Opdrachten week 1 lab 2 – Pointers in C

In de vorige les heb je kennis gemaakt met de SimpleLink™ Wi-Fi® CC3220S LaunchPad™. Je kennis van de programmeertaal C heb je opgefrist. Deze les gaan we die kennis uitbreiden.

Je gaat deze les leren hoe je:

- het adres van de ene variabele kan opslaan in een andere variabele. Deze laatste variabele wordt dan een pointer genoemd;
- pointers kunt gebruiken om een C-functie meerdere waarden terug te laten geven;
- pointers kunt gebruiken om over een C-array of C-string te itereren;
- de executietijd van een stukje C-code kunt bepalen;
- de executietijd van een stukje C-code kunt verkorten door de optimalisatie-instellingen van de compiler te wijzigen.

In het theoriedeel van de les is uitgelegd hoe je pointers in C kunt definiëren en gebruiken.

1.2.1 Een niet al te snuggere programmeur heeft een functie geschreven om twee getallen van laag naar hoog te sorteren. Deze functie met het bijbehorende testprogramma is gegeven in listing 1

```
#include <stdio.h>

void sorteer(int a, int b)
{
    if (a > b)
    {
        int hulpje = a;
        a = b;
        b = hulpje;
    }
}

int main(void)
{
    int x1 = 7, x2 = 8;
    sorteer(x1, x2);
    if (x1 == 7 && x2 == 8)
    {
        printf("Test 1 geslaagd!\n");
```

```
    }
    else
    {
        printf("Test 1 NIET geslaagd!\n");
    }
    int x3 = 8, x4 = 7;
    sorteer(x3, x4);
    if (x3 == 7 && x4 == 8)
    {
        printf("Test 2 geslaagd!\n");
    }
    else
    {
        printf("Test 2 NIET geslaagd!\n");
    }
    return 0;
}
```

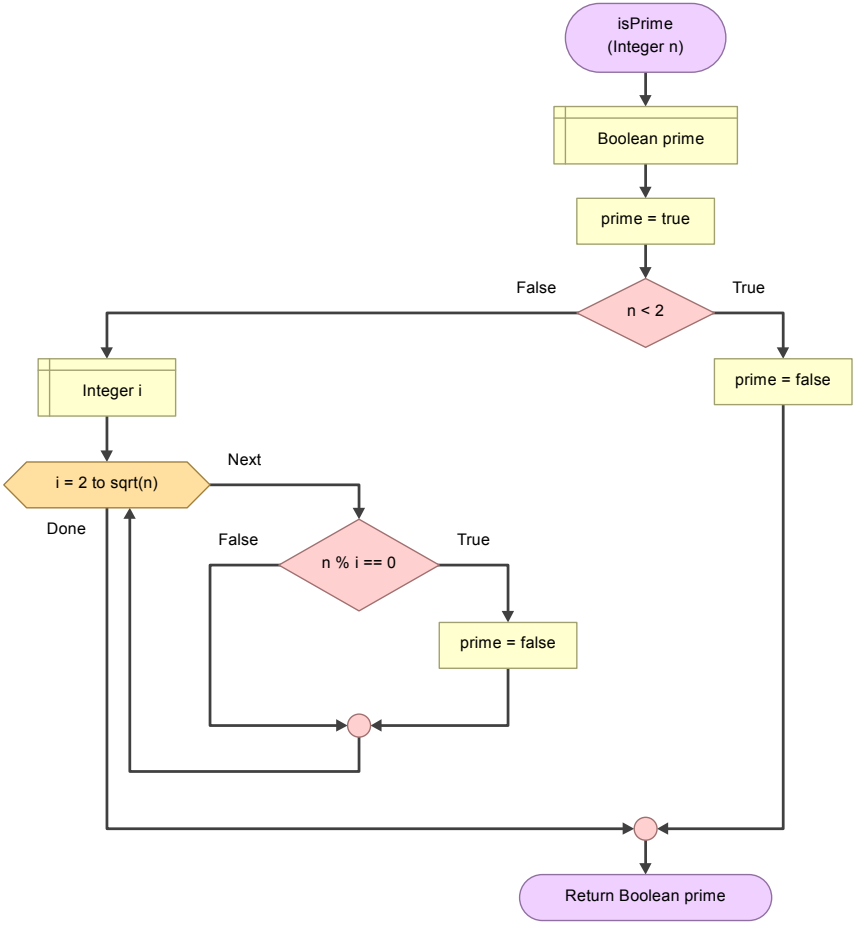
Listing 1: Functie `sorteer`, zie `sorteer.c`

A Voer het programma gegeven in listing 1 uit op jouw LaunchPad. Je ziet dat Test 2 niet geslaagd is. Verklaar dit!

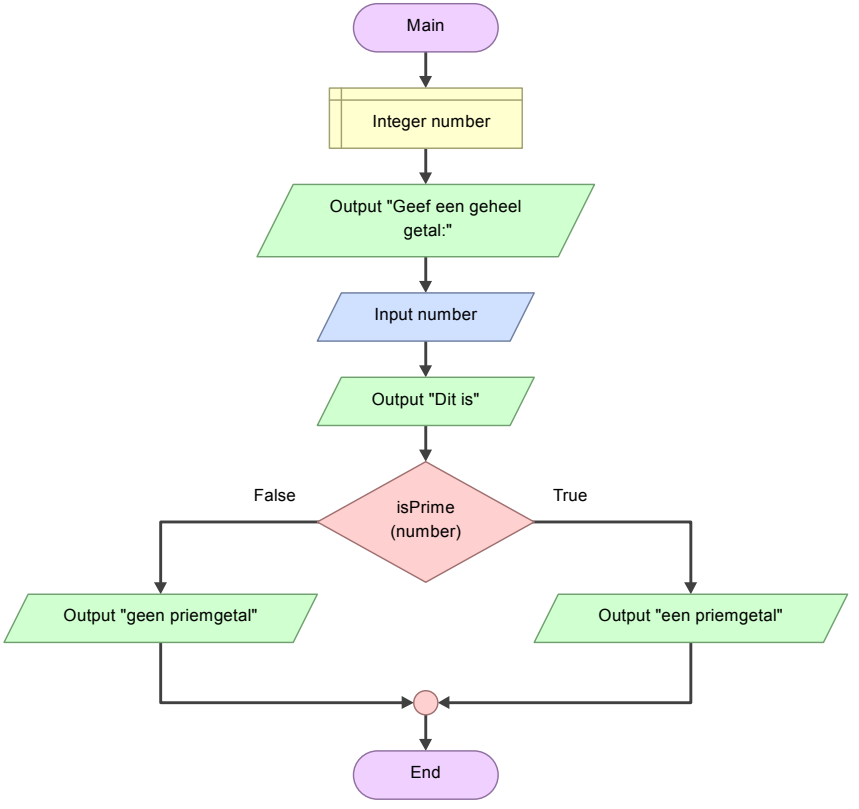
B Pas de functie en het testprogramma aan zodat het sorteren van de getallen wel goed gaat[1].

1.2.2 Gegeven is een functie `isPrime` die bepaalt of een als input argument meegegeven integer een priemgetal is of niet. Het resultaat (een Booleaanse waarde) wordt teruggegeven via de returnwaarde van de functie. De flowchart van deze functie is afgebeeld in figuur 1. Het bijbehorende test programma is afgebeeld in figuur 2. Het bijbehorende C-programma kun je vinden in listing 2.

[1] Tip: vervang de parameters a en b door pointers.



Figuur 1: De flowchart van de functie `isPrime`.



Figuur 2: De flowchart van het testprogramma voor de functie `isPrime`.

```c
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

bool isPrime(int n)
{
    if (n < 2)
    {
        return false;
    }
    int i;
    for (i = 2; i <= sqrt(n); i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}

int main(void)
{
    int number;
    printf("Geef een geheel number: ");
    scanf("%d", &number);
    printf("Het number %d is ", number);
    if (isPrime(number))
    {
        printf("een ");
    }
    else
    {
        printf("geen ");
    }
    printf("priemgetal\n");
    while (1);
    return 0;
}
```

Listing 2: Functie isPrime, zie isPrime.c

Schrijf nu een functie met een input parameter[2] `n` (een geheel getal groter dan 2) en twee output parameters[3]: het kleinste priemgetal groter dan `n` en het grootste priemgetal kleiner dan `n`. Schrijf ook een programma om de functie te testen.

Je kunt een pointer ook naar een element uit een array laten wijzen. Sommige C-programmeurs werken liever met pointers dan met indexen als ze elementen uit een array moeten bewerken. In het verleden leverde het gebruik van pointers in plaats van indexen veel snellere machinecode op, maar de C-compilers zijn tegenwoordig zo goed dat het niet veel meer uitmaakt. In listing 3 is een programma gegeven waarin twee functies gegeven zijn die beide het aantal karakters in een string bepalen[4]. De ene functie maakt gebruik van een indexvariabele en de andere gebruikt een pointer. De standaard functie `strlen` wordt gebruikt om de resultaten te verifiëren.

```
#include <stdio.h>
#include <string.h>

size_t strlen_met_index(char s[])
{
    size_t i = 0;

    while (s[i] != '\0')
    {
        i++;
    }
    return i;
}

size_t strlen_met_pointer(char *s)
{
    char *p = s;
```

[2] Een input parameter wordt gebruikt om een waarde als argument door te geven aan een functie. Een 'normale' parameter in C is dus een input parameter.

[3] Een output parameter wordt gebruikt om een waarde terug te geven vanuit een functie. In C kun je daarvoor de returnwaarde gebruiken, maar dan kun je slechts één waarde teruggeven. Als je meerdere waarden terug wilt geven dan kun je gebruik maken van pointers als parameters.

[4] Het aantal karakters in een string is altijd positief. Daarom is voor het returntype van de functies gekozen voor het type `size_t`. Een variabele van dit type kan in C gebruikt worden om de index of de lengte van een array in op te slaan, zie http://en.cppreference.com/w/c/types/size_t.

```
    while (*p != '\0')
    {
        p++;
    }
    return p - s;
}

int main(void)
{
    char* test = "Hoe lang is deze string?";
    size_t len1 = strlen_met_index(test);
    size_t len2 = strlen_met_pointer(test);
    size_t len3 = strlen(test);
    if (len1 != len3)
    {
        printf("str_len_met_index() werkt niet goed!\n");
    }
    else
    {
        printf("str_len_met_index() werkt naar verwachting\n");
    }
    if (len2 != len3)
    {
        printf("str_len_met_pointer() werkt niet goed!\n");
    }
    else
    {
        printf("str_len_met_pointer() werkt naar verwachting\n");
    }

    while (1);
    return 0;
}
```

Listing 3: Twee verschillende implementaties van de functie `strlen`, zie `pointerVersusIndex.c`

1.2.3 Bestudeer het programma dat gegeven is in listing 3. Zorg dat je het volledig begrijpt. Vraag indien nodig je practicumdocent om uitleg. Maak vervolgens een project aan zodat je het programma dat gegeven is in listing 3 kunt uitvoeren op de CC3220S LaunchPad. Bepaal nu de executietijd van de verschillende implementaties van de

functie `strlen`. Je kunt eenvoudig het aantal klokpulsen bepalen dat nodig is om een bepaald stukje code uit te voeren door nadat je de debugger hebt gestart in het menu Run › Clock vinkjes te zetten bij Enable en Show, zie figuur 3. Rechtsonder in CCS verschijnt nu een klein klokje waarbij het aantal klokperioden die de processor heeft uitgevoerd wordt weergegeven, zie figuur 4. Als je door je programma stapt zie je hoeveel klokperioden de processor nodig heeft om de code uit te voeren. Je kunt de Clock weer resetten door dubbel te klikken op het klokje. De klokfrequentie van de ARM Cortex-M4 waarop je programma draait (intern in de CC3220S) is 80 MHz. Vul nu tabel 1 in. Welke conclusies trek je hieruit?

Figuur 3: Je kunt de Clock aanzetten via het menu Run › Clock.

Figuur 4: Het aantal klokperioden dat de processor heeft uitgevoerd wordt rechtsonder in CCS, achter het klokje, weergegeven.

Tabel 1: De executietijd van de verschillende implementaties van de functie `strlen`.

Implementatie	Aantal klokperioden	Executietijd in µs.
`strlen_met_index`		
`strlen_met_pointer`		
`strlen`		

1.2.4 De vergelijking die je bij opdracht 1.2.3 hebt gemaakt is niet helemaal eerlijk. Je hebt de compiler namelijk nog niet verteld dat je machinecode wilt die zo snel mogelijk

uitgevoerd wordt. Dit kun je doen door de zogenoemde optimizer aan te zetten. Open het menu Project > Properties. Kies vervolgens voor Build > ARM Compiler > Optimization en zet het optimization level op 2 – Global Optimizations[5]. Zet de Speed vs. size trade-offs op 5 (speed). Compileer het programma opnieuw en stap door het programma om de executietijd van de verschillende functies te bepalen. Je ziet dat dit nog niet zo gemakkelijk is omdat de compiler hele delen van de code heeft weggeoptimaliseerd. Het is eigenlijk alleen nog maar mogelijk om vast te stellen hoeveel klokperioden elke functie duurt door de code op machinecodeniveau te debuggen. De docent heeft dit voor je gedaan, zie tabel 2. Volgende week leer je zelf hoe je dit kunt doen. Welke conclusies trek je uit de meetresultaten die weergegeven zijn in tabel 2?

Tabel 2: De executietijd van de verschillende implementaties van de functie `strlen` gecompileerd met optimization level 2.

Implementatie	Aantal klokperioden	Executietijd in µs.
`strlen_met_index`	136	1,7000
`strlen_met_pointer`	117	1,4625
`strlen`	0	0,0000

Hoe kan het dat het uitvoeren van de standaard functie `strlen` geen enkele tijd meer kost? Hoe kan de processor de lengte van de string `test` bepalen zonder ook maar één instructie uit te voeren?[6]

1.2.5 Zet het optimization level van het project weer terug op off, zodat je het programma weer beter stap voor stap kunt debuggen. In de praktijk zet je de optimizer altijd uit, als je een programma aan het ontwikkelen bent. Pas als je het programma aflevert (de zogenoemde Release versie), zet je de optimizer aan.

[5] Als je het optimization level nog hoger zet worden de functieaanroepen en returns helemaal weggeoptimaliseerd. De code van de verschillende functies wordt dan door elkaar gehusseld en het is niet meer mogelijk om te bepalen hoe lang elke functie duurt.

[6] De compiler weet wat de standaard functie `strlen` doet. Omdat de string `test` een constante tekst bevat, kan de compiler zelf al uitrekenen wat de waarde van de variabele `len3` zal worden (24). Bij het vertalen van het programma vervangt de compiler de variabele `len3` door het getal 24. Dat is ook de reden dat de debugger niet stopt bij het uitvoeren van de regel `size_t len3 = strlen(test);`. Deze regel komt namelijk helemaal niet meer voor in de machinecode die door de compiler is aangemaakt.

1.2.6 In listing 4 zijn twee functies gegeven die de inhoud van een C-string omkeren. De ene functie maakt gebruik van indexen en is al geïmplementeerd. De andere functie moet gebruik maken van pointers en moet nog geïmplementeerd worden. Maak een project aan zodat je de code die gegeven is in listing 4 kunt uitvoeren op de CC3220S LaunchPad. Implementeer ook de functie `keerom_met_pointers` en verzeker je ervan dat deze functie correct werkt.

```
#include <stdio.h>
#include <string.h>

void keerom_met_indexen(char *s)
{
    size_t eerste = 0;
    size_t laatste = strlen(s) - 1;

    while (laatste > eerste)
    {
        char hulp = s[eerste];
        s[eerste] = s[laatste];
        s[laatste] = hulp;
        eerste++;
        laatste--;
    }
}

void keerom_met_pointers(char *s)
{
    char *p_eerste = s;
    char *p_laatste = s + strlen(s) - 1;

    // vul hier je code in
}

int main(void)
{
    char test1[] = "Keer dit eens om";
    printf("\"%s\" omgekeerd is ", test1);
    keerom_met_indexen(test1);
    printf("\"%s\"\n", test1);

    char test2[] = "Keer dit eens om";
```

```
    printf("\"%s\" omgekeerd is ", test2);
    keerom_met_pointers(test2);
    printf("\"%s\"\n", test2);

    while (1);
    return 0;
}
```

Listing 4: Twee verschillende implementaties van de functie keerom, zie keerom.c