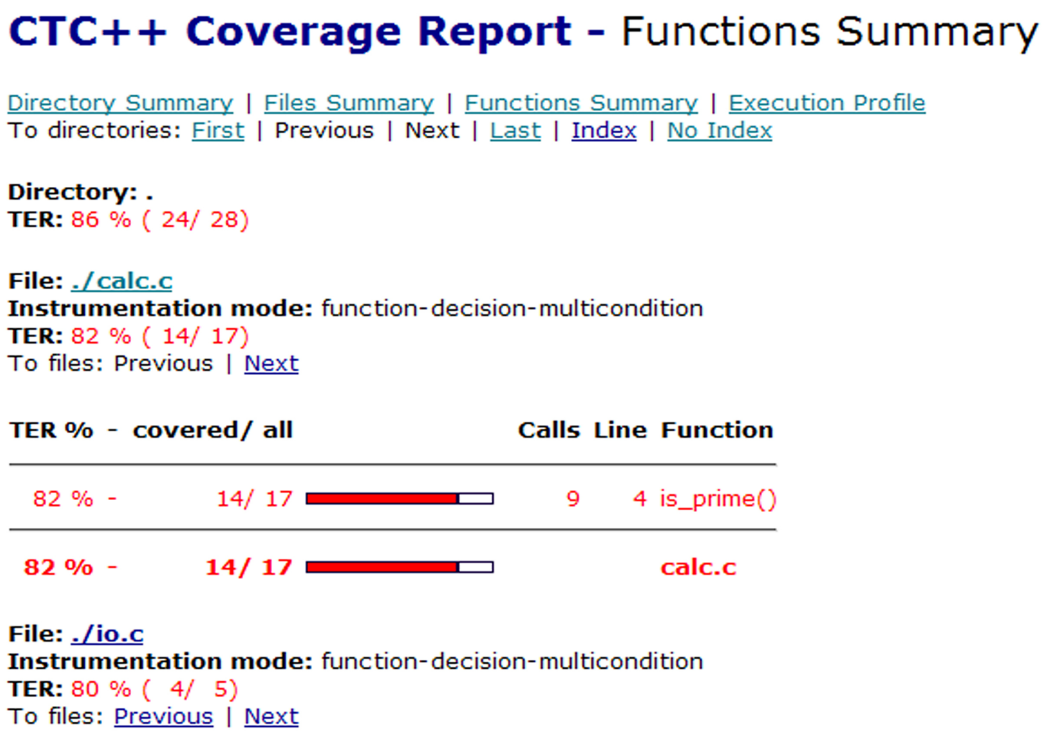
Testwell CTC++: analyse de couverture de code pour systèmes embarqués critiques

par le professeur Dr. Daniel Fischer (université de sciences appliquées d'Offenbourg, Allemagne)
traduction de l'anglais par Olivier Casse

On utilise souvent du logiciel pour systèmes embarqués dans les systèmes critiques. Dans ce domaine des malfonctions peuvent provoquer des accidents ou des dommages de grande ampleur, voire même la perte de vie humaine. En conséquence, les normes de sécurité telles que la DO178-C (aviation), l'ISO 26262 (automobile) ou EN 50128 (ferroviaire) exigent une preuve rigoureuse de couverture de code. Selon la criticité, un niveau approprié de couverture de code doit être appliqué.
La couverture des fonctions est calculée en comptant le nombre de toutes les fonctions appelées et en le divisant par le nombre total des fonctions existantes dans le code embarqué.
L'avantage de ces tests est assez faible parce que le flux de contrôle à l'intérieur des fonctions est complètement ignoré.
La couverture des déclarations recense les déclarations qui ont été exécutées par des essais. Déjà à ce niveau, le code mort peut être repéré, les déclarations n'ayant pas encore de cas de test peuvent être trouvées aussi.
La couverture de branches est calculée sur la base de toutes les branches primitives sans les signaler explicitement.
La méthode MC / DC (Modified Condition Decision Coverage) est calculée en tenant compte de toutes les conditions atomiques d'une condition composite. Pour chaque état atomique, il doit être prouvé par une paire de cas de tests, que la décision finale est affectée par cette condition atomique tandis que les autres conditions atomiques restent inchangées. Ce niveau de couverture est obligatoire pour du logiciel à sécurité critique dans les industries aéronautique et automobile.



CTC++ Coverage Report - Functions Summary

Directory Summary | Files Summary | Functions Summary | Execution Profile
To directories: First | Previous | Next | Last | Index | No Index

Directory: .
TER: 86 % (24/ 28)

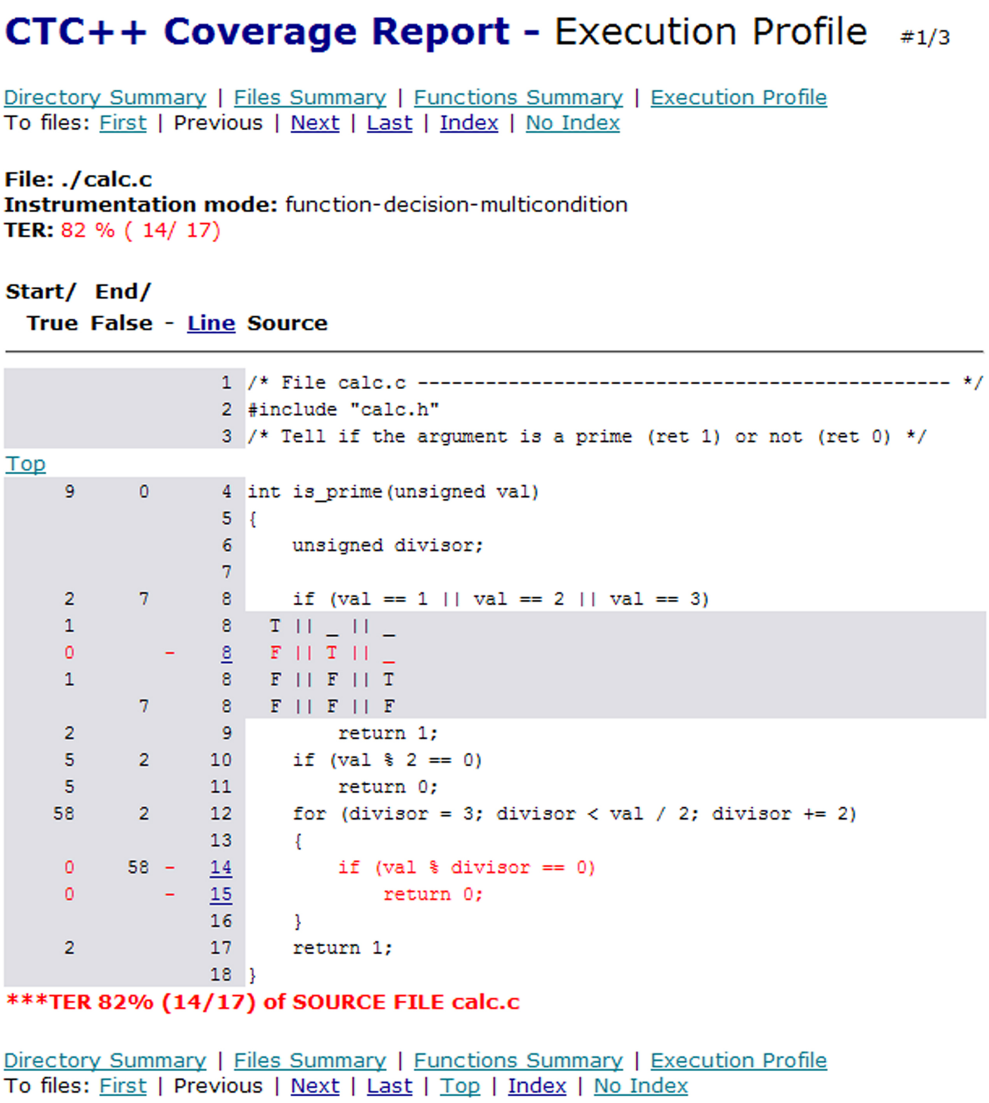
File: ./calc.c
Instrumentation mode: function-decision-multicondition
TER: 82 % (14/ 17)
To files: Previous | Next

TER % - covered/ all	Calls	Line	Function
82 % - 14/ 17	9	4	is_prime()
82 % - 14/ 17			**calc.c**

File: ./io.c
Instrumentation mode: function-decision-multicondition
TER: 80 % (4/ 5)
To files: Previous | Next

Testwell CTC++ montre la couverture de code pour toutes les fonctions

Pour l'analyse de couverture de code, le code source du logiciel embarqué est instrumenté et de préférence exécuté à la fois sur l'ordinateur hôte et sur la plate-forme cible. Pour certains systèmes embarqués plusieurs limitations doivent être considérées, comme moins de mémoire RAM et ROM disponibles. Cette réalité ne permet pas une instrumentation du code complète.
Pour effectuer une analyse de couverture, évidemment il doit y avoir une plus grande utilisation de mémoire qu'avec le code source non instrumenté. L'instrumentation entraîne une plus grande consommation de RAM et de ROM. Cela peut s'avérer un véritable défi particulièrement pour les systèmes embarqués compacts. Une instrumentation partielle avec des cycles d'essais répétés pourrait être une solution. Pour réduire l'utilisation de RAM, il y a aussi la possibilité de réduire la taille d'un compteur de 32 bits à 16 bits voire même jusqu'à 8 bits. S'il est seulement important que le code soit couvert et pas combien de fois, alors il est possible de remplacer les compteurs 32 bits par des bits individuels. C'est ce que l'on appelle la couverture de bits et peut réduire le besoin en taille de RAM supplémentaire par un facteur de près de 32. Cette technologie est soutenue par l'analyseur de couverture Testwell CTC++ de Verifysoft Technology.



CTC++ Coverage Report - Execution Profile #1/3

Directory Summary | Files Summary | Functions Summary | Execution Profile
To files: First | Previous | Next | Last | Index | No Index

File: ./calc.c
Instrumentation mode: function-decision-multicondition
TER: 82 % (14/ 17)

Start/ True	End/ False		Line	Source
			1	/* File calc.c ---------- */
			2	#include "calc.h"
			3	/* Tell if the argument is a prime (ret 1) or not (ret 0) */
Top				
9	0		4	int is_prime(unsigned val)
			5	{
			6	unsigned divisor;
			7	
2	7		8	if (val == 1 || val == 2 || val == 3)
1			8	T || _ || _
0		-	8	F || T || _
1			8	F || F || T
	7		8	F || F || F
2			9	return 1;
5	2		10	if (val % 2 == 0)
5			11	return 0;
58	2		12	for (divisor = 3; divisor < val / 2; divisor += 2)
			13	{
0	58	-	14	if (val % divisor == 0)
0		-	15	return 0;
			16	}
2			17	return 1;
			18	}

***TER 82% (14/17) of SOURCE FILE calc.c**

Directory Summary | Files Summary | Functions Summary | Execution Profile
To files: First | Previous | Next | Last | Top | Index | No Index

La partie rouge de code n'est pas encore couverte. Un test où la première condition est fausse et la seconde est vraie doit être ajouté afin de parvenir à une couverture complète du code.

Pour plus d'informations: http://www.verifysoft.com/fr_ctcpp.html