Mittuniversitetet

MID SWEDEN UNIVERSITY

Metoder och verktyg i mjukvaruprojekt

Laboration 3 – Dokumentation & Testning

Laborationen syftar till att ge praktisk erfarenhet av enhetstestning med ramverket Catch samt dokumentation med hjälp av Doxygen!

Innehållsförteckning

Laboration 3

Denna laboration fokuserar på de moment som behandlas av lektionsmaterial 4 och 5, d.v.s. dokumentation samt testning. Dessutom fördjupas användningen av byggskript som du nu själv skall skapa från grunden. I den första uppgiften utförs alla förberedelser som behövs alltigenom laborationen och det är därför viktigt att du studerar informationen noggrant.

I den andra uppgiften får du en uppsättning testfall för vilka du skall skapa implementationen. Tanken är att du ska få ett smakprov på hur mycket hjälp bra tester kan ge. Tänk framför allt på när du har arbetat med en större mängd kod under en längre tid och behöver göra någon större ändring. Du kan alltid köra testskriptet och få omedelbar feedback på vad dina ändringar hade för effekt.

I den tredje uppgiften är rollerna i stort sett omvända. Nu är det du som ska skapa testerna. Dina tester kommer då att köras mot minst en färdig implementation (av en kalkylator). Tanken är att du ska få öva på att skriva tester för kod som ännu inte finns. Det är något du måste kunna göra för att använda någon form av TDD.

Basrepot för laborationen finner du i kursens **Bitbucket Team** och det innehåller allt du behöver för att genomföra laborationsarbetet. Klona detta repo till din lokala dator:

```
$ git clone git@bitbucket.org:miun_dt042g/studid_laboration_3_vt21.git
```

Utför löpande commits alltigenom arbetet tillsammans med beskrivande meddelanden. Det skall vara möjligt att få en överskådlig bild av arbetet utifrån denna historik!

Uppgift 1 - Förberedelse

Lägg till `.gitignore`, tillsammans med lämpligt innehåll, samt en `README.md` under projektroten. Dokumentation i form av **HTML output** skall genereras med hjälp av **Doxygen** och för att ge god projektstruktur skall konfigurationsfilen (`doxyfile`) placeras under mappen `Docs/` som du finner under projektroten. Det är viktigt att denna konfigurationsfil blir inkluderad i versionshanteringen och att relativa sökvägar används, exempelvis för `OUTPUT_DIRECTORY`. Tänk dock på att den genererade dokumentationen (**HTML**) måste uteslutas från versionshantering.

```
project_root/
   Assignment_2/
      ...
   Assignment_3/
      ...
   Docs/
      html/
      images/
      Doxyfile
      main_page.dox
   .gitignore
   CMakeLists.txt
   README.md
```

Under `Docs/` finner du en fil vid namn `main_page.dox` som saknar textinnehåll. Detta dokument skall fungera som dokumentationens huvudsida och behöver därför tillföras lite information. Innehållsvärdet för denna sida är inte viktig, men taggen `@mainpage` måste användas tillsammans med lite text du själv väljer.

Lek gärna runt med andra taggar och ett tips kan vara att använda taggen `@section` för att ge ytterligare struktur, samt att referera till externa sidor (`@page`) med hjälp av `@ref`. Det kan även vara intressant att inkludera bilder på sidan men tänk då på att placera alla bildfiler i en dedikerad mapp under `Docs` samt att relativa sökvägar används i konfigurationsfilen.

Projektet saknar ett byggskript som kommer vara nödvändigt för att kunna kompilera lösningarna för resterande uppgifter. Vid det här laget bör du vara tämligen van vid **CMake** och det blir därför din uppgift att själv skapa det byggskript som kommer behövas. Börja med att skapa en `CMakeLists.txt` under projektroten och tillför lämpligt basinnehåll. Kom ihåg att inkludera miljövariabeln `TOOLS_INCLUDE` eftersom du kommer behöva åtkomst till biblioteket **Catch**. Du behöver inte skapa några *run configurations* ännu, utan tar sådana allteftersom de behövs i kommande uppgifter!

Redovisa genomförandet av första uppgiften genom att lägga till beskrivande information i `README.md` under rubrik **Uppgift 1** och lägg gärna till dina egna reflektioner och synpunkter kring uppgiften!

Uppgift 2 - Stack

Testskriptet i **`/Assignment_2/test/test-stack.cpp`** testar en **`Stack`**. Skapa en implementation som klarar alla tester. Stacken skall vara baserad på en egen implementation av en länkad lista, d.v.s. du får inte använda **`array`** eller standardbibliotekets länkade lista. Modifiera eller ersätt **`/include/stack.h`** med din egen version och skapa nödvändiga *run configurations* i byggskriptet. Lägger du till fler filer så skall god struktur bibehållas där **`.h`** placeras under include samt **`.cpp`** under **`src/`**. Tänk även på att enbart en **`main()`** får existera inom scope för exekvering så testerna måste byggas fristående, d.v.s. att en eventuell annan **`main()`** utesluts för denna exekvering.

> *Observera att testkoden **inte** skall modifieras!*

Redovisa genomförandet av andra uppgiften genom att lägga till beskrivande information i **`README.md`** under rubrik **Uppgift 2** och lägg gärna till dina egna reflektioner och synpunkter kring uppgiften!

Uppgift 3 - Kalkylator

Kika på den gamla versionen av laborationen **`Labb_testning_old.pdf`** som du finner under mappen för uppgiften **`/Assignment_3/`**. I den här versionen av uppgiften har du, till skillnad från den gamla, en existerande implementation och din uppgift blir att testa denna. Stilen är upp till dig… du kan använda **`TEST_CASE/SCENARIO`** och / eller **`SCENARIO / GIVEN / WHEN / THEN`** som du själv önskar. Lägg till din testkod i **`/Assignment_3/test/test-calc.cpp`**. Skapa nödvändiga *run configurations* i byggskriptet, d.v.s. minst två… en för testerna samt en för implementationen.

> *I den här uppgiften skall du alltså **inte** implementera någon kalkylator, utan du skall testa en. Du **får** naturligtvis även implementera en om du vill provköra ditt testskript mot den – testa dina tester, så att säga – men det går egentligen lite emot poängen med uppgiften.*

Både testkoden och resulterande felmeddelanden skall vara så tydliga och snygga som möjligt, och testfallen skall kunna urskilja korrekta eller felaktiga implementationer. Med andra ord skall testet kunna finna buggar och för att bli godkänd måste du hitta **minst** två av förekommande buggar. Uppgiften kommer även bedömas baserat på hur testkoden ser ut.

> *Ett tips är att titta på hur man bör hantera flyttal.*

Prototyperna under **`/Assignment_3/include/MathExpression.h`** skall dokumenteras med taggen **`@test`** som anger vad som skall testas och buggar som har hittats skall dokumenteras med taggen **`@bug`**. Du får använda fler dokumentationstaggar om du vill, men **Doxygen** skall iallafall prydligt generera separata listor för tester och buggar. Kom ihåg att ange relativa sökvägar i konfigurationsfilen!

Redovisa genomförandet av tredje uppgiften genom att lägga till beskrivande information i **`README.md`** under rubrik **Uppgift 3** och lägg gärna till dina egna reflektioner och synpunkter kring uppgiften!

Redovisning

Utför en formell inlämning i avsedd inlämningslåda i kursens Moodle-instans. Projekttillstånd och nödvändiga filer hämtas direkt från ditt dedikerade studentrepo, så se till att detta repo är uppdaterat innan inlämningen. Om dedikerade arbetsförgreningar har använts under arbetet är det viktigt att ändringar sammanförs med **master** inför redovisningen!

Arbetet skall bedrivas enskilt med individuell redovisning, och pararbete är inte tillåtet!