Opdrachten week 7 les 2 – Arrays in C

In deze les ga je jezelf verder verdiepen in de programmeertaal C. Je leert hoe je:

- meerdere variabelen van hetzelfde type kan samennemen in een array;
- de elementen van een array één voor één kan benaderen;
- een array kan meegeven en bewerken in een functie;
- een array kunt gebruiken om samples van de ADC in op te slaan;
- een laagdoorlaatfilter in software kunt implementeren door het voortschrijdend gemiddelde te berekenen.

In les 1 van week 3 heb je kennisgemaakt met het datatype `list` in Python. In een variabele van dit datatype kan je meerdere variabelen samennemen, die je dan vervolgens met een index kan adresseren. De programmeertaal C kent geen ingebouwd datatype `list`. In plaats daarvan kun je een zogenoemde array gebruiken. De belangrijkste verschillen tussen een lijst in Python en een array in C zijn:

- Een lijst kan elementen van verschillende typen bevatten. Alle elementen van een array moeten van hetzelfde type zijn.
- Een lijst is dynamisch, dat wil zeggen dat het programma elementen aan een lijst kan toevoegen en/of elementen uit een lijst kan verwijderen. Een array is statisch, dat wil zeggen dat het aantal elementen nadat de array is aangemaakt niet meer gewijzigd kan worden.
- Een lijst kan bewerkt worden met operatoren en methodes. Een array heeft geen operatoren en geen methodes.
- De elementen van een lijst kunnen benaderd worden met behulp van indexering met een positieve index (van voor naar achter: index `0`, `1`, `...`) of met een negatieve index (van achter naar voor: index `-1`, `-2`, `...`). De elementen van een array kunnen alleen benaderd worden met behulp van indexering met een positieve index (van voor naar achter: index `0`, `1`, `...`).

Python en C kennen beide een **`for`**-statement. Met behulp van het **`for`**-statement in Python kun je rechtstreeks door de elementen van de lijst `l` itereren door middel van de code **`for`** `e` **`in`** `l:`. In C is dit niet mogelijk. Wel is het in C mogelijk om met behulp van indexering door de elementen van een array die `n` elementen bevat te itereren met de code **`for`** `(i = 0; i < n; i++)`. Dit komt overeen met de Python code **`for`** `i in range(0, n):`.

Lees nu paragraaf 2.4 punt 8 en paragraaf 2.5 van het boek[1].

Je begint deze les met enkele basisoefeningen met arrays in C. Daarbij maak je gebruik van de debugger van CCS om de correcte werking van je programma te verifiëren.

7.2.1 Maak in CCS een nieuw project aan met de naam `opdr_7.2.1`. Om te voorkomen dat bepaalde variabelen weggeoptimaliseerd worden, moet de optimizer uitgezet worden. Kies de menu-optie Project › Properties. Kies vervolgens (aan de linkerkant) voor Build › MSP430 Compiler › Optimization en zet het 'Optimization level' op 'off'. Kopieer de code die gegeven is in listing 1 in de file `main.c`. Voeg de benodigde code toe en test het programma door een breakpoint te zetten op de `return 0`-instructie. Je kunt dan in het 'Variabeles window' de waarde van de variabele `som` zien. Als het goed is, dan heeft de variabele `som` de waarde 55.

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int som = 0;

    // Vul hier je code in zodat de variabele som gelijk
    // wordt aan de som van alle getallen in de array a

    return 0;
}
```

Listing 1: Code bij opdracht 7.2.1, zie opdr7.2.1.c

In listing 1 wordt de array a gedefinieerd en meteen geïnitialiseerd:

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
```

In dit geval kan de compiler het aantal elementen in de array zelf bepalen. Je kunt dus ook de volgende code gebruiken:

```
int a[] = {1,2,3,4,5,6,7,8,9,10};
```

[1] Carl Burch. *C for Python programmers*. 2011. URL: http://www.cburch.com/books/cpy/.

Je kunt ook een array definiëren zonder deze meteen te initialiseren. Bijvoorbeeld een array genaamd b die bestaat uit 20 integers kun je als volgt definiëren:

```
int b[20];
```

Je kunt het aantal elementen van een array door de compiler laten uitrekenen met behulp van de **sizeof**-operator[2]. De array b kun je bijvoorbeeld met de onderstaande code vullen met nullen:

```
int i;
for (i = 0; i < sizeof b/sizeof b[0]; i++)
{
    b[i] = 0;
}
```

In de bovenstaande code wordt het aantal bytes waaruit de array b bestaat (**sizeof** b) gedeeld door het aantal bytes waaruit het eerste element van de array b bestaat (**sizeof** b[0]). Het resultaat van deze deling is, uiteraard, het aantal elementen dat de array b bevat. Het voordeel van het gebruik van **sizeof** is dat je de **for**-loop niet aan hoeft te passen, als je de grootte van de array b bij de definitie aanpast.

7.2.2 Schrijf een C programma waarin een array b van 30 integers wordt aangemaakt. Vul deze array met de kwadraten van de indexen, dus 0, 1, 4, 9 enz., met behulp van een **for**-loop. Bepaal daarna de som van alle getallen in de array b met behulp van een aparte **for**-loop. Test het programma door een breakpoint te zetten op de **return** 0-instructie. Als het goed is, dan is de som van alle getallen in de array gelijk aan 8555.

In de volgende opdrachten ga je arrays en functies met elkaar combineren. Je gaat eerst bekijken wat er gebeurt als je een array als functieparameter gebruikt. Als je in C een gewone parameter gebruikt, bijvoorbeeld een **int**, dan wordt het argument dat bij de aanroep aan de functie wordt meegegeven *gekopieerd* naar de parameter. Als deze parameter een nieuwe waarde krijgt in de functie, dan blijft het argument in de aanroepende code *ongewijzigd*. Deze methode wordt 'call by value' genoemd omdat de *waarde* van het argument wordt doorgegeven aan de parameter. Bij een array gaat dit anders. Als je een array als parameter gebruikt, dan wordt het argument (dat moet dan vanzelfsprekend ook een array zijn) *niet* gekopieerd. In plaats daarvan wordt het beginadres van de array die als argument wordt

[2] Zie eventueel http://en.cppreference.com/w/c/language/sizeof.

gebruikt doorgegeven aan de parameter. Als de elementen van deze parameter een nieuwe waarde krijgen in de functie, dan is het argument in de aanroepende code ook *gewijzigd*. Deze methode wordt 'call by reference' genoemd omdat het *adres* van het argument wordt doorgegeven aan de parameter[3]. Het is belangrijk om het verschil tussen 'call by value' en 'call by reference' goed te begrijpen.

7.2.3 Maak in CCS een nieuw project aan met de naam `opdr_7.2.3`. Om te voorkomen dat bepaalde variabelen weggeoptimaliseerd worden, moet de optimizer uitgezet worden, zie opdracht 7.2.1. Kopieer de code die gegeven is in listing 2 in de file `main.c`. Test het programma door een breakpoint te zetten op de **`return`** `0`-instructie. Je kunt dan in het 'Variabeles window' de waarde van de variabelen `argument` en `argument_array` zien.

```
#include <msp430.h>

void test1(int parameter)
{
    parameter = 0;
}

void test2(int parameter_array[])
{
    parameter_array[0] = 0;
    parameter_array[1] = 0;
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    int argument = 1;
    test1(argument);
    int argument_array[] = {2, 3};
    test2(argument_array);
    return 0;
}
```

Listing 2: Code bij opdracht 7.2.1, zie `opdr7.2.3.c`

[3] Het adres verwijst (oftewel refereert) naar de variabele (in dit geval een array). Vandaan de naam 'call by reference'.

A De variabele `argument` krijgt in de functie `main` de waarde 1. Vervolgens wordt de functie `test1` aangeroepen met de variabele `argument` als argument. In de functie `test1` wordt de parameter genaamd `parameter` gelijk gemaakt aan 0. Wat is de waarde van de variabele `argument` aan het einde van het programma? Is hier sprake van 'call by value' of van 'call by reference'?

B De elementen van de array genaamd `argument_array` krijgen in de functie `main` de waarden 2 en 3. Vervolgens wordt de functie `test2` aangeroepen met de variabele `argument_array` als argument. In de functie `test2` worden de elementen van de parameter genaamd `parameter_array` gelijk gemaakt aan 0. Wat is de waarde van de elementen van de array `argument_array` aan het einde van het programma? Is hier sprake van 'call by value' of van 'call by reference'?

7.2.4 Maak in CCS een nieuw project aan met de naam opdr_7.2.4. Kopieer de code die gegeven is in listing 3 in de file `main.c`. Dit programma bevat de functie `maak_array4_nul` die de eerste 4 elementen van de parameter array genaamd a nul maakt. Test het programma door een breakpoint te zetten op de **`return 0`**-instructie.

```
#include <msp430.h>

void maak_array4_nul(int a[])
{
    int i;
    for (i = 0; i < 4; i++)
    {
        a[i] = 0;
    }
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    int rij1[4] = {1, 2, 3, 4};
    maak_array4_nul(rij1);
    return 0;
}
```

Listing 3: Code bij opdracht 7.2.4, zie opdr7.2.4.c

Stel dat je nu een functie wilt maken genaamd `maak_array_nul` die *alle* elementen van een parameter array genaamd a nul maakt. Het is echter niet mogelijk om het aantal elementen van de parameter array a in de functie te bepalen. Omdat alleen het beginadres van de als argument gebruikte array wordt doorgegeven aan de functie, geeft de expressie **`sizeof`** a altijd de waarde 2 onafhankelijk van het aantal elementen dat de array bevat[4]. Probeer zelf maar! Je moet dus het aantal elementen van de array als tweede parameter van de functie definiëren. Bij aanroep kun je dan de array en het aantal elementen van de array als twee aparte argumenten meegeven aan de functie. Implementeer de functie `maak_array_nul` en test deze functie met arrays van verschillende lengten. Tip: Als je deze functie als volgt aanroept, berekent de compiler de lengte van de array `rij`:

```
maak_array_nul(rij, sizeof rij / sizeof rij[0]);
```

7.2.5 Maak in CCS een nieuw project aan met de naam `opdr_7.2.5`. Kopieer de code die gegeven is in listing 4 in de file `main.c`. Dit programma bevat de functie `som` die de som bepaald van de eerste `n` elementen van de array `a`. Voeg de benodigde code toe en test het programma door een breakpoint te zetten op de **`return`** `0`-instructie.

```
#include <msp430.h>

int som(int a[], int n)
{
    // vul hier de benodigde code in
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    int rij[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int somrij = som(rij, sizeof rij / sizeof rij[0]);
    return 0;
}
```

Listing 4: Code bij opdracht 7.2.5, zie opdr7.2.5.c

[4] Een adres wordt door de MSP430 compiler opgeslagen in 2 bytes. De parameter array a bevat het beginadres van een array en is dus altijd 2 bytes groot.

In C kun je niet alleen met gehele getallen, maar ook met reële getallen werken. Een reëel getal wordt in de programmeertaal C opgeslagen als een zwevendekommagetal (Engels: floating-point number). Je kunt kiezen tussen twee verschillende datatypes **`float`** en **`double`**. Een variabele van het type **`float`** wordt opgeslagen in 32 bits (4 bytes) en heeft ongeveer 7 significante cijfers. Een variabele van het type **`double`** wordt opgeslagen in 64 bits (8 bytes) en heeft ongeveer 15 significante cijfers. In programma's die op een pc of telefoon moeten draaien, wordt bijna altijd het datatype **`double`** gebruikt omdat de processoren die daar gebruikt worden beschikken over speciale hardware waardoor ze net zo snel met variabelen van het type **`double`** kunnen rekenen als met variabelen van het type **`float`**. Ook aan RAM-geheugen heb je op een pc of telefoon meestal geen gebrek. In programma's die op een kleine microcontroller zoals de MSP430 moeten draaien, wordt bijna altijd het datatype **`float`** gebruikt omdat de processoren in deze microcontrollers niet beschikken over speciale hardware om floating-point berekeningen uit te kunnen voeren. Deze berekeningen worden uitgevoerd in librarycode waarbij een floating-point berekening wordt geëmuleerd met integer berekeningen. Ook het beschikbare RAM-geheugen is op een kleine microcontroller vaak erg beperkt. Zo heeft de MSP430G2553 maar 512 bytes RAM.

7.2.6 In deze opdracht ga je werken met een array van floating-point getallen. Gebruik de volgende array om de functies mee te testen:

```
float cijfers[] = {3.2, 5.7, 8.5, 9.1};
```

A Bij opdracht 3.1.1 heb je in Python een functie genaamd `avg` geschreven die het gemiddelde van een rij getallen berekent en teruggeeft. Schrijf nu in C een functie genaamd `avg` die het gemiddelde van een rij getallen berekent en teruggeeft. Deze rij getallen wordt als argument, via een C `array`, doorgegeven aan de functie. Het aantal elementen waaruit de rij bestaat wordt als tweede argument meegeven aan de functie. Het juiste resultaat is `6.625`.

B Bij opdracht 3.1.2 heb je in Python een functie genaamd `std_dev` geschreven die standaardafwijking van een rij getallen berekent en teruggeeft. Hoe je de standaardafwijking aangeduid met σ toen hebt berekend is gegeven in vergelijking (1). Hierin is N het aantal getallen in de rij, x_i het i^{de} getal uit de rij en μ het gemiddelde van de rij.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2} \quad \text{met} \quad \mu = \frac{1}{N}\sum_{i=1}^{N}x_i \tag{1}$$

Schrijf nu in C een functie genaamd `std_dev` die de standaardafwijking van een rij getallen berekent en teruggeeft. Deze rij getallen wordt als argument, via een C `array`, doorgegeven aan de functie. Het aantal elementen waaruit de rij bestaat wordt als tweede argument meegeven aan de functie. In C kun je de wortel berekenen met de functie `sqrt` die is gedefinieerd in de standaard include-file `math.h`[5]. Als het goed is, geeft deze functie uitgevoerd op de MSP430 de waarde `2.35730267` terug. De Python functie gaf als antwoord `2.3573024837725005`. Verklaar dit verschil!

C Schrijf een functie genaamd `max` die de maximale waarde van een rij getallen bepaald en teruggeeft. In figuur 1 is een mogelijke implementatie van deze functie in een flowchart weergegeven. De Flowgorithm code kun je hier vinden: `max.fprg`.

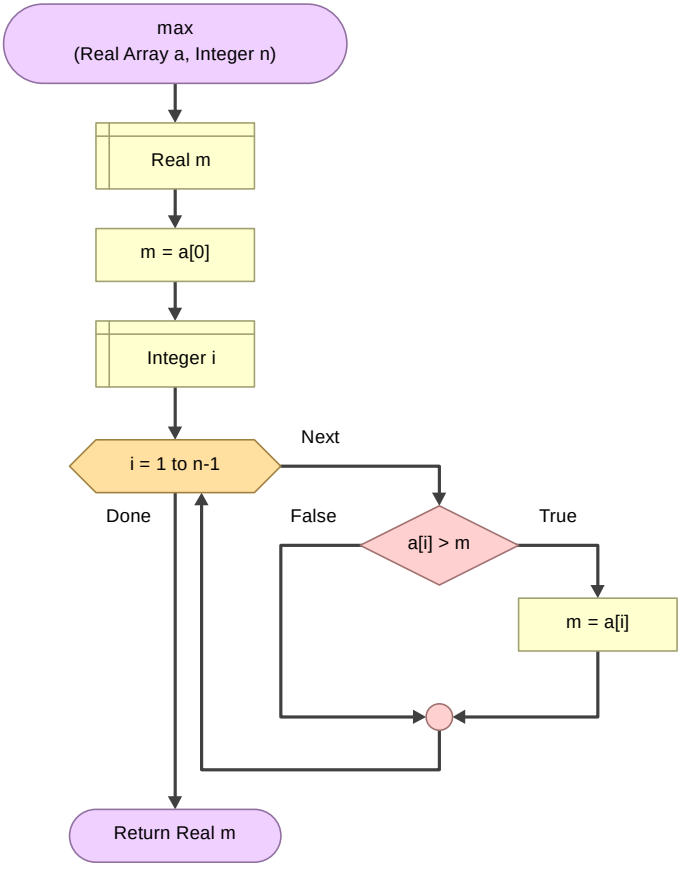
Je gaat nu verder met het programma dat je voor opdracht 7.1.12 hebt gemaakt. Bij het testen van dit programma bleek de gemeten waarde van de temperatuur nogal te schommelen, zie figuur 2. Je gaat nu een softwarematig filter toevoegen aan dit programma door middel van het berekenen van het voortschrijdend gemiddelde[6].

7.2.7 Breid het programma dat je hebt geschreven bij opdracht 7.1.12 nu uit met een berekening van het voortschrijdend gemiddelde van de laatste tien berekende temperaturen. Als het programma start, zijn er nog geen tien temperaturen gemeten en moet het gemiddelde over de beschikbare temperaturen worden berekend. De implementatie van de benodigde berekening is gegeven in figuur 3. De variabelen `temps` en `aantal` moeten **`static`** gedefinieerd worden omdat hun waarde tussen de interrupts door bewaard moet blijven. De variabelen `temp` en `gem` moeten **`static`** gedefinieerd worden zodat hun waarde geplot kan worden in CCS. Zorg ervoor dat de waarde van de variabelen `temp` en `gem` geplot worden in CCS door twee breakpoints te definiëren in de ISR zoals je bij opdracht 7.1.12 hebt geleerd.

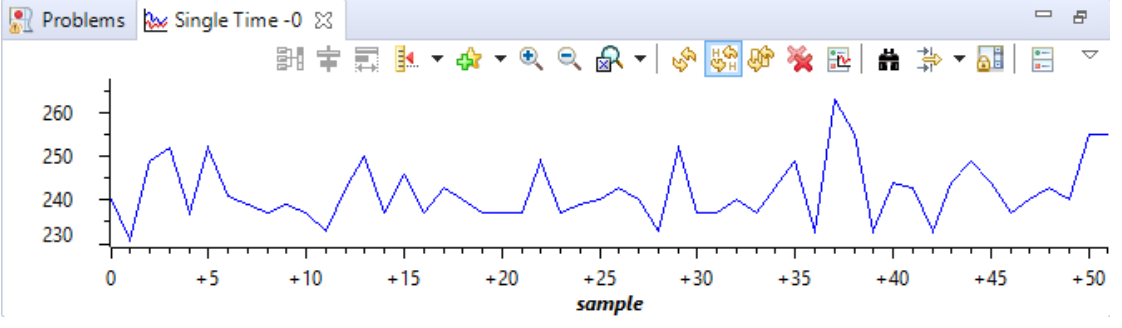
In figuur 4 is een mogelijke uitvoer van het programma van opdracht 7.2.7 gegeven. In de bovenste grafiek is de gemeten temperatuur weergegeven en in de onderste grafiek is het voortschrijdend gemiddelde van de laatste tien metingen weergegeven. Let op het veschil in

[5] Zie eventueel: `http://en.cppreference.com/w/c/numeric/math`.

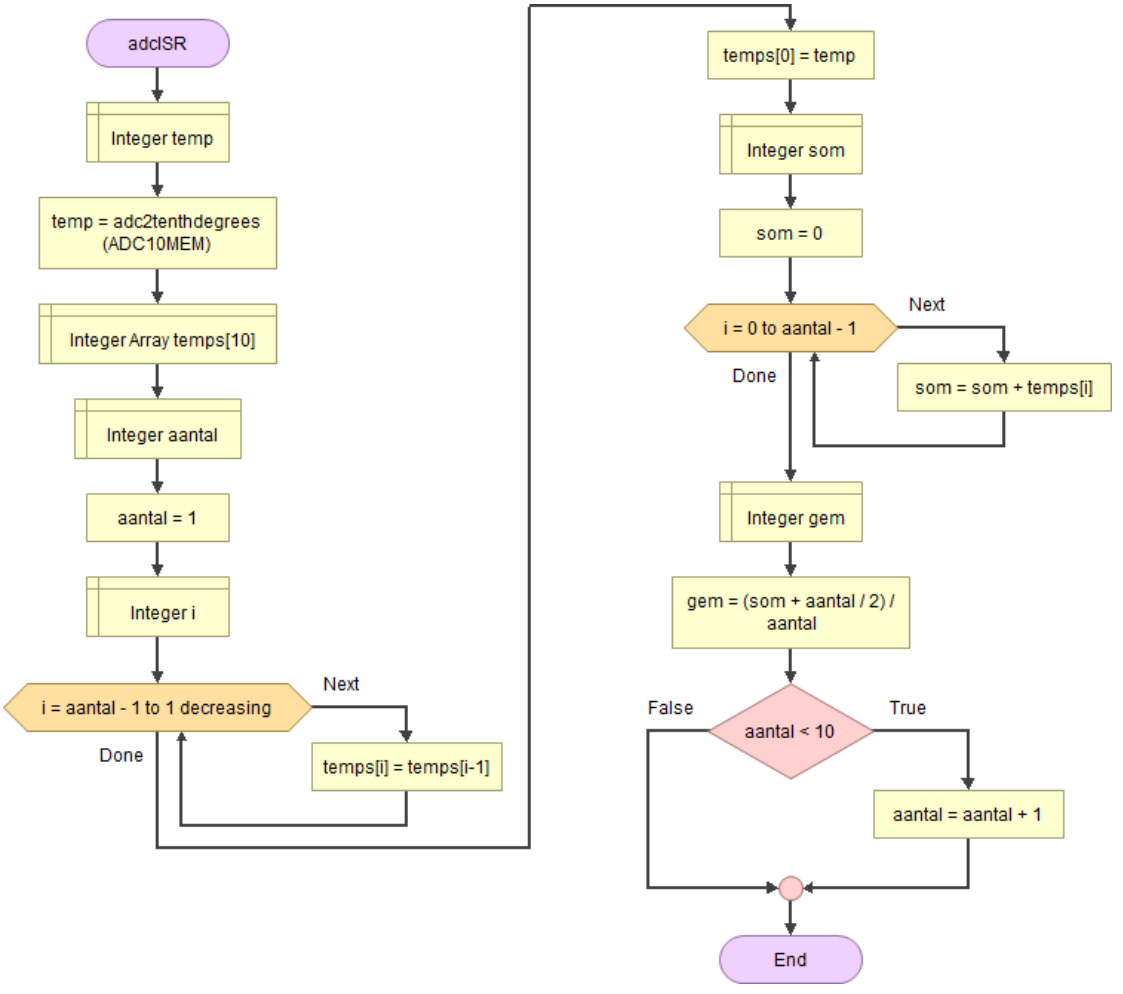
[6] Zie eventueel `https://nl.wikipedia.org/wiki/Voortschrijdend_gemiddelde`.



Figuur 1: Een algoritme om de maximale waarde in een rij te bepalen.



Figuur 2: Plotten van de temperatuur in CCS.

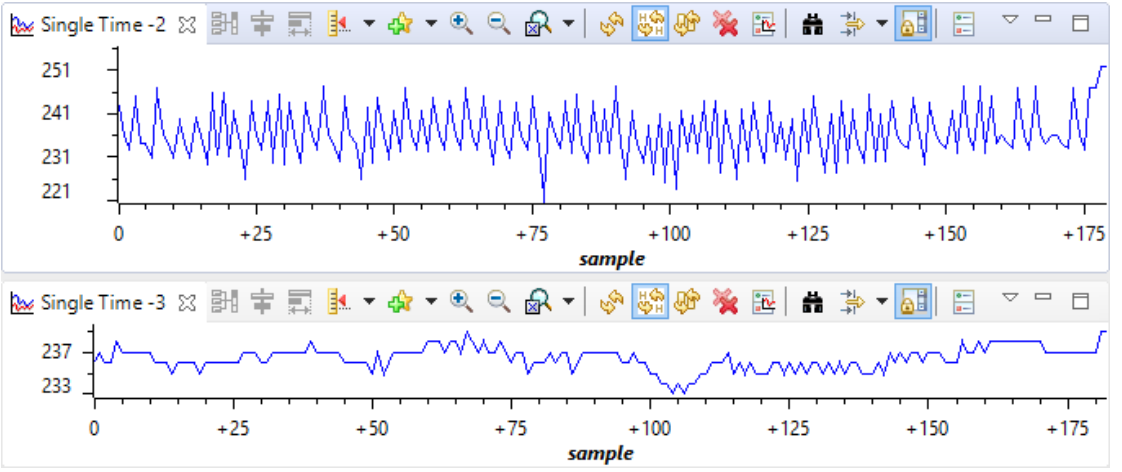


Figuur 3: Het algoritme om het voortschrijdend gemiddelde te berekenen.

de schaalverdeling op de y-as. Bij de gemeten temperatuur varieert de waarde tussen 22,1 °C en 25,1 °C. Bij de gemiddelde temperatuur varieert de waarde tussen 23,3 °C en 23,9 °C.

Oefening

Je hebt deze les geleerd hoe je arrays en functies in C kunt gebruiken. Het is verstandig om een en ander uit je hoofd te leren. Flashcards kunnen je daarbij helpen. Er is een stok Anki flashcards beschikbaar zodat je ze regelmatig kunt herhalen. Deze stok kun je hier downloaden: EMS10 Week 7 Les 2.apkg.



Figuur 4: Mogelijke uitvoer van het programma van opdracht 7.2.7.

Verdieping

7.2.8 Je gaat bij deze opdracht verder met het project dat je bij opdracht 7.1.16 hebt gecommit. Breid dit programma uit met een berekening van het voortschrijdend gemiddelde van de laatste tien gemeten temperaturen. Print de waarde van dit voortschrijdend gemiddelde op het oleddisplay. De geprinte waarde van de temperatuur zal nu minder sterk schommelen. Wat gebeurt er als je het voortschrijdend gemiddelde over de laatste vier gemeten temperaturen berekent in plaats van over de laatste tien?