Десето вежбање

Вежба 1

Испитивања функционалности софтвера могуће је извршити на неколико различитих нивоа. Базични ниво се односи на јединично испитивање (тзв. *Unit testing*) помоћу којег се верификују функционалности засебних целина програма - најчешће на нивоу функције или одређеног модула.

У овој вежби је представљено аутоматско јединично тестирање модула кружног бафера (имплементираног у деветом вежбању). Обично се за тестирање програма користе посебна радна окружења ради једноставније организације процеса тестирања. У овом случају се користи једно од најмањих таквих окружења за Це језик – Јунити (*Unity* - http://www.throwtheswitch.org/unity).

Испратити следеће кораке:

1. Направити статичку библиотеку *CircularBuffer* и ископирати изворне датотке из фолдера *exercises/CircularBuffer* у направљени пројекат.
2. Превести пројекат.
3. Направити нови извршни (*Executable*) пројекат *CircularBufferTest* и ископирати датотеке из фолдера *exercises/CircularBufferTest* у креирани пројекат.
4. Повезати пројекат са статичком библиотеком *CircularBuffer* (статичке библиотеке рађене су у Дану 6).
5. Превести и потом покренути пројекат.
6. Анализирати излаз програма.
7. Анализирати и разумети тестне случајеве дефинисане у датотеци *CircularBufferTest.c*. Направити измене у тестним случајевима тако да бар 2 произвољна теста падну. Покренути опет програм и потврдити да модификовани тестови заиста не пролазе.
8. Прегледати све изворне датотеке пројекта *CircularBufferTest* и разумети њихову намену (зашто је потребан *FormatOutputSpy* модул?).

Напредни Це

Вежба 2

У претходно креирани пројекат *CircularBufferTest* потребно је додати следеће испитне случајеве:

1. *CircularBufferTest_WrapAround* - овим тестом је потребно испитати гранични случај када се „обиђе цео круг“ тј. када смо дошли до краја физичке меморије бафера и даљи упис је потребно наставити на почетку физичке меморије бафера. Довољно је уписати један нови елемент и верификовати да су све уписане вредности у баферу очекиване.

2. *CircularBufferTest_PutToFullDoesNotDamageContents* – испитати ситуацију уписа у већ комплетно напуњен бафер и верификовати да претходно уписане вредности нису промењене.

3. *CircularBufferTest_PrintNotYetWrappedAndIsFull* – верификовати испис на терминал у ситуацији када је бафер у потпуности пун, али ниједна вредност претходно није очитана тј. нисмо ни једном до сада „обишли цео круг“.

4. *CircularBufferTest_PrintOldToNewWhenWrappedAndFull* – верификовати испис на терминал у ситуацији када је „направљен цео круг“ тј. упис се поново врши на почетку физичке меморије (ситуација као у тесту *CircularBufferTest_WrapAround*).

Вежба 3

Системско тестирање је још једно од кључних нивоа испитивања рада софтвера. Под овим се подразумева тестирање комплетно интегрисаног софтвера. У овој вежби је илустровано системско тестирање exo апликације, која на задати улаз одговара идентичним излазом. С обзиром да се главна функција ове апликације састоји од само једне функције, системско тестирање се може аутоматизовати тако да се извршава из самог кода апликације. Као и у претходној вежби, Јунити се користи као тест окружење за аутоматизацију испитивања.

Испратити следеће кораке:

1. Направити извршни пројекат *Echo* и ископирати сав садржај фолдера *exercises/Echo* (укључујући и комплетан фолдер *EchoTest*) у направљени пројекат.

2. Укључити статичку библиотеку *CircularBuffer* у пројекат (као у 4. кораку Вежбе 1).
3. Превести и потом покренути пројекат.
4. Ручним уносом на стандардни улаз тј. терминал испробати бар 4 различита смислена случаја и потврдити да програм заиста функционише очекивано.
5. У датотеци *Echo.h* откоментарисати макро *ECHO_TEST*.
6. Поново превести и покренути пројекат.
7. Анализирати излаз програма, а потом додати бар још 2 различита смислена тестна случаја у *EchoTest.c*.
8. Превести, покренути пројекат и потврдити да програм функционише очекивано.

Вежба 4

У општем случају, у системском тестирању се на апликацију гледа као да је црна кутија (тзв. *Black-Box Testing*), без икаквог познавања кода и унутрашње структуре програма. Приликом оваквог начина тестирања, потребно је да имамо само извршну датотеку (у овом случају *Echo.exe*).

У овој вежби је илустровано системско тестирање ехо апликације коришћењем *Linux Bash* скрипте којом се „симулирају“ кориснички улази и проверавају излази апликације:

1. Отворити командни терминал.
2. Поставити се у фолдер *EchoTest* који се налази у пројекту *Echo* (унутар вашег радног простора (*workspace*)“.
3. Покренути тест скрипту следећом командом: *sh EchoTest_Run.sh*
4. Погледати излаз скрипте и потврдити да сви тестови пролазе.
5. Отворити *EchoTest_Run.sh* у текстуалном едитору и анализирати код.
6. Додати у скрипту 2 тестна случаја која сте написали и додали у претходној вежби (Корак 7 у Вежби 3).
7. Покренути опет скрипту и утврдити да све пролази.

Напредни Це

Напомена: Начин тестирања помоћу скрипте је потпуно независно од тестирања представљеног у Вежби 3, тако да целокупно *EchoTest* окружење засновано на Јунитију и направљено у самом коду више није потребно.

Вежба 5

У овој вежби потребно је подесити механизам за аутоматску проверу усклађености Це кода са MISRA стандардом.

TI компајлер за C6000 породицу процесора нуди могућност провере усклађености кода са MISRA 2004 стандардом (најновија верзија MISRA стандарда је 2012). Обично се за такву врсту провере користе специјализовани алати, или у оквиру компајлера за дату циљну платформу постоји таква могућност. Међутим, GCC не поседује способност такве провере, а тренутно не постоји довољно функционалан специјализовани алат који је бесплатан, или се може без комерцијалне лиценце користити у наставне сврхе. Зато се ослањамо на TI компајлер чију лиценцу имамо. Идеја је да као додатни корак након успешног билдовања програма буде обављено његово поновно превођење TI компајлером са укљученом провером испуњености MISRA 2004 правила.

Ова вежба нема неки посебан крајњи резултат нити проверу, већ служи само да се постави окружење.

1. Прекопирати компајлер за TI C6000 породицу процесора у Ваше окружење.

- Из материјала за вежбу прекопирати садржај директоријума c6000_7.4.23.
- Израз $ТИ_ПУТАЊА означаваће у даљем тексту путању на коју сте прекопирали садржај директоријума

2. Додати у билд процес нов корак након успешног билдовања

- Направити нови пројекат
- У особинама пројекта (Properties) поставити се на следећу путању: C/C++ Build / Settings / Build Steps
- У поље Post-build step / Command унети следећи текст:

```
$ТИ_ПУТАЊА/bin/cl6x --include_path=$ТИ_ПУТАЊА/include --
compile_only --check_misra=all $(C_SRCS)
```

- За потпуно разумевање горње линије погледати у упутству за употребу TI компајлера (spru187u.pdf) поглавља 2.2, 2.3.1 и 6.3.

Напомене:

1. Приликом уноса путање у Post-build step обратити пажњу да је име компајлера cl6x („цл6икс“), не c16x.
2. Потребно је додати привилегије за извршавање. У командном терминалу поставити се на локацију $ТИ_ПУТАЊА/bin/ и покренути следећу команду:

   sudo chmod +x *

Вежба 6

У нови пројекат који је направљен у претходној вежби додати датотеку main.c. У ту датотеку ставити садржај program.c датотеке из Дана 1 (прво вежбање на курсу) и покренути билдовање.

Ако је све постављено како треба, конзолни испис ће бити затрпан извештајем о проблемима са MISRA правилима. Усредсредите се на информације које се тичу main.c датотеке (користите претрагу у прозору са конзолним исписом). **Размислите** о сваком нарушеном правилу и о начину како би се оно могло задовољити. За сада није битно да преправите код нити да за свако нарушено правило имате решење.

Вежба 7

Пробајте да преправите код из претходне вежбе, тако да буде задовољено што је могуће више MISRA правила. За разумевање правила (и одлуке како их најбоље задовољити) помоћи ће вам следећи документи:

- MISRA 2004 стандард (misra-c-2004.pdf)

- Списак правила које је дозвољено прекршити и разлог и околности под којима је то дозвољено (MISRA C 2004 Permits.pdf) и
- Табела пресликавања MISRA 2004 правила на MISRA 2012 правила, са образложењима зашто су нека правила промењена (појачана, релаксирана, избачена или су уведена нова) (MISRA C 2012 Addendum 1 - Rule Mapping.pdf)

У MISRA 2004 стандарду погледајте опис сваког нарушеног правила, а онда пробајте да пронађете то правило и у друга два документа.

Посебну пажњу обратите на следећа правила:

2.2 - Ово правило је у новијој верзији стандарда промењено из обавезног у препоручљиво. Анализирајте разлоге за увођење тог правила и за касније избацивање из скупа обавезних правила. Сами одлучите да ли желите да га се придржавате.

5.7 - Избачено у MISRA 2012 стандарду. Размислите о овом правилу и одлучите да ли ћете га се придржавати.

8.1 - Релаксирано у MISRA 2012 стандарду. Анализирајте шта је промењено у овом правилу између два стандарда. (Фраза „internal linkage" означава да је функција приватна)

17.4 - Релаксирано у MISRA 2012 стандарду и промењено из обавезног у препоручљиво. Анализирајте разлике и размислите о њима. Присетите се различитих начина на које можемо декларисати да функција прима низ као параметар - у томе се крије поштовање овог правила.

20.9 - Да ли је ово правило могуће испоштовати у овом задатку?

Попишите сва правила која су иницијално била нарушена и поред сваког наведите како сте га решили.

За одређена правила закључак може бити да не желимо или не можемо да их задовољимо. Коришћењем командне линије или прагми (поглавља 6.3, 6.9.1 и 6.9.25 у упутству за TI C6000 компајлер) обезбедите да анализатор (TI компајлер) изостави проверу тих конкретних правила, на нивоу целог кода или само дела кода. Задатак је готов када анализатор више не буде пријављивао ни једно нарушено правило у main.c датотеци.

Вежба 8

RT - RK
COMPUTER BASED SYSTEMS

Напредни Це

Играјте се мало и анализирајте разне кодове са ранијих вежби. Посебно обратите пажњу на код из претходне вежбе (Структуре података).