Eindopdracht 1

Veel embedded applicaties beschikken over een command line interface (CLI). Deze interface is meestal niet bedoeld voor de eindgebruiker van het embedded product, maar kan bijvoorbeeld bij onderhoudswerkzaamheden worden gebruikt om:

- log gegevens op te vragen;
- bepaalde tests uit te voeren;
- bepaalde instellingen te wijzigen.

Als eerste eindopdracht voor EMS31 ga je zo'n CLI voor de CC3220S LaunchPad ontwerpen, implementeren en uitgebreid testen. Je laat hiermee zien dat je de leerdoelen uit week 1 tot en met 6 hebt behaald en dat je een betere C-programmeur bent geworden.

Je laat zien dat je hebt geleerd hoe je:

- de *interface* van een module in een `.h` bestand kunt *declareren*;
- de *implementatie* van een module in een `.c` bestand kunt *definiëren*;
- deze module kunt gebruiken en testen op een pc met behulp van professionele tools zoals Windows Subsystem for Linux (WSL), Visual Studio Code, make en CMake;
- snel en efficiënt kunt werken met WSL door een aantal eenvoudige Linux-commando's te gebruiken zoals `cd`, `wget`, `unzip`, `rm`, `mkdir` en `cp`;
- C-code systematisch kan testen met behulp van het eenvoudige test-framework GoogleTest;
- een ontwikkelmethode voor software genaamd Test Driven Development (TDD) kan toepassen;
- code die je in meerdere applicaties wilt gebruiken in een Git submodule kan plaatsen;
- zo'n submodule kunt gebruiken onder Windows Subsystem for Linux;
- zo'n submodule kunt gebruiken in Code Composer Studio;
- dynamisch geheugen kunt alloceren en weer vrij kunt geven;
- een FIFO-buffer kunt implementeren die dynamisch groeit en krimpt met behulp van een *singly linked list*;
- *stubs* kunt gebruiken om te testen of jouw code goed werkt als het geheugen vol is en om te testen of jouw code het gealloceerde geheugen netjes vrijgeeft;
- een FIFO-buffer als een *user defined type* kunt definiëren waardoor je meerdere buffers in een programma kunt gebruiken;

- softwareontwikkeling kan doen volgens de zogenoemde Trunk-based development met Feature flags methode;
- gelijktijdig meerdere nieuwe features voor een embedded applicatie kunt ontwikkelen;
- fouten bij het gebruik van dynamische geheugenallocatie zoals geheugenlekken kunt opsporen met speciale tools zoals valgrind;
- weet dat je genoeg testcode hebt geschreven door gebruik te maken van test coverage metingen;
- gebruik kunt maken van verschillende *C coding standards*;
- gebruik kunt maken van verschillende *static code analysis* tools, waaronder Cppcheck, om de kwaliteit van je C-code te verbeteren;
- een datastructuur kunt implementeren die dynamisch groeit en krimpt met behulp van `struct`s en pointers.

De te ontwikkelen CLI kan gebruikt worden om de leds van de CC3220S LaunchPad mee aan te sturen. We beschouwen de 3 leds als een binair getal waarbij groen het meest significante bit en rood het minst significante bit voorstelt. De waarde zes (binair 110) wordt dus aangegeven door alleen de groene en gele led te laten branden. De CLI moet beschikken over de volgende specifieke commando's:

`nul`	zet de decimale waarde 0 op de leds
`een`	zet de decimale waarde 1 op de leds
`twee`	zet de decimale waarde 2 op de leds
`drie`	zet de decimale waarde 3 op de leds
`vier`	zet de decimale waarde 4 op de leds
`vijf`	zet de decimale waarde 5 op de leds
`zes`	zet de decimale waarde 6 op de leds
`zeven`	zet de decimale waarde 7 op de leds
`schakelroodaan`	zet de rode led aan
`schakelrooduit`	zet de rode led uit
`schakelroodom`	toggle de rode led
`schakelgeelaan`	zet de gele led aan
`schakelgeeluit`	zet de gele led uit
`schakelgeelom`	toggle de gele led
`schakelgroenaan`	zet de groene led aan
`schakelgroenuit`	zet de groene led uit
`schakelgroenom`	toggle de groene led

Verder moet de CLI ook beschikken over de volgende generieke commando's:

`einde`	sluit het programma af
`af`	sluit het programma af

De commando's moeten 'case insensitive' zijn. Dat wil zeggen dat de commando's een, `EEN`, `Een` en `eeN` allemaal hetzelfde zijn. De CLI moet na het intypen van een commando gevolgd door het indrukken van de Enter-toets het ingetypte commando uitvoeren. Als het ingetypte commando niet bestaat, dient een passende foutmelding te worden gegeven. De CLI moet gebruikt kunnen worden via de UART-interface van de CC3220S LaunchPad. We maken gebruik van het terminal programma Tera Term[1], zodat we ANSI/VT100 terminalcommando's[2] kunnen gebruiken. Elk printbaar karakter moet na ontvangst teruggestuurd (geëchood) worden over de UART-verbinding. Als de gebruiker op de Backspace-toets drukt dan moet het laatst ontvangen karakter genegeerd worden. Je kunt de karakterstring `"\x1b[D \x1b[D"` terugsturen om er voor te zorgen dat het karakter ook in het terminalprogramma wordt gewist. Niet printbare karakters die geen speciale functie hebben in de CLI moeten genegeerd worden.

Om deze CLI gebruiksvriendelijk te maken, *kunnen* de volgende extra features geïmplementeerd worden:

- **alias commando**
  Voeg een commando genaamd `alias` toe waarmee een *bestaand* commando een alternatieve naam kan krijgen. Bijvoorbeeld na het invoeren van het commando:

  ```
  alias zero nul
  ```

  kan het commando `zero` gebruikt worden, als 'alias' van het al bestaande commando `nul`, om de leds uit te zetten. Het commando `nul` kan ook nog steeds gebruikt worden om de leds uit te zetten. Zorg voor een passende foutmelding als het commando `alias` niet correct wordt gebruikt.
- **command history**
  Het is gemakkelijk voor de gebruiker als een eerder ingevoerd commando eenvoudig herhaald kan worden. Dit kan geïmplementeerd worden door de uitgevoerde

[1] Zie https://ttssh2.osdn.jp/index.html.en.

[2] Deze commando's kun je bijvoorbeeld gebruiken om de tekst in kleur weer te geven of om de pijltjestoetsen te verwerken, zie https://www2.ccs.neu.edu/research/gpc/VonaUtils/vona/terminal/vtansi.htm.

commando's in een LIFO-queue[3] op te nemen. De gebruiker kan dan met de ↑- en ↓-toets door deze queue 'heenlopen'.

E1.1 Implementeer de CLI die de bovenstaande commando's kan uitvoeren, *zonder* de hierboven beschreven extra features op de CC3220S LaunchPad. De commando's met hun bijbehorende functie *moeten* in een dynamische boomstructuur worden opgeslagen. Zodat de programmeur op eenvoudige wijze commando's aan de CLI kan toevoegen. Op pagina 5 worden nog enkele implementatietips gegeven. Ontwikkel en test deze software zoveel mogelijk op de pc, door, waar nodig, gebruik te maken van stubs[4]. Ontwikkel de software zoveel mogelijk volgens de TDD-methode[5]. Maak daarbij gebruik van GoogleTest.

E1.2 Controleer de ontwikkelde software op geheugenfouten. Maak daarbij gebruik van valgrind.

E1.3 Zorg ervoor dat je voldoende testcode hebt geschreven om (bijna) alle ontwikkelde software te testen. Maak daarbij gebruik van test coverage metingen.

E1.4 Verbeter de kwaliteit van de ontwikkelde software door gebruik te maken van static code analysis tools en coding standards. Maak daarbij op zijn minst gebruik van Cppcheck.

E1.5 Schrijf een kort verslag waarin je de belangrijkste ontwerpbeslissingen en alle testresultaten (zowel op de pc als op de CC3220S LaunchPad) beschrijft.

Met bovenstaande opdrachten kun je maximaal 70 punten verdienen. De overige 30 punten kun je verdienen door de onderstaande opdrachten uit te voeren:

[3] Een LIFO-queue (Last In First Out) wordt ook wel een stack genoemd.

[4] Zie https://nl.wikipedia.org/wiki/Stub.

[5] Bij de zogenoemde 'Test Driven Design'-methode wordt eerst een falende test geschreven. Vervolgens wordt de code geschreven om de test te laten slagen. Hierna wordt de code indien nodig vereenvoudigd en beter leesbaar en onderhoudbaar gemaakt (dit wordt in het Engels 'code refactoring' genoemd) en weer getest. Hierna wordt weer een falende test geschreven en wordt de beschreven cyclus herhaald.

E1.6 Implementeer de hierboven beschreven extra features van de CLI: alias commando en command history op de CC3220S LaunchPad. Maak hierbij gebruik van de Git workflow "Trunk-based development met Feature flags". Ontwikkel en test deze features zoveel mogelijk op de pc volgens de TDD-methode met behulp van GoogleTest. Op pagina 7 wordt nog een implementatietip voor de command history feature gegeven.

E1.7 Controleer de toegevoegde code op geheugenfouten. Maak daarbij gebruik van valgrind.

E1.8 Zorg ervoor dat je voldoende testcode hebt geschreven om (bijna) alle toegevoegde code te testen. Maak daarbij gebruik van test coverage metingen.

E1.9 Verbeter de kwaliteit van de toegevoegde software door gebruik te maken van static code analysis tools en coding standards. Maak daarbij op zijn minst gebruik van Cppcheck.

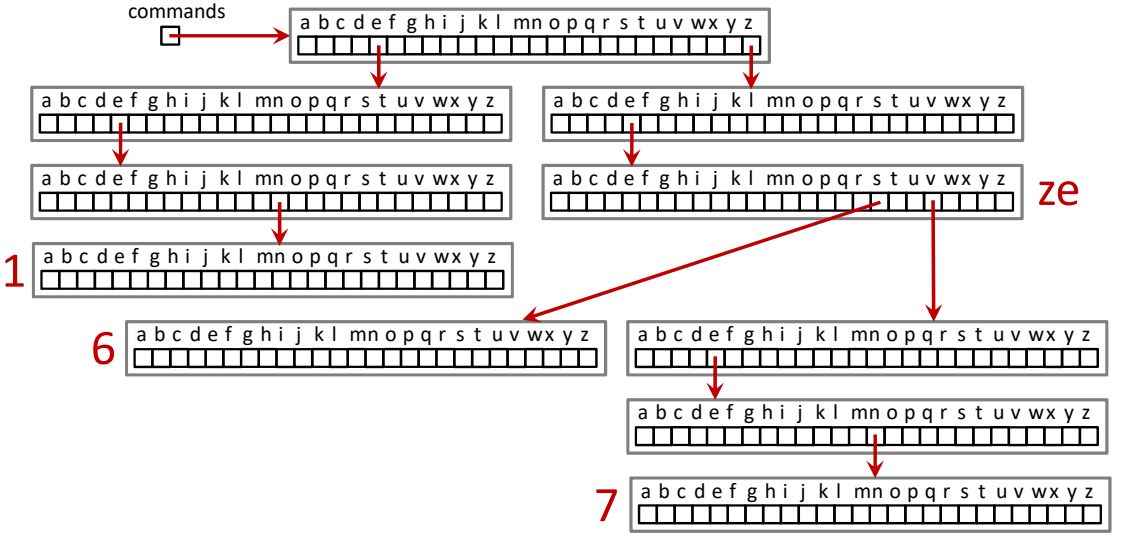
E1.10 Schrijf een kort verslag waarin je de belangrijkste ontwerpbeslissingen en alle testresultaten (zowel op de pc als op de CC3220S LaunchPad) van de toegevoegde features beschrijft.

Implementatietips

Karakters die binnenkomen via een UART-verbinding kun je meteen inlezen in je programma. Als je een programma uitvoert op de pc in een CLI zoals de terminal in WSL, dan worden ingetypte karakters pas doorgestuurd naar het programma als je op de Enter-toets drukt. Dit wordt de *cooked mode* genoemd. Voor deze applicatie is dat niet handig. Je kunt ervoor zorgen dat karakters wel meteen worden doorgegeven aan het programma, zodra ze ingetypt worden, door de zogenaamde *raw mode* te gebruiken. Hoe je dit doet is afhankelijk van het platform en het type terminal dat je gebruikt en kun je vinden op `https://bitbucket.org/HR_ELEKTRO/ems31/wiki/RawTerminal.md`. Ingetypte karakters kunnen vervolgens meteen worden ingelezen met behulp van de functie `getchar`.

Als datastructuur om de commando's in op te slaan moet je gebruik maken van een met dynamische geheugenallocatie aangemaakte zogenoemde 'boom' (Engels: *tree*). Een tree bestaat uit een zogenoemde *root* met vertakkingen (pointers) naar volgende *nodes*. In dit geval bevat elke node 26 vertakkingen (pointers): één voor elke letter uit het alfabet. Je

kunt dan ingetypte commando's eenvoudig vinden door per letter de betreffende vertakking (pointer) te volgen, zie figuur 1. Alle 'lege' pointers in figuur 1 bevatten de waarde NULL.



Figuur 1: Een mogelijke datastructuur om de commando's in op te slaan.

Het type van elke node in de tree zou dan als volgt gedefinieerd kunnen zijn:

```
typedef void (*command_function)(void);

typedef struct command_node_tag {
    command_function function;
    struct command_node_tag *next[26];
} command_node;
```

Als de command_node een geldig commando vertegenwoordigt, dan:

- verwijst de functiepointer `function` naar de functie die aangeroepen moet worden als het betreffende commando uitgevoerd moet worden.
- verwijzen de 26 pointers uit de array `next` naar commando's met hetzelfde begin als het betreffende commando.

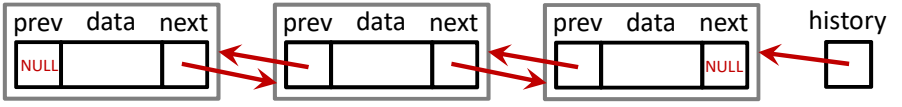
Als de command_node geen geldig commando vertegenwoordigt, dan is de pointer `function` gelijk aan NULL.

Als de CLI alleen de commando's een, zes en zeven zou bevatten, dan zou de tree opgebouwd zijn zoals gegeven in figuur 1. De command_node gemarkeerd met 1 in figuur 1 bevat dan de functiepointer naar de functie die moet worden uitgevoerd als het commando een is ingetypt.

De command_nodes gemarkeerd met 6 en 7 bevatten respectievelijk de functiepointer voor de commando's zes en zeven.

De hier beschreven implementatiemethode met behulp van een tree maakt de implementatie relatief eenvoudig, maar kost wel veel geheugen. Je kunt ook voor een implementatie kiezen die minder geheugenruimte kost, bijvoorbeeld een alfabetisch geordende binaire boom[6] van commando's.

De command history feature kan relatief eenvoudig gerealiseerd worden met behulp van een zogenoemde *doubly linked list*. Elk element (vaak *node* genoemd) uit zo'n lijst bevat een pointer genaamd `next` naar het volgende en ook een pointer genaamd `prev` naar het vorige element, zie figuur 2. De pointer `data` wijst naar het betreffende ingevoerde commando.



Figuur 2: Een mogelijke datastructuur om de command history in op te slaan.

Het type van elke node in de doubly linked list zou dan als volgt gedefinieerd kunnen zijn:

```
typedef struct history_node_tag {
    char *command;
    struct history_node_tag* prev;
    struct history_node_tag* next;
} history_node;
```

De pointer `history` wijst naar het laatst ingevoerde commando. Met behulp van de `prev`- en `next`-pointers kan relatief eenvoudig door de eerder ingevoerde commando's 'heengelopen' worden.

In figuur 3 is te zien hoe de CLI op een pc zou kunnen werken.

Inleveren

De code moet worden ingeleverd in Bitbucket via Git in de aan jullie groep toegekende repositories. Het verslag moet per groep worden ingeleverd in Pdf-formaat in Brightspace bij Eindopdracht 1.

[6] Zie eventueel: https://en.wikipedia.org/wiki/Binary_tree.

Figuur 3: Een mogelijke implementatie met behulp van stubs van de CLI op een pc.