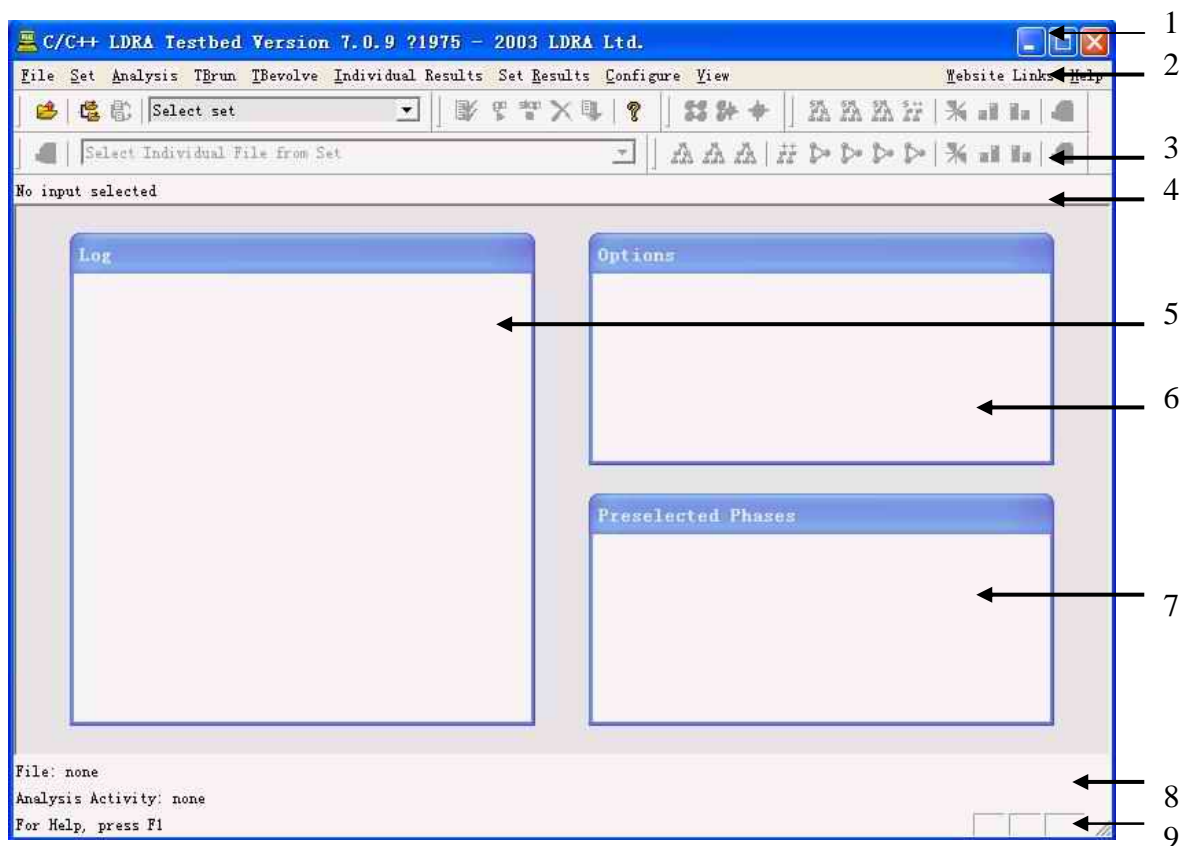
用 Testbed 进行分析的源代码要符合以下要求：

- 符合编程语言标准；
- 没有语法错误（编译能通过）；
- 如果要做动态分析的话，需要用户的程序能够运行；

二. Testbed 图形用户界面概述

2.1 图形用户界面

下面这个图将介绍 Testbed 图形用户界面分哪几个区域以及都包含那些内容：



(图 2-1)

1. **标题栏** 标题栏显示的是当前 Testbed 的版本信息和版权信息；
2. **菜单栏** 通过菜单栏用户可以点击下拉菜单来完成文件选取，分析，以及察看结果等个项功能；
3. **工具栏** 工具栏中是 Testbed 的一些常用功能的快捷按钮；
4. **输入源文件** 显示的是当前被选中的源文件或 set 的名字；如果正在分析一个 set，则当前正在被分析的单个文件的名字将被显示；
5. **Log 窗口** 这个窗口告诉用户工具当前正在进行什么分析操作；
6. **选项窗口** 这个窗口用来显示当前选择可哪些分析项；
7. **执行状态窗口** 这个窗口显示的是已经被执行了的分析项；
8. **分析状态栏** 这里显示的是当前正在执行的分析项；
9. **状态栏** 这里显示的是当前选中的菜单项的功能，也包快捷键信息；

2.2 Testbed 菜单

Testbed 的大多数命令都包含在了 Testbed 的下拉菜单中了。用户可以通过点击菜单的方式完成期望的操作。

注意：菜单中的某些选项是要在完成了其他的一些分析后才可用的；如果菜单中的某些选项始终是灰色的（不可用），这是由于您没有购买软件相应的功能模块。

2.3 快捷键

Testbed 的一些菜单选项可以通过快捷键来访问。相应的菜单的快捷键就是菜单名字中带下划线的字母加 **Alt** 键，下拉菜单中的选项的快捷键是相应的选项名字中带下划线的字母；如何我们要通过快捷键方式执行 **select** 命令着操作如下：

先按 **Alt+F** 键
再按 **S** 键

三. Testbed 和编译器设置

Testbed 可以在自己的集成环境下编译插装之后的程序，但是要求编译器能够在命令行模式下正常运行。

3.1 以命令行使用方式安装编译器

要正确的设置你的编译器，使其在 GUI 模式和命令行模式都能正常的使用，具体的设置请查看编译器的相关文档。

下面我们将介绍如何设置 MSVC 和 Borland 的编译器。

3.1.1 在 Windows9x 上安装 VC++

在 Windows95/98 的系统上设置 MSVC 编译器最好的办法是在你的 autoexec.bat 中加入一个调用。在“开始”菜单中点“运行”，然后输入 sysedit，在 autoexec.bat 中加入如下内容：

对于 MSVC4.0:

```
call c:\msdev\bin\vcvars32.bat
```

对于 MSVC5.0:

```
call C:\PROGRA~1\DEVSTU~1\VC\BIN\VCVARS32.BAT
```

对于 MSVC6.0:

```
call C:\PROGRA~1\MICROS~3\VC98\BIN\VCVARS32.BAT
```

3.1.2 在 WindowsNT, 2000 和 XP 上安装 VC++

Windows NT 系统的路径设置在系统环境变量中设置。Windows2000 的环境变量的设置为：开始—>控制面板—>系统—>高级—>环境变量。



MSVC 编译器在安装的时候会有对话框询问是否添加编译器的路径设置到系统环境变量，选择“是”就会自动完成系统环境变量的添加；如果没有自动添加，那么就需要手动添加。察看 vcvars32.bat 来确定需要添加哪些环境变量，下面是一个例子：

```
include
C:\Program Files\Microsoft Visual Studio\VC98\atl\include;C:\Program Files\Microsoft Visual
Studio\VC98\mf\include;C:\Program Files\Microsoft Visual Studio\VC98\include
lib
C:\Program Files\Microsoft Visual Studio\VC98\mf\lib;C:\Program Files\Microsoft Visual
Studio\VC98\lib
MSDevDir
C:\Program Files\Microsoft Visual Studio\Common\MSDev98
path
C:\Program Files\Microsoft Visual Studio\Common\Tools\WinNT;C:\Program Files\Microsoft
Visual Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual Studio\Common\Tools;C:
\Program Files\Microsoft Visual Studio\VC98\bin
```

请根据您的机器的相应路径进行设置。

3.2 测试编译器安装是否正确

打开一个 dos 窗口，输入编译命令，看能否得到正确的结果。下面是一个 VC 的例子：

```
C:\WINNT>cl
Microsoft ® 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright © Microsoft Corp 1984-1997. All rights reserved.
usage: cl [ option... ] filename... [ /link linkoption... ]
```

上面的结果说明 MSVC 编译器的路径设置正确，下面是设置不正确的情况：

```
C:\WINNT>cl
Bad command or file name
```

同时，您还需要设置编译器的 lib 和 include 相应环境变量，您可以用 set 命令察看您当前的设置：

```
C:\WINNT>set
TMP=C:\WIN95\TEMP
TEMP=C:\WIN95\TEMP
PROMPT=$p$g
winbootdir=C:\WIN95
COMSPEC=C:\WIN95\COMMAND.COM
MSINPUT=C:\PROGRA~1\MSINPUT
MSDEVDIR=C:\Program Files\Microsoft Visual Studio\Common\MSDev98
```



```
PATH=C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\Wbem;C:\Program
Files\Microsoft Visual Studio\Common\Tools\WinNT;C:\Program Files\Microsoft Visual
Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual
Studio\Common\Tools;C:\Program Files\Microsoft Visual Studio\VC98\bin
include=C:\Program Files\Microsoft Visual Studio\VC98\atl\include;C:\Program
Files\Microsoft Visual Studio\VC98\mfc\include;C:\Program Files\Microsoft Visual
Studio\VC98\include
LIB=C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib;C:\Program Files\Microsoft
Visual Studio\VC98\
windir=C:\WINNT
```

如果您已经完成了上面的设置，就可以用 LDRA Testbed 的一个例子程序来进行编译试验，如果一切正确的话，将会得到如下的结果：

```
C:\Testbed>cl testrian.c
Microsoft ® 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright © Microsoft Corp 1984-1997. All rights reserved.
testrian.c
Microsoft ® 32-Bit Incremental Linker Version 5.02.7132
Copyright © Microsoft Corp 1992-1997. All rights reserved.
out:testrian.exe
testrian.obj
```

3.3 Borland 公司的编译器

3.3.1 太多的警告产生的错误

Borland 的编译器编译时对警告数目是有限制的，当警告的数目超过规定的数目时就会产生编译报错。我们需要将编译器的警告数设到最大 255 个，命令行下的参数为：

bcc32 -g255

如果警告数目还是超出的话，我们可以设置关掉如下两个报警：

- Function should return a value
- Call to function with no prototype

相应的命令行下的参数是 **-wrvl -wpro**

3.3.2 DOS 命名问题

一些 Borland C++ 编译器不支持 DOS 8.3 文件名的限制。如果您遇到这样的问题请将 Testbed.in 中的 DOSNAMES 设置 TRUE。Testbed.ini 在您的 Windows 的安装目录下。

在 开始菜单->运行 下使用下面的命令来编辑 Testbed.ini 文件

```
%windir%/notepad testbed.ini
```

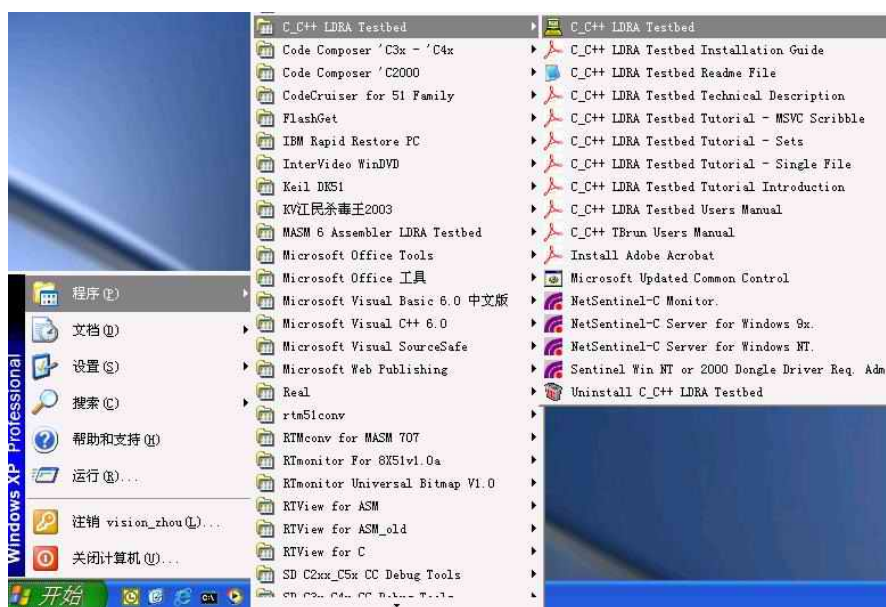
3.3.3 输出名设置

Borland 编译器在输出选项 `-o` 和输出文件名之间不能有空格。

四. 分析单个文件（GUI）

4.1 运行 Testbed

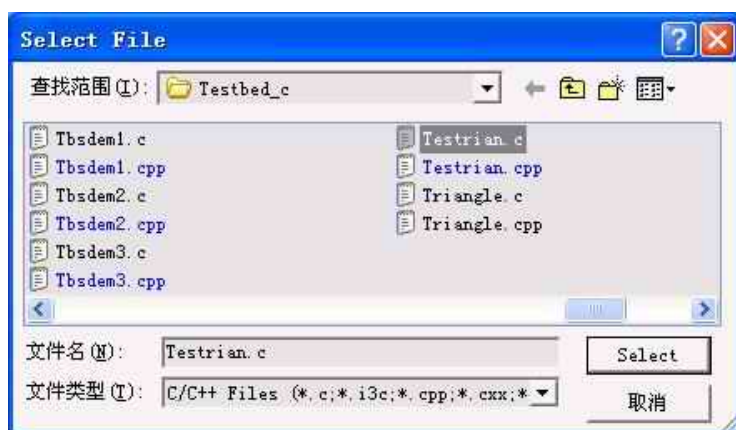
通过“开始”->“所有程序”->“C_C++ LDRA Testbed”->“C_C++ LDRA Testbed”来运行 Testbed。如下图：



Testbed 运行之后如图（图 2-1）。

4.2 选择源文件 testrian.c/cpp

在 **F**ile 菜单下选择 **S**elect File 选项，选择要分析的文件；

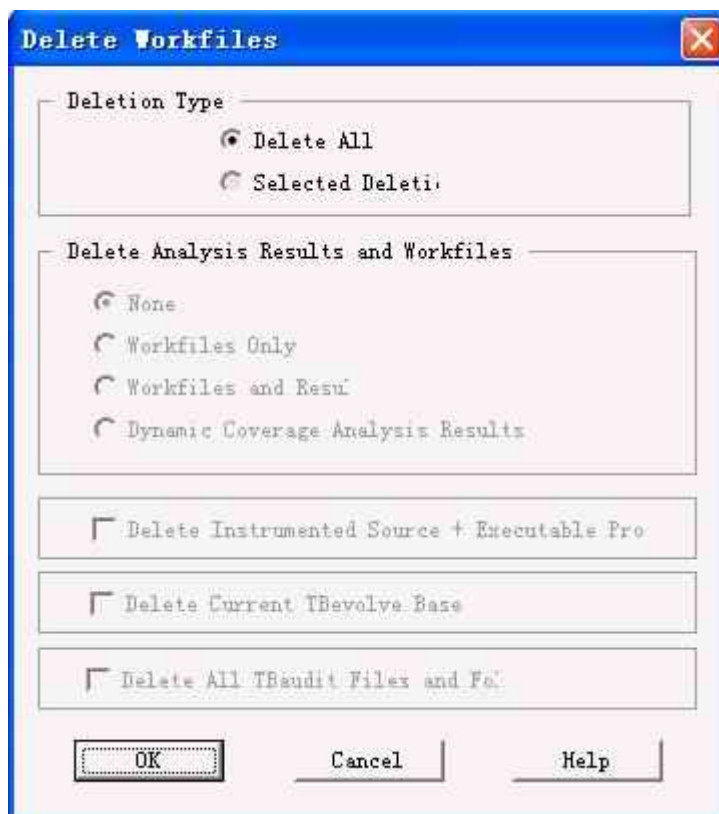


选择好要分析的文件，点击 Select 按钮完成。

4.3 删除源文件已经存在的工作文件

如果源文件 `testrian.c/cpp` 已经分析过了，相应的工作文件，结果文件已经存在了，请按照下面的步骤来删除这些工作文件。

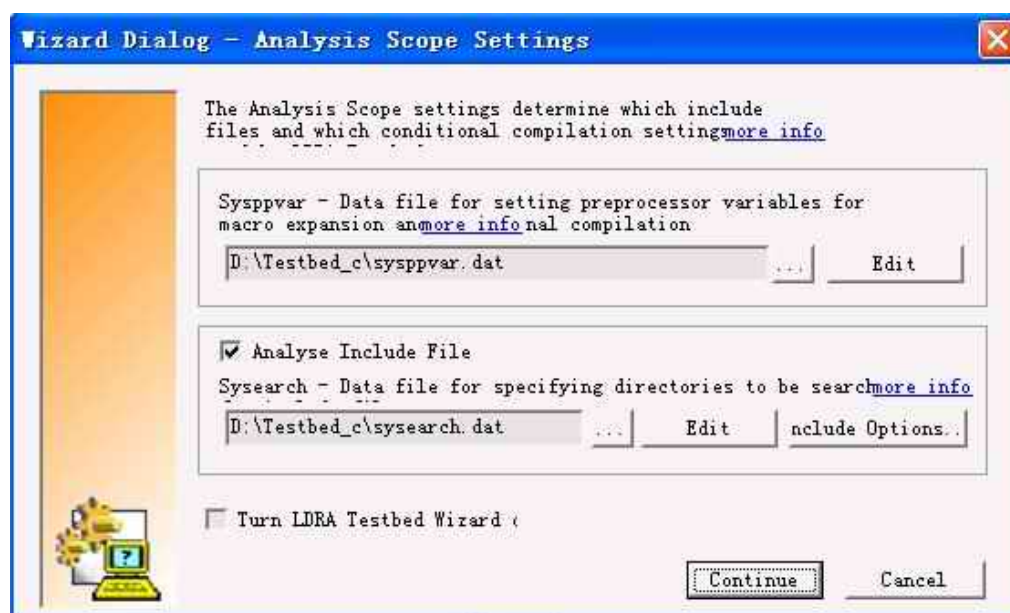
在 Analysis 菜单下选择 Delete Workfiles，删除要删除的相应的结果；



4.4 Testbed 向导对话框

当新的文件或一个 set 被调入 Testbed 时，向导窗口将被激活。

这个对话框允许用户修改和当前文件相关的 `Sysppvar.dat` 文件和 `Sysearch.dat` 文件。

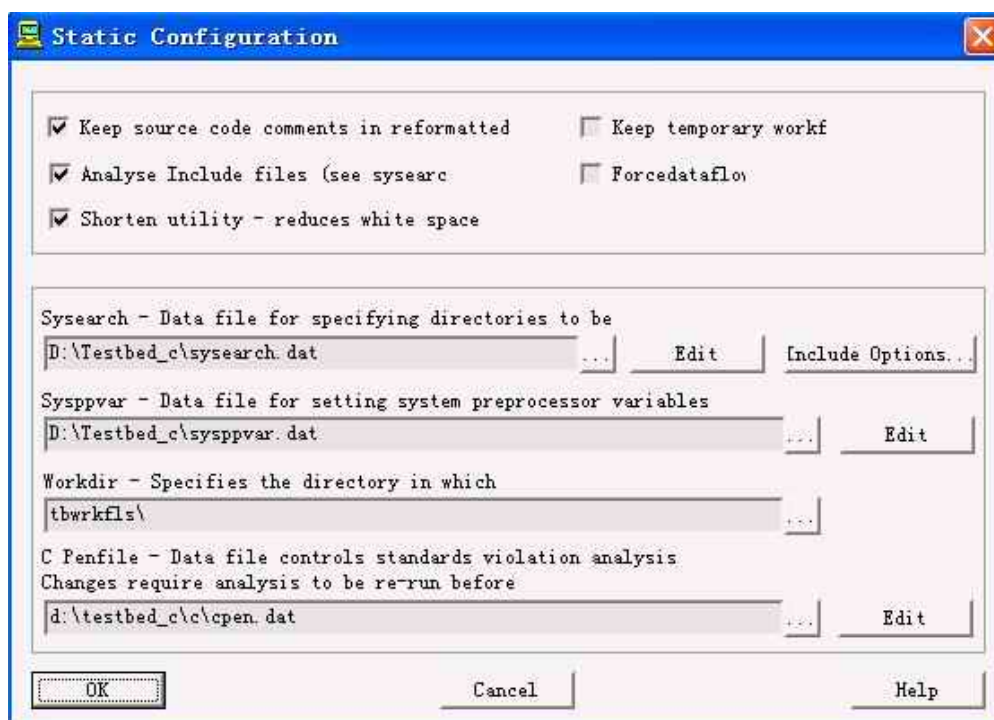


4.5 检查分析范围设置

点击菜单栏中的 **C**onfigure, 在下拉菜单中点击 **S**tatic Options 选项, 将会出现如下窗口:

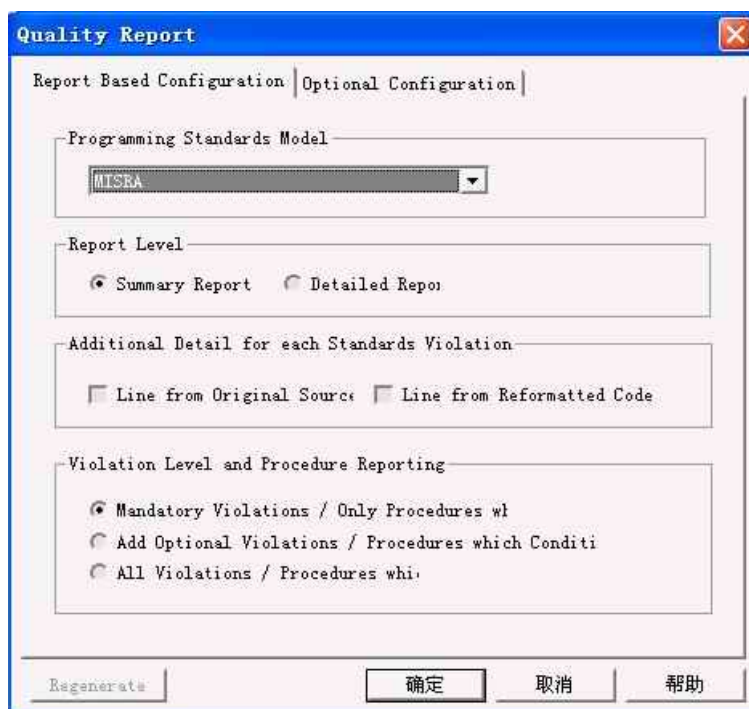
这个窗口用来设置和静态分析相关的选项:

- 是否分析 include 文件;
- 通过 Sysearch.dat 添加搜索路径;
- 通过 Sysppvar.dat 添加宏 (只针对 C/C++);

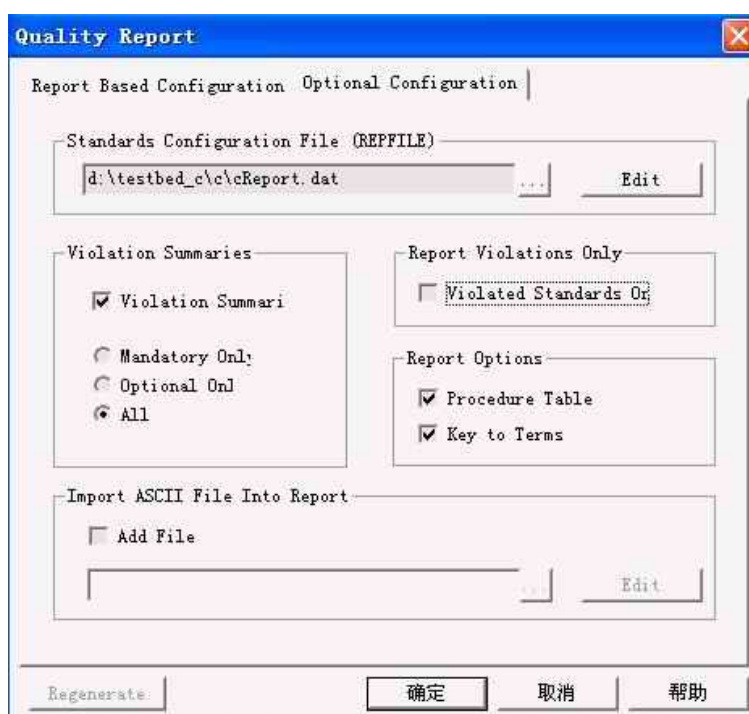


4.6 质量模型设置

点击菜单栏中的 **C**onfigure，在下拉菜单中点击 **Q**uality Report Options 用户能够通过下拉菜单选择使用哪个编码规则。

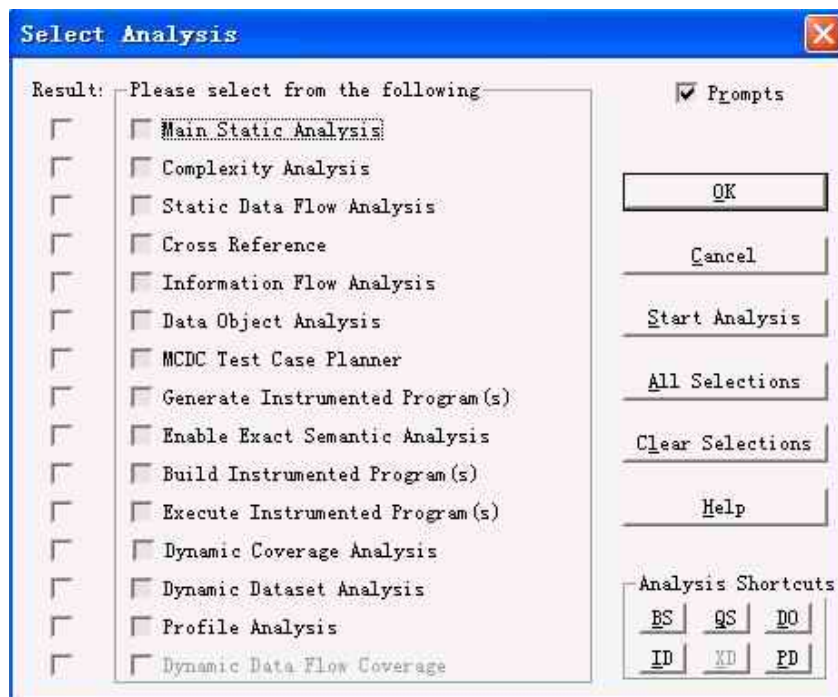


选择 **O**ptional Configuration 页，点击 **E**dit 按钮来编辑 creport.dat 文件；Testbed 通过这个文件来选择进行哪些规则检查，用户可以在该文件中定义自己的质量模型。



4.7 选择分析选项

用户能够通过对话框来选择 Testbed 进行哪些分析。点击 **A**nalysis 下拉菜单，选择 **S**elect Analysis，就会弹出选择分析的对话框。



用户通过选择确认框来选择进行哪些分析，然后选择 **S**tart Analysis 按钮开始进行分析。

4.8 清除当前的选择

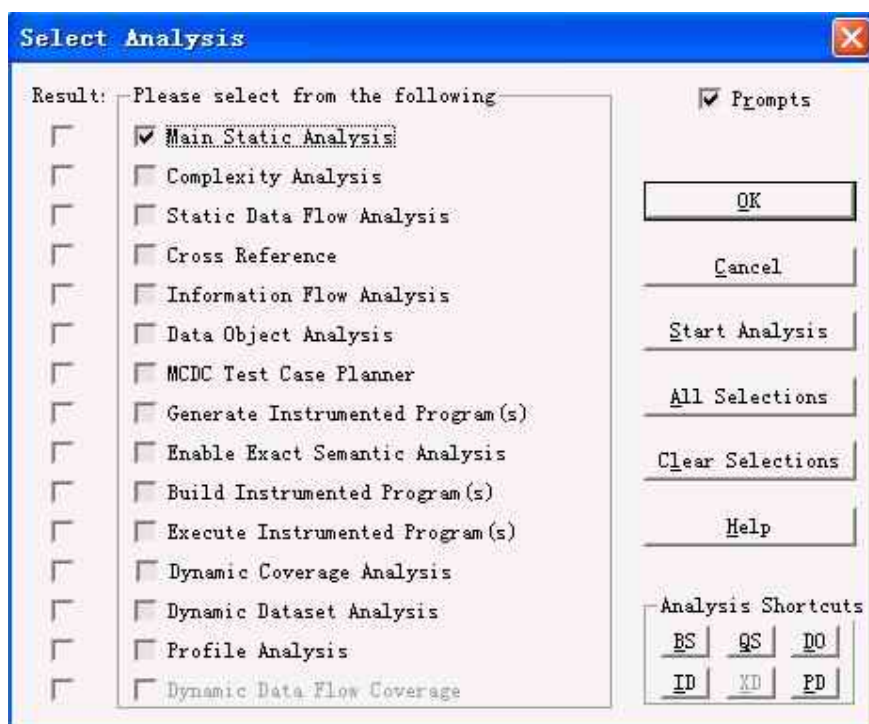
如果已经有选择了的分析选项，请选择“Clear Selections”按钮来清除已经选择的一些选项。

五. 主要静态分析

关于 Main Static Analysis（主要静态分析）的详细内容请参见 Testbed Users Manual 的 539 页。

5.1 运行静态分析并察看结果

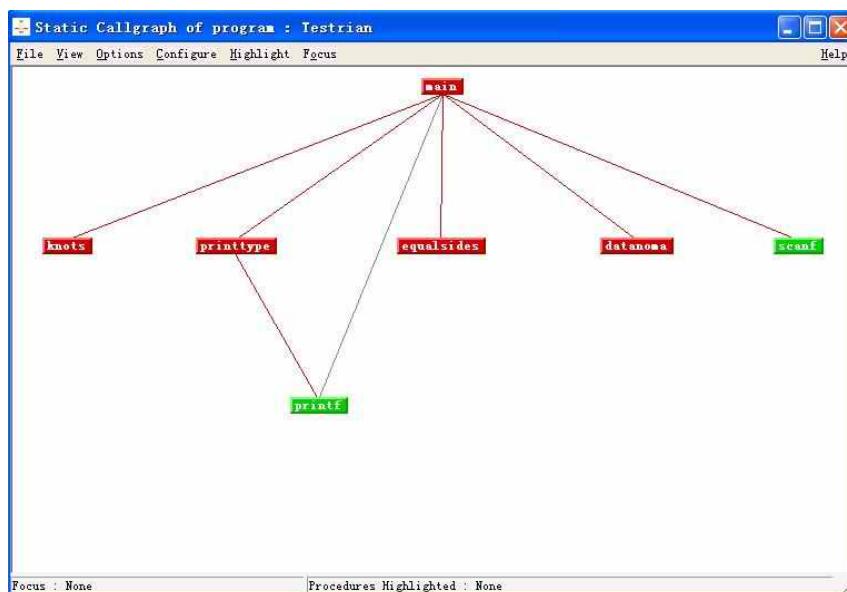
在 Main Static Analysis 前的确认框中打勾，然后单击 **Start Analysis** 来开始分析。如果您选中了 OK 按钮，对话框会消失，那么您要通过在 Analysis 菜单中选中 Perform Analysis 来开始分析。



在 Testbed 的 log 窗口中会显示当前工具正在进行的操作，分析结束的时候会弹出一个消息窗口，点 OK 键确认。

5.1.1 图形化显示分析结果

单击 **Individual Results** 菜单，选择 **Graphical Results**，单击 Static Callgraph，将会弹出系统调用图的窗口。



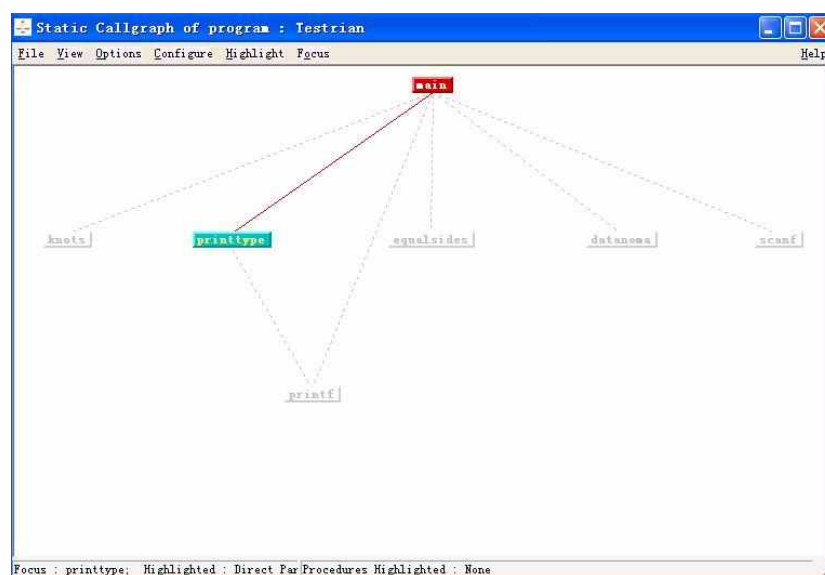
通过 View 菜单中的 Zoom in/out/auto 选项来改变程序的调用图的显示模式。

通过 Option 菜单中的 Exclude/Include System Calls 选项来将当前分析文件的外部函数和系统函数从调用图中去掉。

通过 View 菜单中的 Numbered Nodes 选项来图例化显示调用图中的函数。

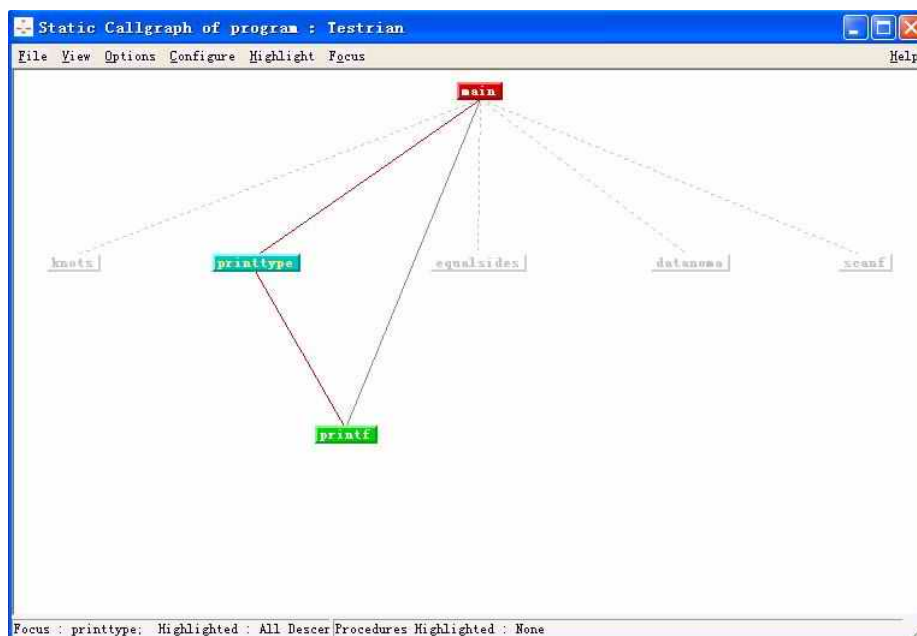
将鼠标箭头放到 `datanoma` 节点上单击右键，弹出的选择菜单允许高亮或者去除当前节点。选择 **Prune** 来去除 `datanoma` 节点，我们将看到 `datanoma` 节点被从调用图中去掉了；在窗口的空白处单击右键选择 **Unprune Graph** 来恢复节点 `datanoma`。

将鼠标箭头放到 `printtype` 节点上，单击右键选择 **Highlight Parents**；这样在调用图中 `printtype` 和调用它的函数将被高亮显示。在 `testrian.c/cpp` 中只有 `mian` 函数调用了 `printtype`。



在上一步我们不选择 **Highlight Parents** 而是选择 **Highlight Children**；这样在调用图中 **printtype** 和它调用的函数将被高亮显示。在 **testrian.c/cpp** 中只有 **printf** 函数被 **printty** 调用了。

在右键菜单中选择 **Highlight Ancestors** 和 **Highlight Descendants** 来高亮节点相关的整个调用关系。



在窗口空白处的任意地方单击右键，选择 **Remove Highlighting**。

在 **printtype** 节点上再次单击右键，选择 **Select as Parent**，这样 **printtype** 会做为调用图的父节点来显示。在窗口空白处的任意地方单击右键，选择 **Restore Original Graph** 来恢复到初始状态。

需要注意在调用图中跨层的线使用灰色显示的。递归调用（**backwards calls**）是用橙色显示的，其他的在 **Configure** 菜单下的 **Node Colour Key** 对话框中有相关的说明。

通过 **File** 菜单下的 **Save As** 可以将调用图保存为 **bitmap** 图。

5.1.2 文本显示分析结果

单击 **Individual Results** 菜单，选择 **Text Results**，选择 **Overview Report (HTML)**。

Overview Report 报告将打开，我们将看到关于指定的编程规则 passed 或 failed 的总体的报告。

Overall Results - Programming Standards

	Quality Result	Unique Standards
		Failure Ratio (%)
main	Fail	2
knots	Fail	5
datanoma	Fail	4
printtype	Fail	5
equalsides	Fail	3

单击 **Individual Results** 菜单，选择 **Text Results**，选择 **Quality Report (HTML)**。

Overall Quality Summary

Totals of Violations for Selected Quality Standards

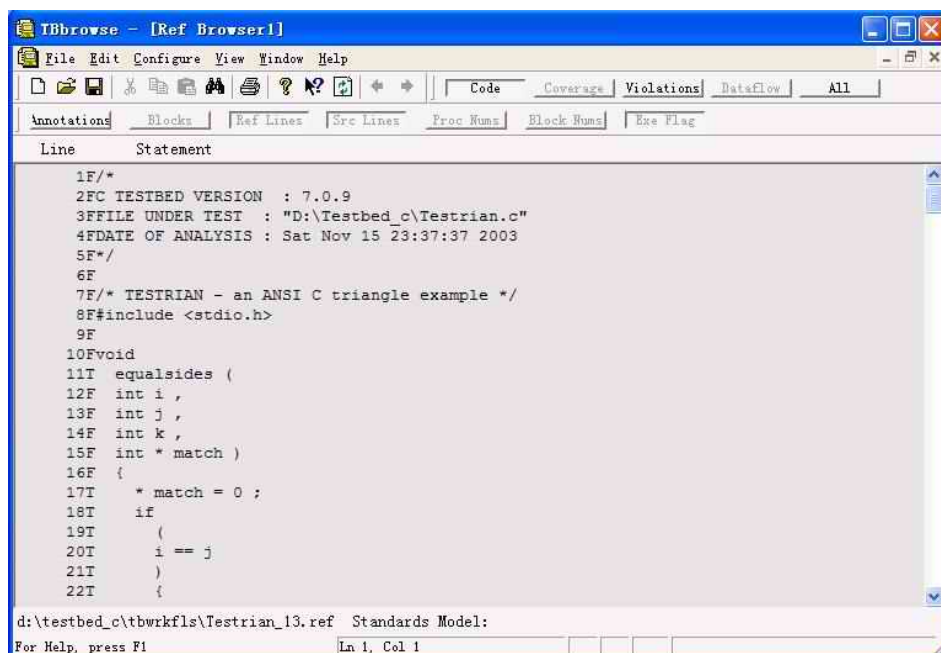
✓ indicates required Analysis Phase results are not yet available.

Number of Violations	(M) Mandatory (Required) Standards	MISRA Code
0	No brackets to loop body (added by Testbed).	MISRA 59
26	No brackets to then/else (added by Testbed).	MISRA 59
25	goto detected	MISRA 56
0	Anonymous field to structure.	MISRA 108, 113
0	Number of parameters does not match.	MISRA 78
0	Use of break statement in loop.	MISRA 58
0	Use of continue statement.	MISRA 57

从前面的列表中选择 **knots** 函数，相应的函数的规则检查的详细的结果将显示，点击 **goto detected** 的超链接将显示具体的规则（违反的部分将以红色显示）。

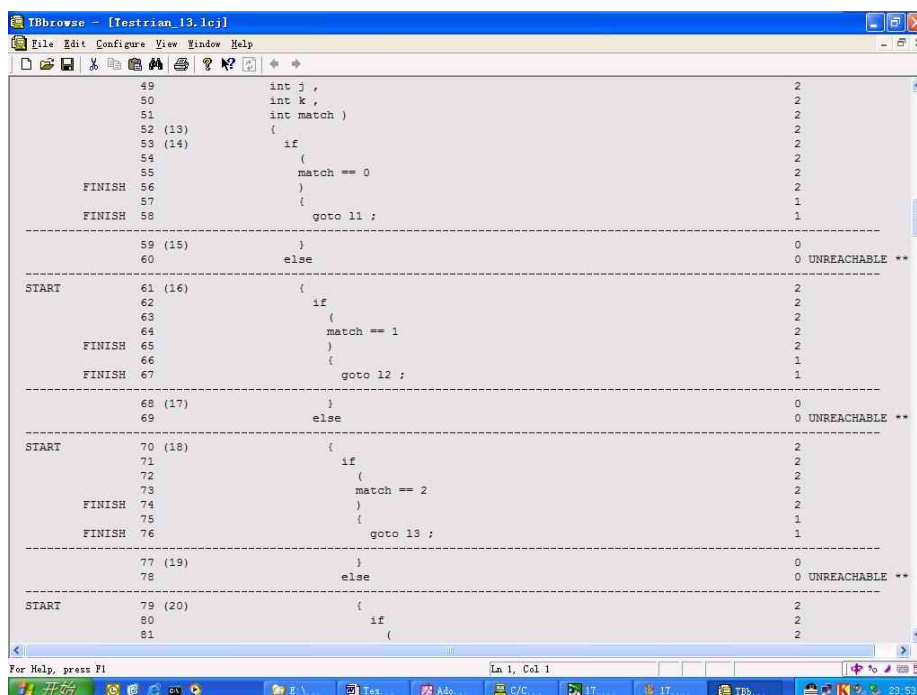
这时规则说明的 **HTML** 文档就被打开了，它给出了相应的规则的说明和示例。

单击 **Individual Results** 菜单，选择 **Text Results**，选择 **Reformatted Code**，将看到格式化代码，通过格式化代码我们可以查看宏展开的情况以及做些一致性检查。



在 Text Results 中选择 LCSAJ Report, 将显示代码和相应的 LCSAJ 定义, 在右边的一列是相应的 LCSAJ 密度, 在文件的底部有一个所有的 LCSAJ 密度的一个列表。

LCSAJs 密度对是软件可维护性度量的一个重要指标, 同时也是进行严格的覆盖率测试的一个基础。



在上面的例子的 LCSAJ 报告中我们可以看到程序中有不可达代码。

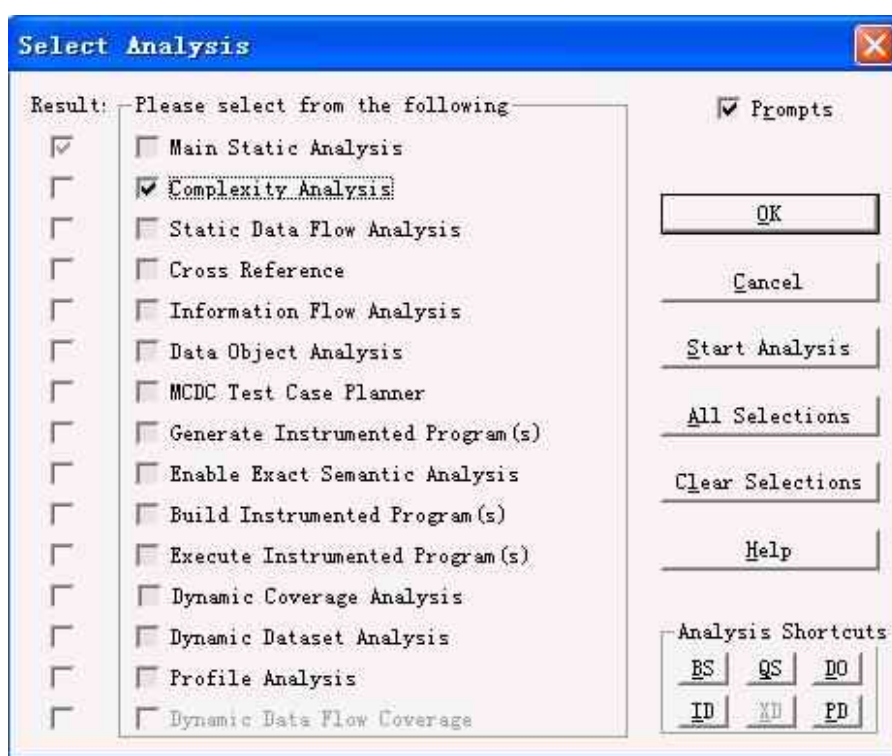
注意: Text Results 菜单中 Metrics Report 要在做了复杂度分析后才产生, 所以现在是不可用的。

六. 复杂度分析

关于 Complexity Analysis（复杂度分析）的详细内容请参见 Testbed Users Manual 的 571 页。

6.1 运行复杂度分析并察看结果

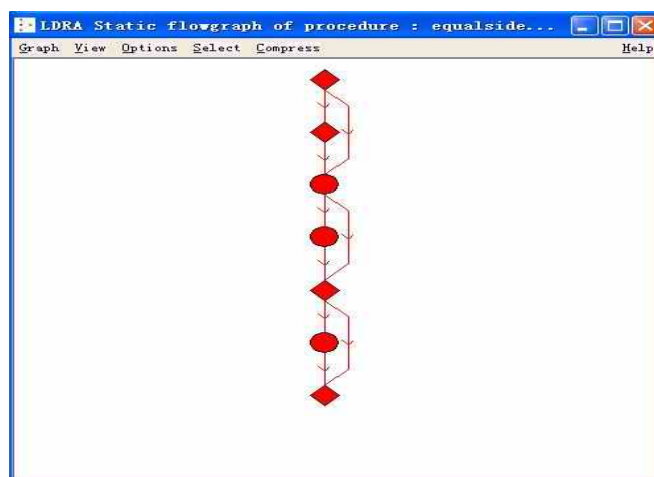
在 Complexity Analysis 前的确认框中打勾，然后点击 **Start Analysis** 来开始分析。如果您选中了 OK 按钮，对话框会消失，那么您要通过在 Analysis 菜单中选中 Perform Analysis 来开始分析。



在 Testbed 的 log 窗口中会显示当前工具正在进行的操作，分析结束的时候会弹出一个消息窗口，点 OK 键确认。

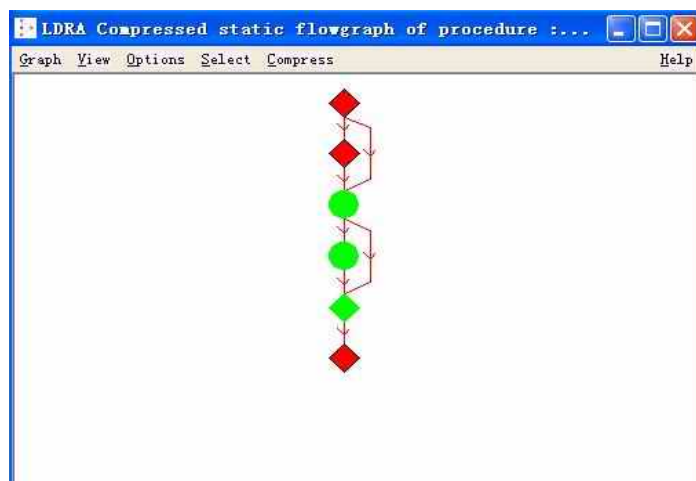
6.1.1 图形化显示分析结果

点击 **Individual Results** 菜单，选择 **Graphical Results**，点击 **Static Flowgraph**，将会弹出程序控制流图的窗口。或者您可以通过左键点击系统调用图中的红色节点来进入相应函数的控制流图。



在控制流图中点击节点可以调出相应的格式化源代码，图中菱形代表该节点所包含的源代码有违反编码规则的情况存在。

点击 **Compress** 选择 **SPV (Structured Programming Verification) Compress**。首先会高亮一块结构化的代码，随后将其化简为一个节点，然后再高亮再化简，最后如果整个程序是结构化的那么将化简为一个节点。

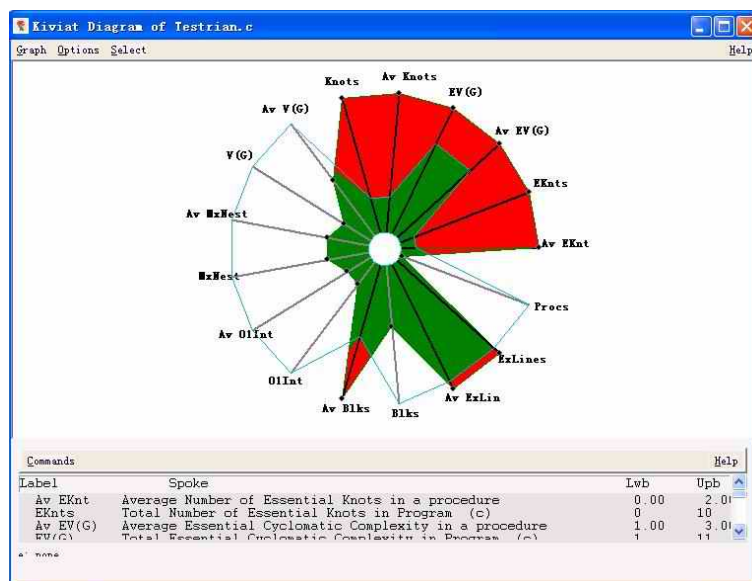


点击 **Individual Results** 菜单，选择 **Graphical Results**，选择 **Standard Kiviat**。**Kiviat** 图以图形的方式显示被分析的代码在软件质量度量方面和预设的质量模型之间的符合情况。

以图例的方式显示哪些度量指标超出了预设的上下限指标。

在 **Graphical Results** 菜单中还有三种特定的 **Kiviat** 图：

- 清晰性 **Kiviat** 图：关于代码清晰性方面的度量结果的 **Kiviat** 图，表明代码的可读性和易理解性；
- 可维护性 **Kiviat** 图：关于代码可维护性方面的度量结果的 **Kiviat** 图，表明代码的可维护性；
- 可测试性 **Kiviat** 图：关于代码可测试性方面的度量结果的 **Kiviat** 图，表明代码的可测试性；

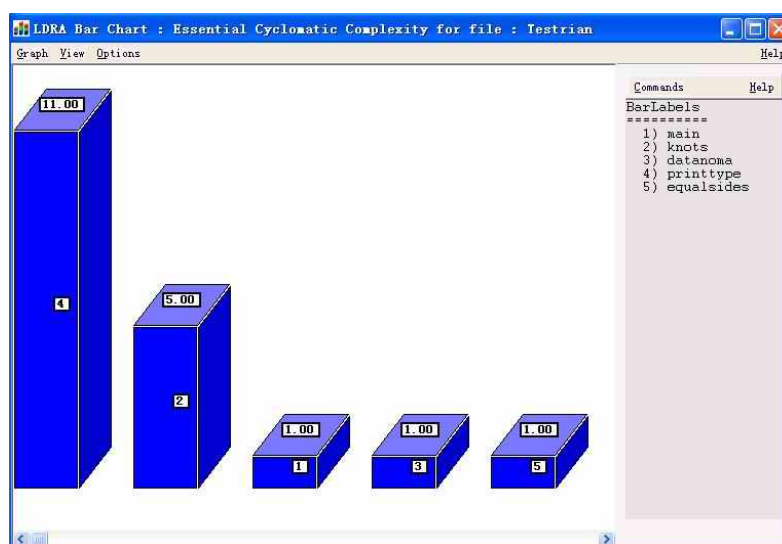


针对 C++ 的代码，还有一些关于 OO 的 Kiviat 图。

在 Graphical Results 菜单中选择 Static Bar Charts，来察看柱状图，可以选择的内容包括：

- 基本节点数
- 基本圈复杂度
- 节点数
- 圈复杂度
- 基本块数
- 可执行格式化代码行数

等



6.1.2 文本显示分析结果

单击 Individual Results 菜单，选择 Text Results，选择 Metrics Report 来察

看复杂度分析的详细结果。



Quality Report 是编码规则检查的报告。

七. 静态数据流，交叉索引，信息流和数据对象分析

关于 Static Data Flow (静态数据流分析) 的详细内容请参见 Testbed Users Manual 的 603 页；

关于 Cross Reference (交叉索引) 的详细内容请参见 Testbed Users Manual 的 619 页；

关于 Information Flow (信息流分析) 的详细内容请参见 Testbed Users Manual 的 623 页；

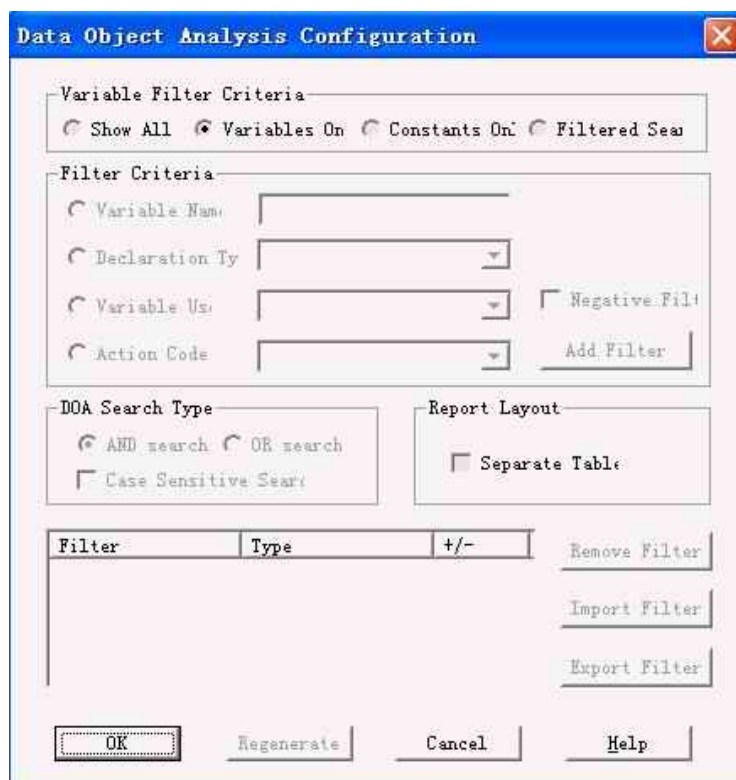
7.1 运行各项分析

在 Analysis 菜单中选择 Select Analysis 选项，选择下列分析选项：

- Static Data Flow Analysis
- Cross Reference
- Information Flow Analysis
- Data Object Analysis

然后选择 OK；

在 configure 菜单中选择 Data Object Analysis Options；将会弹出 Data Object Analysis Configuration 的对话框。



在这里我们可以设置数据对象分析的规则，默认的规则是分析文件中所

有的变量；

通过 Export 按钮我们可以导出当前的分析规则到数据文件中；

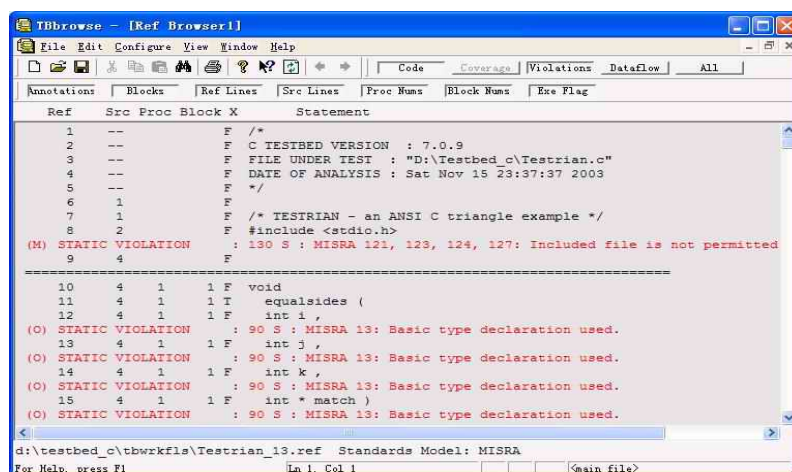
通过 Import 按钮我们可以将已有的分析规则通过数据文件导入。

在设置好 DOA 分析规则后，选择 Perform Analysis 开始分析。

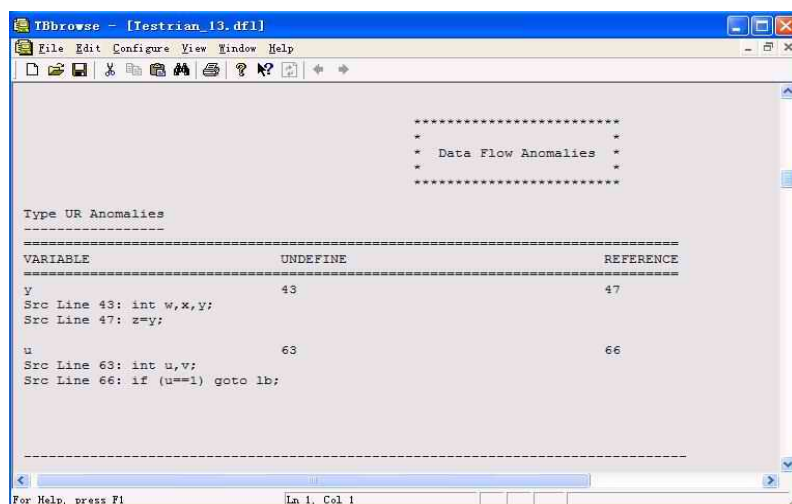
7.2 察看分析结果

7.2.1 察看静态数据流分析结果

分析结束后，在 Text Results 菜单中选择 Reformatted Code，在弹出的窗口中，点击工具栏的 Violations，Dataflow，Annotations 按钮，单击 All 按钮将会显示所以的相关信息。

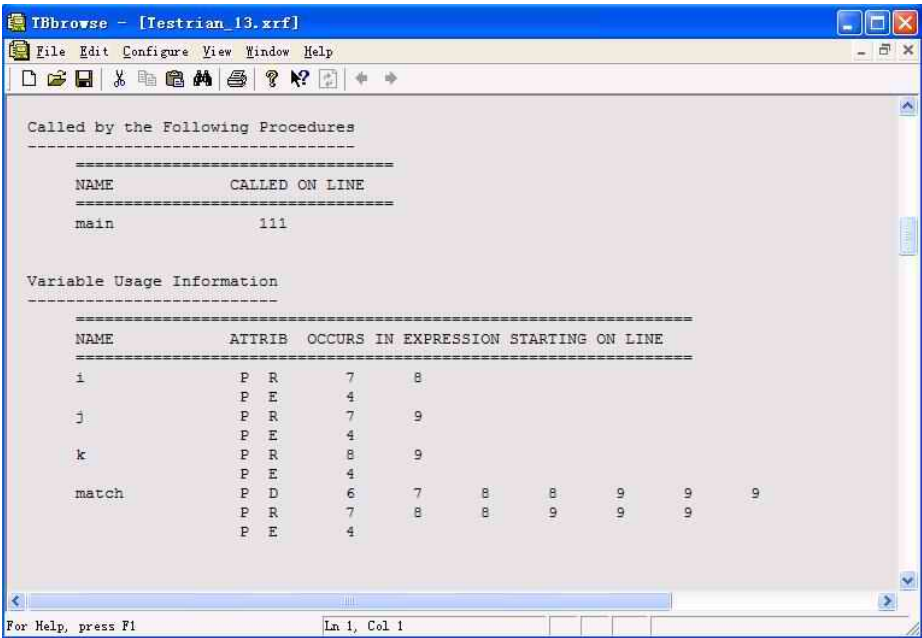


也可以通过点击 Individual Results -> Text Results 菜单下的 Data Flow Analysis Report 来察看静态数据流分析报告。



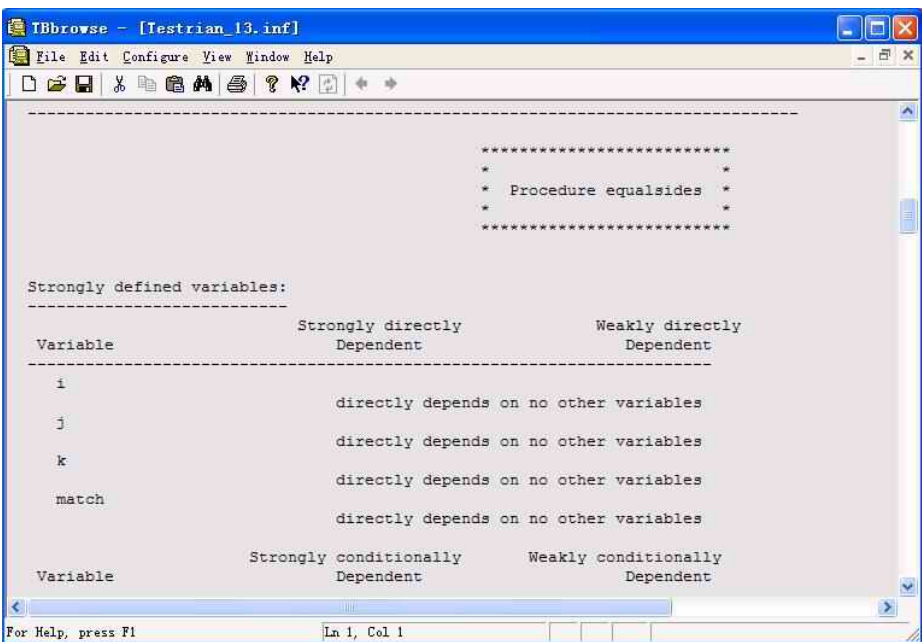
7.2.2 察看交叉索引的结果

通过点击 Individual Results->Text Results 菜单下的 Cross Reference Report 来察看交叉索引的结果。



7.2.3 察看信息流分析结果

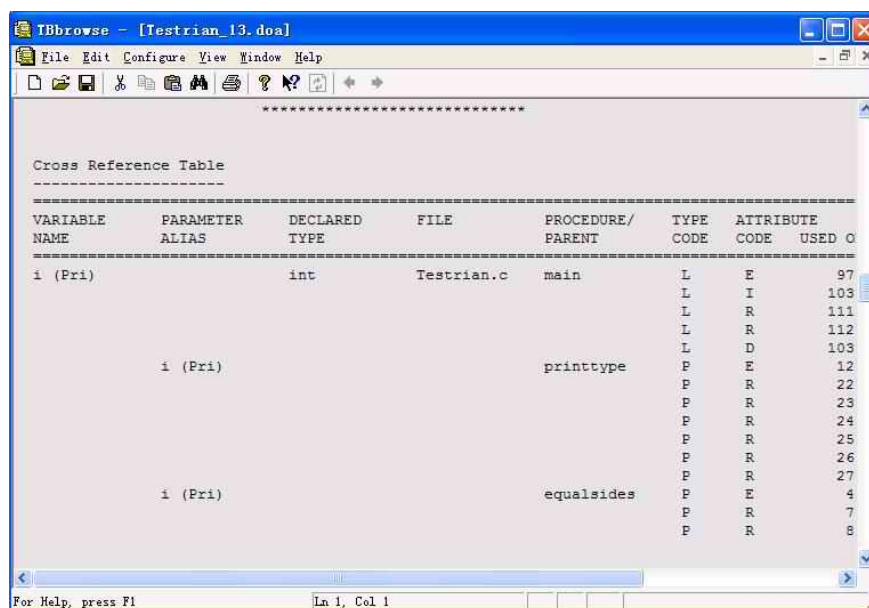
通过点击 Individual Results->Text Results 菜单下的 Information Flow Analysis Report 来察看信息流分析结果。



信息流分析是 TBsafe 可选模块的功能，因此您要有 TBsafe 模块才能进行信息流分析。

7.2.4 察看数据对象分析结果

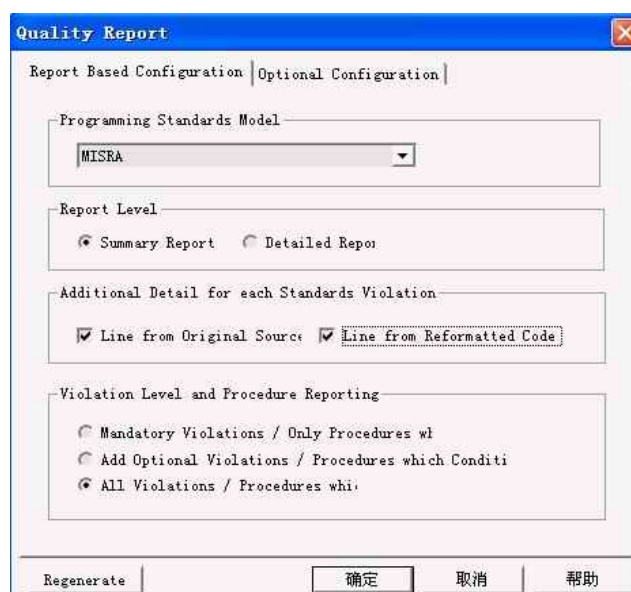
点击 Individual Results → Text Results 菜单下的 Data Object Analysis Report 来察看数据对象分析结果。



VARIABLE NAME	PARAMETER ALIAS	DECLARED TYPE	FILE	PROCEDURE/ PARENT	TYPE CODE	ATTRIBUTE CODE	USED O
i (Pri)		int	Testrian.c	main	L	E	97
					L	I	103
					L	R	111
					L	R	112
					L	D	103
	i (Pri)			printtype	P	E	12
					P	R	22
					P	R	23
					P	R	24
					P	R	25
					P	R	26
					P	R	27
	i (Pri)			equalsides	P	E	4
					P	R	7
					P	R	8

7.2.5 察看质量报告

首先通过点击 Configure 菜单下的 Quality Report Options 来设置质量报告的格式和内容。



Quality Report

Report Based Configuration | Optional Configuration

Programming Standards Model

MISRA

Report Level

☒ Summary Report ☐ Detailed Report

Additional Detail for each Standards Violation

☒ Line from Original Source ☒ Line from Reformatted Code

Violation Level and Procedure Reporting

☐ Mandatory Violations / Only Procedures which Violate

☐ Add Optional Violations / Procedures which Conditionally Violate

☒ All Violations / Procedures which Violate

Regenerate 确定 取消 帮助

报告的具体内容：报告中既给出总体的概述也给出所以函数的详细的结果。

函数的详细结果：

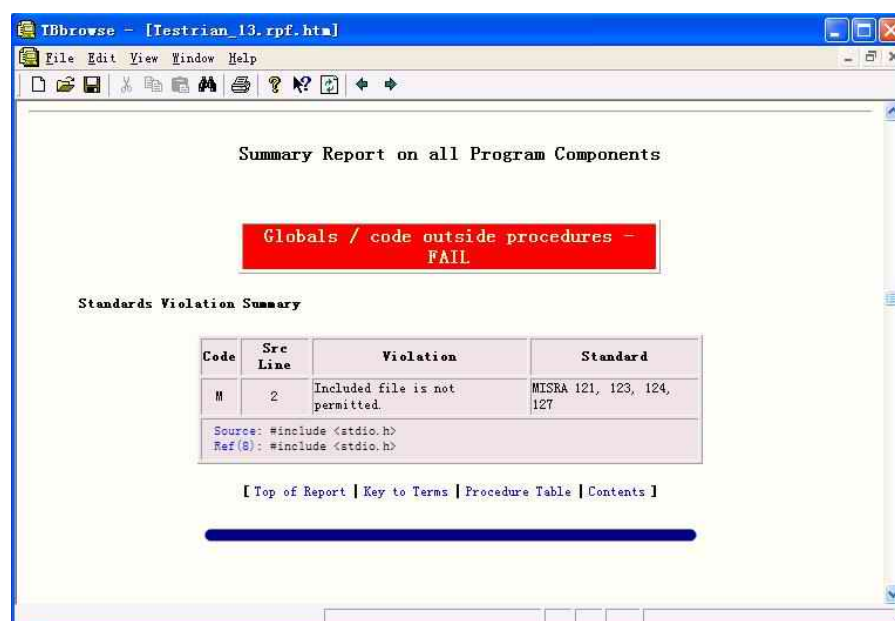
- **Fails Only**: 只显示不符合质量模型的函数；
- **Conditional Passes**: 显示不符合质量模型和部分符合质量模型的函数；
- **All Passes**: 显示所有函数，包括符合质量模型的函数；

规则违反的详细结果：报告中既给出违反的规则，也给出相关的源代码信息。

从对话框中选择 Line from Original Source File 和 Line from Reformatted Code File。

在 Violations Level and Procedure Reporting 区选择 All Violations / Procedures which Pass。

打开 Quality Report 报告我们可以看到所有的数据流信息，同时我们将看到不光有源代码信息还有格式化代码信息。

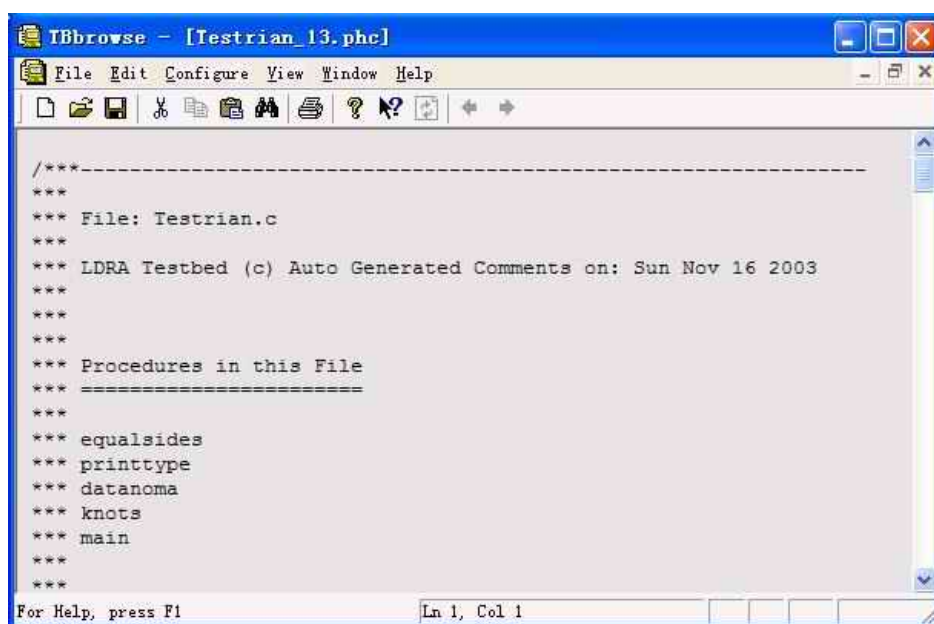


7.2.6 察看其他分析结果

点击 Individual Results ->Text Results 菜单下的 User Defined Types Report 察看用户定义变量报告，报告中详细描述了文件中用户自定义的变量的情况。

点击 Individual Results ->Text Results 菜单下的 Procedure Header Comments Report 来察看函数头注释报告，这是由 Testbed 根据函数自动分

析出来的，可以将其拷贝到源码头部作为函数的头部说明。



八. 动态分析

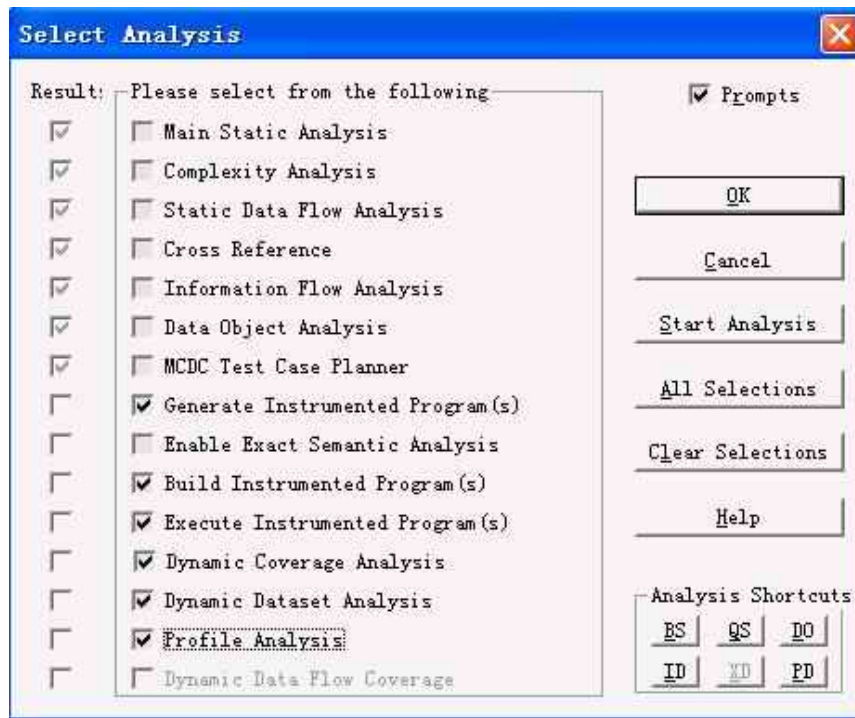
关于 Dynamic Analysis（动态分析）的详细内容请参见 Testbed Users Manual 的 695 页；

8.1 进行动态分析

本节将分步讲解代码插装和进行测试的过程。Testbed 将用来分析测试的覆盖率和测试数据的有效性。

从 **A**nalysis 菜单中选择 **S**elect Analysis 将会弹出 Select Analysis 对话框，选择下面几项进行分析：

- Generate Instrumented Program(s)
- Build Instrumented Program(s)
- Execute Instrumented Program(s)
- Dynamic Coverage Analysis
- Dynamic Data Set Analysis
- Profile Analysis



点击 Start Analysis 开始分析，这时会弹出 Build Configuration 对话框。

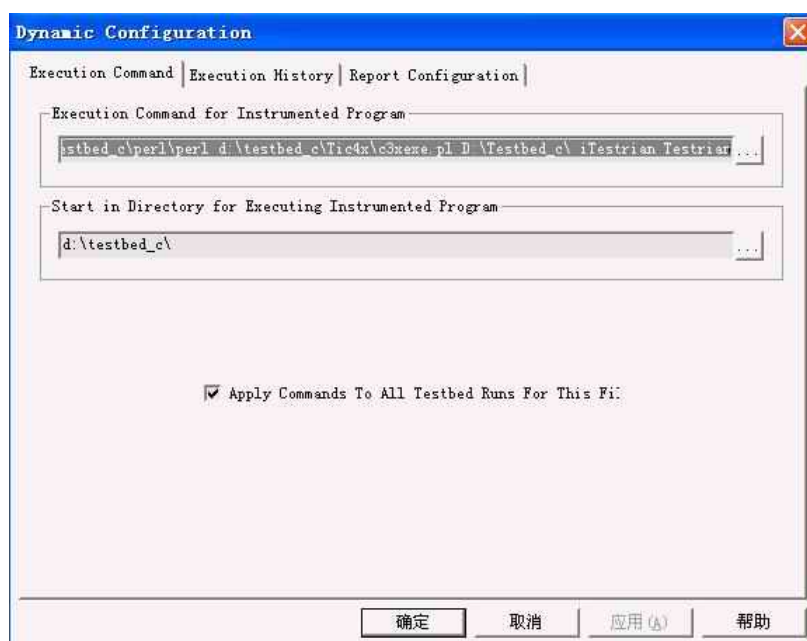
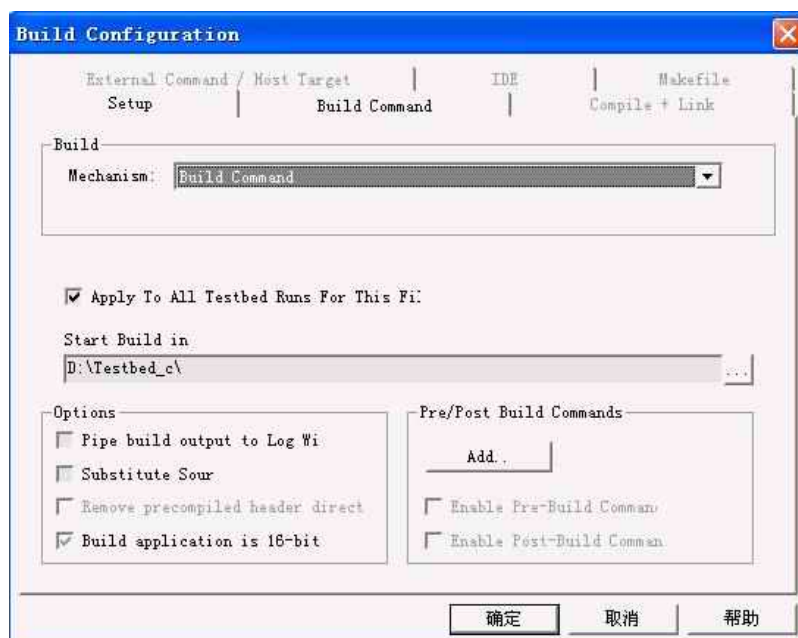
我们将配置 Testbed 在主机上编译程序并在主机上运行。Testbed 也可以配置为将程序在主机上的 Simulator 上运行或者在嵌入式的目标机上运行。

具体的编译命令如下所示：

点击 Ok 确认当前配置。

8.2 选择执行插装程序命令

动态配置窗口将会出现：



上面的编译命令是 Testbed 当前的默认配置；默认配置是从配置文件中读取的：

testbedctl （所有的 UNIX/VMS 所有的非 C/C++ 的 windows 版本）；
default_testbed.dat （没有编译器设置的 C/C++ 的 windows 版本）；
<compiler>_testbed.dat （有编译器设置的 C/C++ 的 windows 版本）；

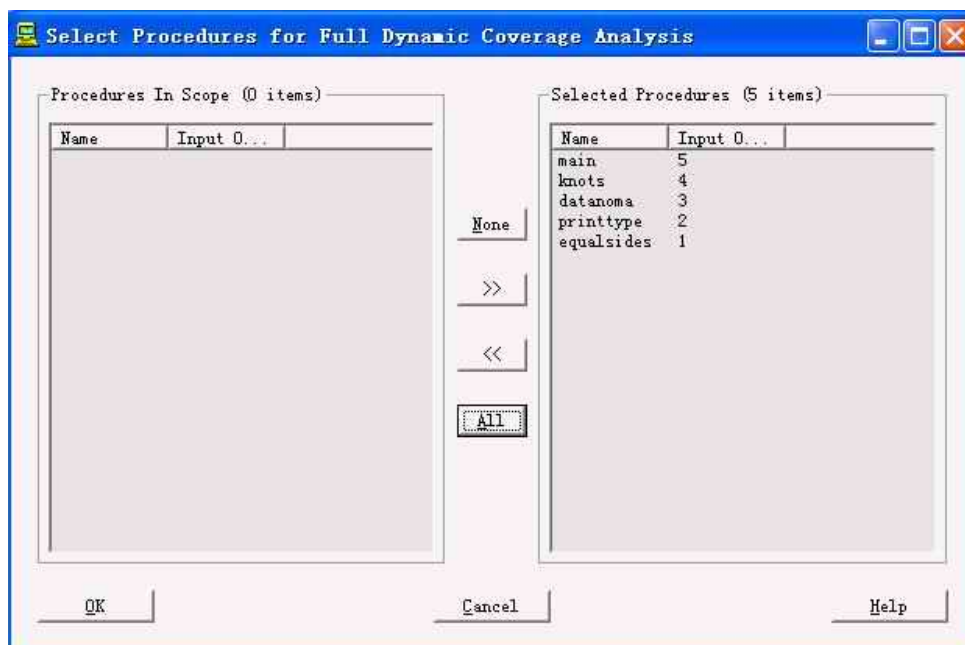
点击 OK 确认当前配置。

8.3 选择动态覆盖率分析选项

动态覆盖率选项对话框将弹出：



选择默认选项，当前的分析将被作为 Run1,下面的窗口询问您要分析文件中的哪些函数，选择 All 分析所有的文件。



点击 OK 确认当前的配置。

8.4 执行分析

现在 Testbed 就开始执行前面选择的分析选项，在执行的过程当中 Log 窗口中将显示当前的分析执行情况。

8.5 执行插装程序

在 Testbed 对源代码插装，编译后，就执行编译好的插装代码，当前用的这个例子是一个根据三角形三边判断三角形是什么三角形的程序；程序运行后会弹出一个 dos 窗口，您需要在窗口中输入数据以便执行当前的测试，具体的内容如下：

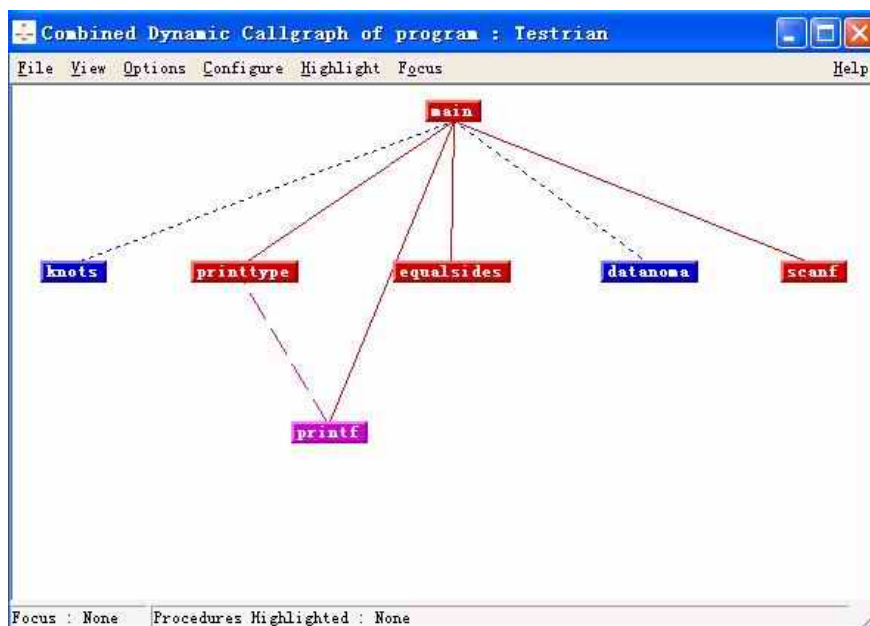
```
input number of
3
input 3 integers
3 4 5
scalene
input 3 integers
3 3 2
isosceles
input 3 integers
4 4 4
equilateral
```

其中数字是需要您输入的内容，其他是程序的输出信息。在执行完插装程序后，Testbed 将自动进行覆盖率分析。

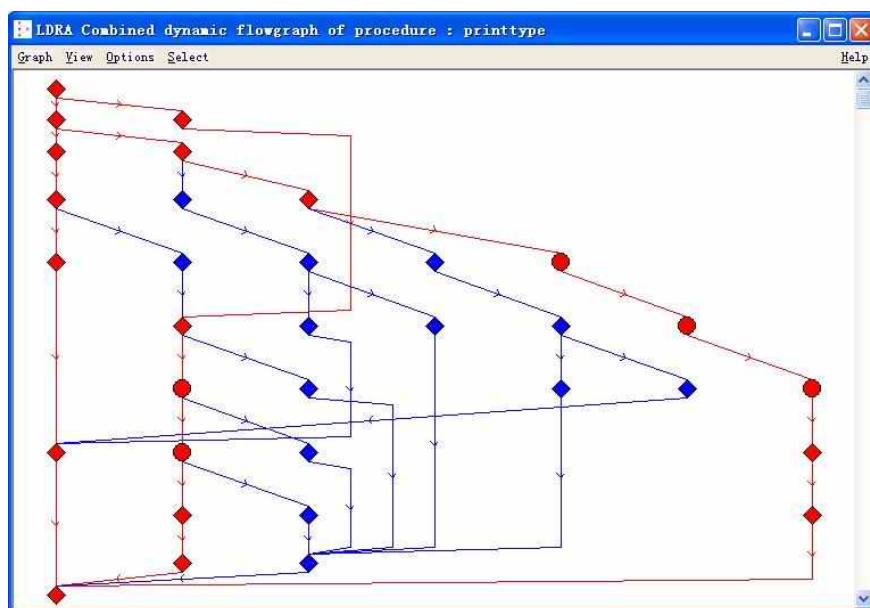
一旦分析完成点击 OK 按钮完成。

点击 **Individual Results** 菜单，选择 **Graphical Results**，点击 **Combined Dynamic Callgraph** 来察看动态执行结果的调用图，图中以不同的颜色来表明运行情况：

- 一直没有调用过的函数用蓝色表示；
- 所有可能的调用都被执行了函数用红色表示；
- 函数至少被调用了一次但不是所有可能的调用都被执行了的用粉红色表示；
- 一直没有被执行的调用线用蓝色的虚线表示；
- 执行了的调用线用红色的实线表示；
- 函数有多种被调用情况，其中的一些被执行了，其它没有被执行的用粉红色的虚线表示；



点击 printtype 节点将弹出 Combined Dynamic Flowgraph:



图中通过颜色来标示显示了哪些节点被执行了，哪些没有被执行：

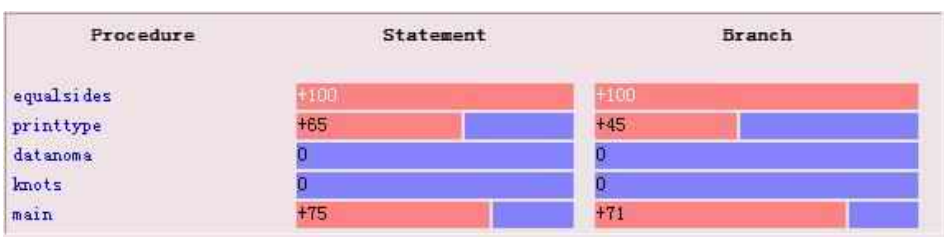
- 没有被执行的节点用蓝色表示；
- 没有被执行的分支用蓝色表示；
- 执行了的节点用红色表示；
- 执行了的分支用红色表示；

左键单击节点可以看到相应节点的源代码，察看源代码有助于您设计测试用例。关掉前面打开的窗口。

从 **Graphical Results** 子菜单选择 **Dynamic Bar Charts**，可以用柱状图的方式来察看覆盖率结果，包括以下内容：

- Total Testedness
- MC/DC
- BCCC
- BCC
- LCSAJ Coverage
- Branch Coverage
- Statement Coverage
- Coverage Metrics

点击 Individual Results 菜单，选择 Text Results，选择 Dynamic Coverage Analysis Report，将看到如下的报告，前面是覆盖率结果的总体描述：



下面是详细的结果：

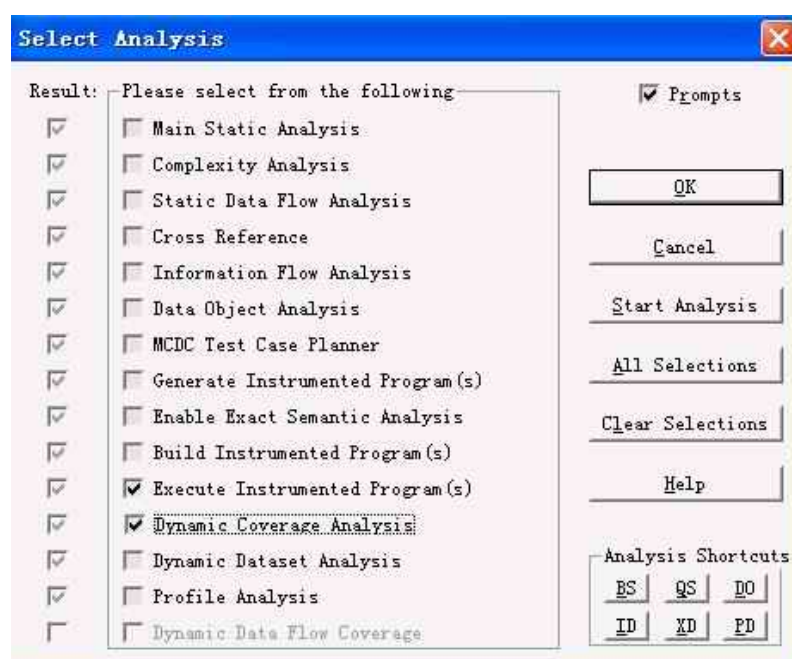
LINE NUMBER	STATEMENT	PREVIOUS RUNS	CURRENT RUN	COMBINED
REF. (SOURCE)				
10 (4)	void	-	-	-
11	equalsides (0	3	3
12	int i ,	-	-	-
13	int j ,	-	-	-
14	int k ,	-	-	-
15	int * match)	-	-	-
16 (5)	{	-	-	-
17 (6)	* match = 0 ;	0	3	3
18 (7)	if	0	3	3
19	(0	3	3
20	i == j	0	3	3
21)	0	3	3
22	{	0	2	2
23	++ * match ;	0	2	2
24 (8)	}	0	2	2
25	if	0	3	3
26	(0	3	3
27	i == k	0	3	3
28)	0	3	3
29	{	0	1	1
30	++ * match ;	0	1	1
31	++ * match ;	0	1	1
32	}	0	1	1
33	;	0	3	3
34 (9)	if	0	3	3

九. 深层次的动态分析

前面我们只是运行了一次程序，输入了一组数据，得到了相应的结果；如果要增加覆盖率，我们可以再次运行程序。

9.1 再次执行插装后的程序

从 **Analysis** 菜单中选择 **Select Analysis** 将会弹出 **Select Analysis** 对话框，选择 **Execute Instrumented Program(s)** 和 **Dynamic Coverage Analysis**，再次执行。



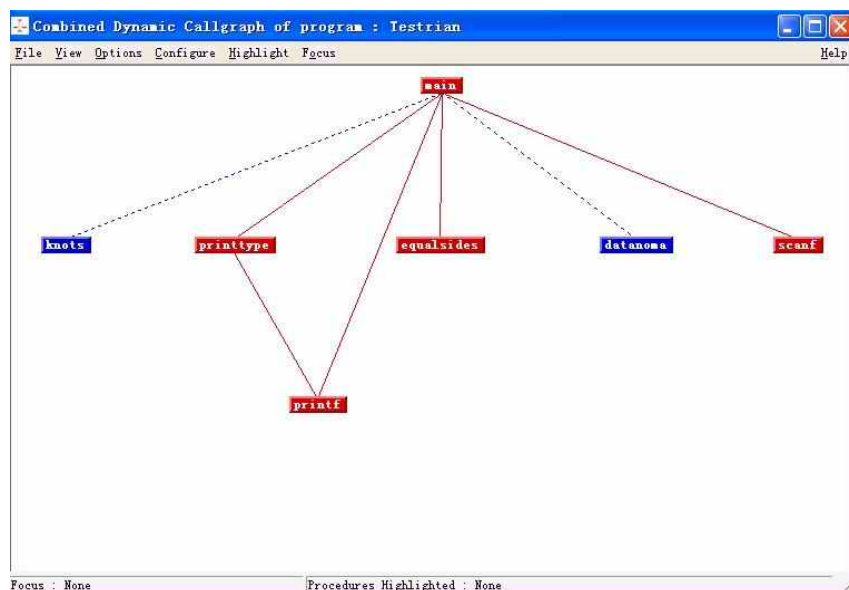
这次动态覆盖率选项框对当前分析的默认值为：**Run2**

程序运行后，会弹出 **dos** 窗口，在窗口中输入数据，具体内容如下：

```
input number of triangle
3
input 3 integers
3 3 3
EQUILATERAL
input 3 integers
32 1 1
NOT A TRIANGLE
input 3 integers
5 5 6
ISOSCELES
```

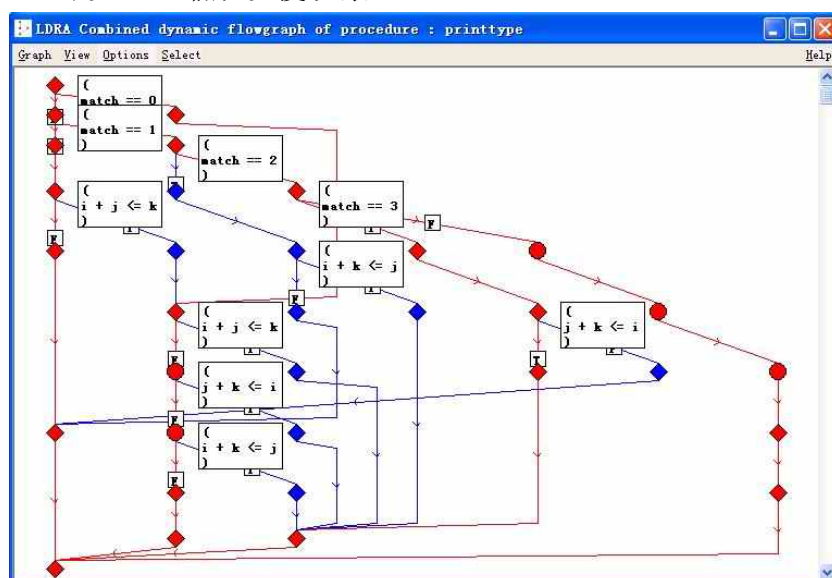
程序执行后，**dos** 窗口会关闭，会进行动态覆盖率分析。

点击 **I**ndividual Results 菜单，选择 **G**raphical Results，点击 Current Dynamic Callgraph，察看当前运行的覆盖率结果。



点击 **I**ndividual Results 菜单，选择 **G**raphical Results，点击 Combined Dynamic Callgraph，可以看到系统覆盖率有所增加。

点击 **I**ndividual Results 菜单，选择 **G**raphical Results，点击 Current Dynamic Flowgraph，可以通过缩放以便观察。



点击 **I**ndividual Results 菜单，选择 **G**raphical Results，点击 Dynamic Bar Charts 选项选择 Coverage Metrics。

点击 **I**ndividual Results 菜单，选择 **T**ext Results，点击 Dynamic Coverage Analysis。

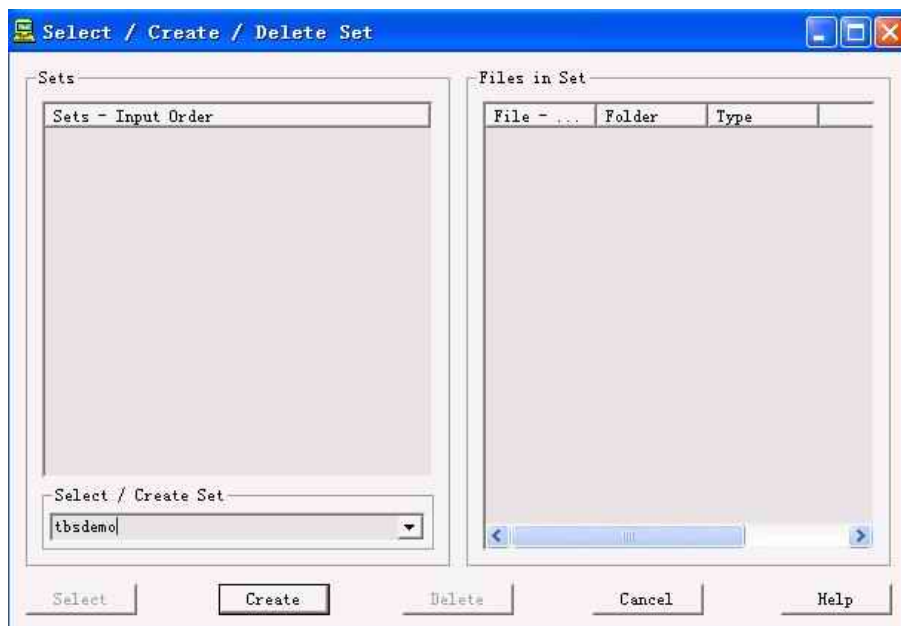
在 **Options** 中选择 **Add Annotations**，我看到在程序的分支点都加上了判断条件的注释，这对于设计测试用例是有帮助的。

十. 以集 (set) 的方式进行分析

Testbed 可以用集 (set) 的方式同时对多个文件进行分析。

从 Set 菜单, 选择 Select/Create/Delete Set 选项。

然后被提示, 往如下的对话框中输入集的相应的名称



集的名字不能为空白, 也不应以空格开头, 还不可包含如下的字符:
\\:*?>|。用不被允许的字符给集命名会产生一个强制集命名的错误消息。
如果往 LDRA Testbed 中输入有效的命名, 这个名称会被确认并登陆到数据库。

输入 tbsdemo, 然后单击 Create 按钮。

10.1 设置集属性

给新集命名后会弹出如下的对话框, 并要求给一个属性。



选择“System”，然后 LDRA Testbed 就会把新集的名字以及其属性保存到它的内部的数据库。

10.2 往集里添加文件

从 Set 菜单下选择 List/Add/Remove Files in Set，将弹出如下窗口：



点击 Add 按钮添加要分析的程序，添加完成后如下：

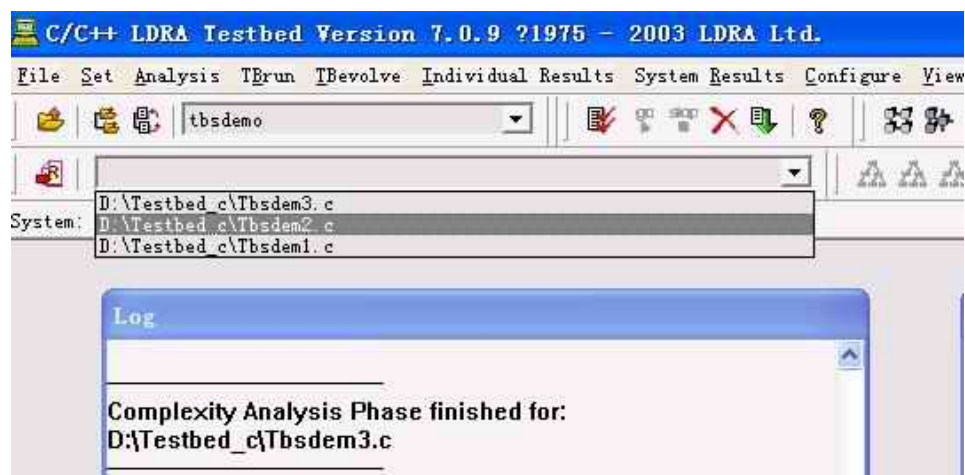


点击 OK 按钮完成。

10.3 集的分析及结果察看

对集的文件的分析 and 前面的单个文件的是一样的。

不同的是集的方式在结果显示上，**System Results** 菜单下是一个集内所以文件的分析结果的汇总；如果要单独察看单个文件的结果，需要在如下的选择框选择想察看的文件：



选定了要察看的文件后，相应的结果察看的方式也和前面单个文件的方式一样。

十一. 附注：数据流分析

例如： dflow1.c

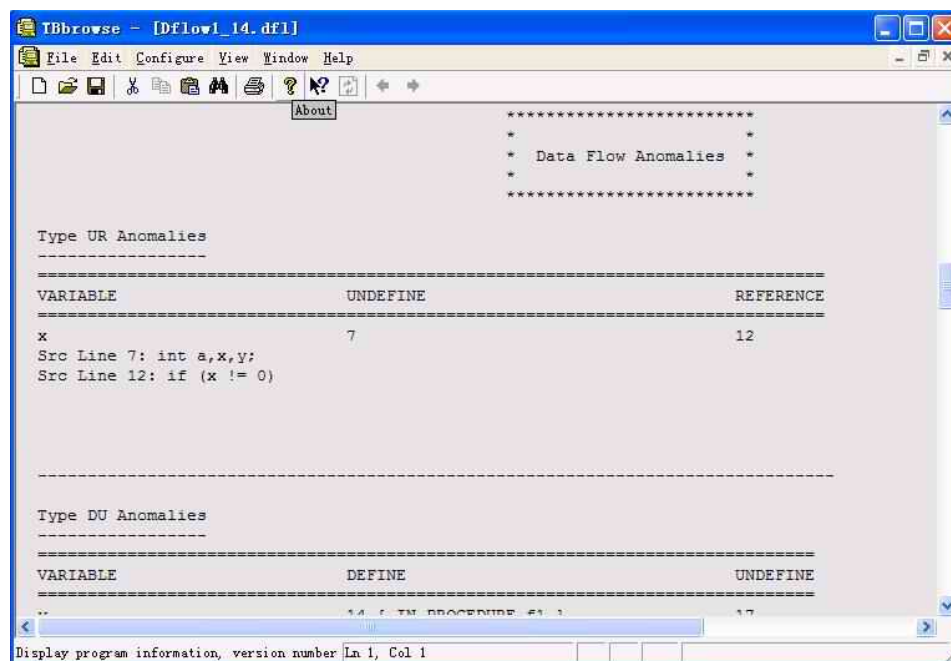
代码如下：

```
void f1(int *p1, int p2, int p3)
{
    *p1 = p3 - p2;
}

main()
{
    int a,x,y;
    printf ("Type in a value for a");
    scanf ("%d",&a);
    printf ("\n");
    y = 0;
    if (x != 0)
    {
        f1 (&y,a,x);
    }
    printf ("A is %d\n",a);
}
```

1) 用 C/CPP LDRA Testbed 分析 dflow1.c, 分析该文件的静态、复杂度和静态数据流。

察看静态数据流报告：



变量 x: UR (未初始化就引用)

变量 y: DU (初始化后未被引用就出作用域了)

变量 y: DD (初始化后未被引用就再次被初始化)

2) x 的 UR 异常是由于在 “if “ 的条件之前未给 x 赋初值引起的。

修正:

在原码中给 x 赋初值或从键盘输入一个值给 x。

3) Y 的 DU 异常是由于 Y 被赋初值 0 后, 或是在 f1 函数中再次赋值后, 都没有被引用就出了其作用域。

修正:

在程序末尾, 在打印语句中加入 y。

4) y 的 DD 异常是在 Y 被赋初始值 0 后如果是走 if 的真分支那么 Y 在没有被引用的情况下就再次被赋了初值。这是由于使用了不完整的 if 语句造成的。

修正:

用 if-then-else 结构重写 if 语句。

5) 重新分析修改后的 dflow1.c, 所有的数据流异常都没有了。

可行的一种修改的办法如下:

```
void f1(int *p1, int p2, int p3)
{
    *p1 = p3 - p2;
}
main()
{
    int a,x,y;
    printf("Type in a value for a");
    scanf ("%d",&a);
    printf ("\n");
    x = 1;
    if (x != 0)
    {f1 (&y,a,x);}
    else
    {y = 0;}
    printf ("A is %d\n",a);
    printf ("Y is %d\n",y);
}
```

十二. 附注：信息流分析

例如：iflow1.c，这个文件包含了一些基本的信息流关系。
代码如下：

```
main(){
    int i,j,k;
}
f14( int i,int *j,int *k,int *z, int *m, int *n)
{
/*LDRA_INFOFLOW j (i#sd i#sc ) */
/*LDRA_INFOFLOW m (k#wd m#wc i#sc ) */
    if( i ==1)
    {
        *j = i;
        *z = 1;
        *m = i;
        *n = *m;
        *m = *z + *k;
    }
    else
    {
        *j = i;
        if( *m ==1)
        {
            *k = i;
            *z = *k;
            *m = 9;
            *n = *m + *z;
            *m = 0;
        }
    }
}
```

用 C/CPPLDRA Testbed 分析 iflow1.c，分析该文件的静态、复杂度、数据流信息流分析。

各种信息流依赖关系在结果中被列出。

实际上这些关系用户自己可以预知的，并可以在原码中给出注释。信息流分析可以验证这些。

下面的注释可在代码中给出：

- j 强直接依赖 i
- j 强条件依赖 i
- m 弱直接依赖 k
- m 强条件依赖 i

- m 弱条件依赖 m

依据信息流结果表再加一些注释来指出 k 、 n 和 z 的依赖关系。

十三. 分析自己的代码

13.1 概述

分析用户自己的代码是根据代码的需求结构，开发环境，存储位置来分析的。

分析代码可以根据用户希望的分析范围来进行。如果只是仅仅对代码做静态分析，无需对 **LDRA TESTBED** 做任何设置，可直接分析。

13.2 基本规则

分析代码的基本原则有：

- 从简单的开始；
- 分析一个简单的程序的容易成功，分析一个系统不容易成功。如果先从一部分入手，一个文件或者几个文件，那么就有更多经验去成功分析整个系统；

13.3 分析范围

分析范围对于成功分析原代码是很重要的。**Testbed** 与编译器是分离的，编译器必须用开发方开发代码时的编译器。**Testbed** 的配置中的编译命令和选项必须和开发是的环境一致。

在分析范围方面必须考虑如下几个问题：

1. 源文件：

分析单个文件时，先启动 **testbed**，然后从菜单 **File** 的子菜单 **select file** 的弹出对话框中选择被测程序。这在前面的文件分析的例子中已经有描述。

分析多个文件时，多个文件可以被放在一个组（**group**）的集或者一个系统（**system**）的集内，然后以集为单位所有的文件一起分析。

如果编译器是 **VC++**，用户可以用 **LDRA Testbed** 分析一个工程文件（**projectname.dsp**），可以分析工程中的单个的文件，也可以分析整个工程文件。

2. 头文件

LDRA Testbed 分析头文件是通过把头文件扩展到相应的原文件中的方法来分析的。头文件的分析是由菜单 **Configure** 的子菜单的 **Static Option** 中的选项开关来控制的

缺省时, LDRA Testbed 扩展被引用的、和原码在同一目录下的头文件。如果头文件在其他的路径下(和原码不同目录),可以在 `sysearch.dat` 文件中指明引用头文件的路径。

例如:

```
#include <My_include.h>
```

如果这个头文件需要被分析,这个的文件路径是 `C:\project\includes\`,那么在文件 `sysearch.dat` 中需要加入

```
I C:\project\includes\
```

这是指 `My_include.h` 和其他在代码中被引用并在此目录下的头文件可以在分析原码时被展开。

`sysearch.dat` 文件,是从 `Configure→static option` 对话框中访问。

3. 宏展开的设置

LDRA Testbed 采用它自己的方式进行原代码宏扩展,和编译器的宏扩展方法一样。因此 LDRA Testbed 需要和编译器一样的设置,这样才能保证采取的分析是相同方式,就象两种不同的环境中用的代码是编辑和同版本的。

编译器的宏扩展信息用多种方式给出给出:

- a) 通过未被LDRA Testbed分析的头文件;
- b) 通过编译器命令行;
- c) 通过一个Makefile或工程文件;

LDRA Testbed 的宏扩展信息是通过创建数据文件 `sysppvar.dat` 文件给出,这个数据文件可以通过 `Configure-→static option` 对话框来创建和访问。

例如:

```
#ifdef _CONSOLE  
scanf("%d",&value);  
#endif
```

如果包含这部分代码的文件被编译,在编译命令行上就有 `_CONSOLE` 的定义。这样就必须在 `sysppvar.dat` 文件中有个对应 `_CONSOLE` 入口

```
_CONSOLE 1
```

如果 LDRA Testbed 没有成功通过宏扩展的设置,那么所有的分析会失败,插装后的代码也不能被编译通过。

分析范围的设置可以通过按 `F6`,然后产生附带的报告(SIR)来具体察看,也可以从菜单 `Individual Results-→Text Results-→Analysis Scope Report` 或者从菜单 `Set Results-→Text Results-→Analysis Scope Report` 来察看。

13.4 编译插装后的代码

13.4.1 概述

做完静态分析，用户在分析列表中可选择自动插装，会产生一个插装文件，通过编译执行插装后的代码，然后 Testbed 分析，可以得到原码的白盒覆盖率报告。

在编译插装后的文件时，需要明白并不是被测系统的所有原文件都要被插装。例如：一个有 300 个文件的系统，用户可以插装其中的一个文件，也可以把 300 个文件都插装。用户在第一个测试覆盖率的实例中，应该首先测试一小部分文件而不是整个系统。

在这个阶段，插装后的代码的编译和执行不必在 Testbed 中进行。可以用插装后的代码代替原代码，然后在开发的环境中编译，在实际的环境中运行。同时，

LDRA Testbed 可以很便利调用一个编译器的命令行，或编译命令文件。

被插装后的文件在执行时，会打开一个历史文件并把分支执行的信息写入该文件，在执行结束关闭该文件。在 LDRA Testbed 的动态分析的选项中，指定执行的历史文件（sourcefilename.exh or history.exh）的路径。

13.4.2 初步

当编译含有插装的文件系统时，在指望自动操作之前，比较简单的做法是手动的用插装后的文件代替系统中的原文件，这样可以获得经验和信心。

静态分析并插装一个或部分原文件，然后把插装的原文件备份一下并把插装后的文件拷贝到原文件所在的位置，修改插装后的文件的名称，即去掉插装后文件的名称的前 inszt_，接着按以前的方式编译你的系统。

执行完测试用例，可以在系统中发现执行的历史文件。只要在 LDRA Testbed 中指定历史文件所在的路径，就可以分析了。

从 Configure->Dynamic Option 弹出的 Dynamic Configure 对话框中的 Execution History 标签下的指定历史文件的位置。然后做动态覆盖率的分析。

13.4.3 自动过程

一旦基本过程已经证实了，用户可以注意自动过程了。

自动操作可以完全脱离 LDRA Testbed 进行。Testbed 不关心插装后的文件的编译和执行，只关心历史文件。

如果适宜的话，工具提供自动编译和执行：

例如 VC++ 的编译环境。

13.4.4 进一步

如果测试允许，用户可以一次插装更多的文件。这种方法对于看覆盖率是怎样获得的是很有效的。

然而，一次只对部分插装，然后针对插装的部分操作，这样可能更有明智。这种方法鼓励更多的可重负和少些特别的测试过程。