**50**

HBO ICT

TI

*Assignments*

C++ Programming & Software Engineering 1

**TICT-V2CPSE1-16**

Studiejaar 2017-2018

|  |  |
| --- | --- |
|  |  |
| Cursuseigenaar | Wouter van Ooijen |
| **Auteur(s)** | Wouter van Ooijen |
| **Datum** | 2017-08-28 |
| Versie | 2.0 |
|  | |

© Institute for ICT, Hogeschool Uterecht, 2017

# Algemeen

De regeling omtrent de practica is beschreven in de studentenhandleiding.

# Week 1: ringtone (C++)

## Opbouwen en bestuderen

De ringtone demo is een applicatie voor de Arduino Due die een ringtone in rtttl formaat afspeelt via een luidsprekertje (met een 100 Ohm serieweerstand!) op pin d7. Bouw de schakeling en probeer de demo met een andere ringtone. Op <http://www.picaxe.com/RTTTL-Ringtones-for-Tune-Command/> vind je een groot aantal rtttl files. Bestudeer hoe de demo in elkaar zit. Bekommer je daarbij niet om het decoderen van de rtttl string, maar let wel op de opdeling tussen player (de luidspreker die kan piepen) en play (een functie die een rtttl string kan afspelen op een player).

## RTTTL string in de Makefile

Maak een copie van de ringtone demo en bouw die om zodat de ringtone (die string met rare tekentjes) in de Makefile staat, en het decoderen van de ringtone string in een PC (Windows) applicatie gebeurt. In de embedded applicatie komt dus alleen maar een reeks aaroepen van player.play() te staan. Hiertoe moet je op de PC de ringtone player zo aanroepen dat hij niet piept maar de file met aanroepen van player.play() produceert. Dit kan je doen zonder de play functie te wijzigen, door een eigen player te schrijven.

## RTTTL file naam in de makefile

Maak vervolgens een andere copie waarin de ringtone wordt gespecificeerd door in de Makefile de ringtone file op te geven. Ook in dit geval moet het decoderen van de ringtone string op de PC gebeuren, niet op de Arduino.

# Week 2: print\_asciz (assembler)

Ga uit van het asm print project (maak er een copie van). Test dit project en bestudeer hoe het werkt.

1. Vervang de print\_asciz functie door de equivalente assembler code. Stop dit in een .asm file, en voeg die file name toe aan de SOURCES in de lokale Makefile. Merk op date er nog een paar details ontbreken.

|  |
| --- |
| **print\_asciz:**  **push { r5, lr }**  **mov r5, r0**  **loop:**  **ldrb r0, [ r5 ]**  **add r0, r0, #0**  **beq done**  **bl put\_char**  **add r5, r5, #1**  **b loop**  **done:**  **pop { r5, pc }** |

2. Neem wat je nu net gemaakt hebt (met print\_asciz in assembler) en vervang nu ook de application() functie door een equivalente subroutine in assembler.

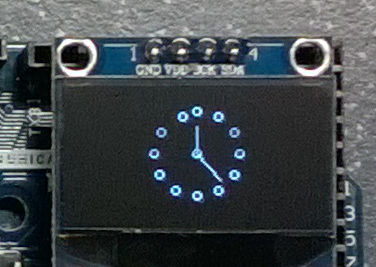
3. Maak een C functie (dus met extern ”C”, of in een extern “C” block) die zijn char argument geconverteerd teruggeeft: een uppercase character wordt lowercase, en een lowercase character wordt uppercase. Andere characters blijven ongewijzigd. Roep deze functie aan in de print\_asciz subroutine om elk charprint te converteren voordat het door put\_char wordt geprint, zodat ”Hello world, the ANSWER is 42!\n” wordt geprint als ”hELLO WORLD, THE answer IS 42!\n!”. Gebruik de test string die in het voorbeeld programma staat, daar staan ook de characters in die net voor en net na de letters staan in de ASCII tabel.

4. Vervang deze C functie door een equivalente assembler subroutine. Let op: je kan in ADD en SUB instructies geen 8-bit literal gebruiken, dus zo’n waarde moet je eerst in een register zetten met een MOV instructie.

# week 3: NEC IR decoder (C++, rtos)

De RTOS reader beschrijft een ontwerp (initial en final class diagram, concurrency diagram, State Transition diagrams). Vertaal dit ontwerp op de in de reader beschreven manier naar een C++ applicatie met 3 tasks.

# week 4: constexpr klokje (C++)



Schrijf een applicatie voor op de Due die op het OLEDje een analoog klokje laat zien met twee wijzers (minuten en uren). Je mag natuurlijk nog een seconden wijzer toevoegen. De plekken van de cijfers (of symbolen) en de eindpunten (en evetueel ook de beginpunten) van de wijzers moet je uitrekenen dmv. sinussen en cosinussen.

Je mag zelf bepalen wat je op de plekken van de cijfers zet (de cijfers, een streepje, een cirkeltje, etc.), maar er moet daar wel iets staan. Als je de cijfers wil tekenen kan je kijken naar hwlib:: font\_default\_8x8, dit zijn plaatjes van de ASCII tekens die ook door de console klasse worden gebruikt.

**Zorg dat je geen run-time floating-point gebruikt.** Gebruik daartoe een constexpr opzoektabel die tijdens het compileren gevuld wordt. Zorg dat je het klokje gelijk kan zetten met een drukknopje per wjzer.

Gebruik de gebufferde versie van de oled klasse. Het tekenen en flushen kost tijd, dus om je klokje redelijk gelijk te laten lopen moet je geen gebruik maken van een delay, maar steeds de huidige tijd opvragen en die gebruiken (gecorrigeerd met een waarde voor het gelijk zetten).

In de voorbeelden zit een klasse die het gerbuik van het oled op de pinnen D18..D21 wat makkelijker maakt.

Extra uitdagingen:

Het is niet perse nodig de sinus en cosinus waarden voor alle hoeken op te slaan, je hebt genoeg aan de sinus (of cosinus) waarden voor 0 .. 45 graden (andere waarden kan je daar uit afleiden door slim te spiegelen), in stappen van 1 tijd-seconde (6 graden).

Naast de analoge klok is er nog ruimte op het oled om de tijd ook digitaal weer te geven.

# week 5: LZ decoder (assembler)

Einddoel: een LZ decoder in assembler op de Arduino.

Stappen:

1. Neem de LZ encoder en breidt die uit met een decompressor/decoder (in C++, op de PC) die zijn uitvoer (via en lambda parameter) naar std::cout schrijft.
2. Pas de aanroep van de LZ compressor/encoder aan zodat er een assembler file wordt gegenereerd die de gecodeerde characters bevat (en die in een volgende stap meegenomen kan worden in een Due applicatie).
3. Schrijf een Due assembler subroutine die de characters die in de vorige stap gegenereerd zijn decodeert (naar keuze naar de standard output of naar de oled).
4. Pas de Makefile aan zodat het coderen/comprimeren en het bouwen van de assembler applicatie automatisch gebeurt als dat nodig is.