Opdrachten week 2 lab 1 – Structs en enums in C

Vorige week heb je kennis gemaakt met de SimpleLink™ Wi-Fi® CC3220S LaunchPad™. Je kennis van de programmeertaal C heb je opgefrist en uitgebreid zodat je nu ook pointers kunt gebruiken in je C-programma's. Deze les gaan we je C-kennis nog verder uitbreiden.

Je gaat deze les leren hoe je:

- meerdere variabelen van verschillende typen bij elkaar kunt opslaan in één samengestelde variabele. Zo'n samengestelde variabele wordt in C een **`struct`** genoemd;
- **`struct`**s kunt opslaan in een array;
- **`struct`**s kunt doorgeven als argument aan een functie en kunt teruggeven als returnwaarde vanuit een functie;
- een zogenoemd enumeratie-type kunt definiëren en gebruiken om je programma's beter leesbaar te maken;
- de compiler kunt vertellen welke versie van de C-standaard je wilt gebruiken.

In het theoriedeel van de les is uitgelegd hoe je **`struct`**s en **`enum`**s in C kunt definiëren en gebruiken.

2.1.1 De verschillende variabelen die worden samengenomen in een **`struct`** kunnen van verschillende types zijn. Maar soms is het ook handig om meerdere variabelen van hetzelfde type samen te nemen in een **`struct`**. In listing 1 is een onvolledig programma gegeven waarin het type `Breuk` is gedefinieerd. Een breuk bestaat, zoals je weet, uit twee gehele getallen een `teller` en een `noemer`. Er is een functie `product` gedefinieerd waarmee twee breuken vermenigvuldigd kunnen worden. Aan het einde van deze functie wordt de functie `normaliseer` aangeroepen om het resultaat te normaliseren. Dit wordt gedaan om de teller en de noemer zo klein mogelijk te houden en zodoende overflow te voorkomen. De breuk wordt genormaliseerd door de teller en de noemer beide te delen door de grootste gemene deler van de teller en de noemer. Tevens wordt er bij het normaliseren voor gezorgd dat de noemer altijd positief is. De functie ggd berekent en retourneert de grootste gemene deler van de twee argumenten die bij aanroep aan de functie worden doorgegeven. De functie is een recursieve[1] implementatie van het algoritme van Euclides[2]. Maak een project aan zodat je dit programma kunt

[1] Als je niet meer weet wat een recursieve functie is, bekijk dan nogmaals opdrachten 2.1.8 t/m 2.1.13 van EMS10.

[2] Zie eventueel: https://nl.wikipedia.org/wiki/Algoritme_van_Euclides

uitvoeren op de CC3220S LaunchPad. Schrik niet als het programma niet compileert. De functie `som` ontbreekt namelijk nog. Implementeer een functie `som` die de som van twee breuken berekent, normaliseert en retourneert. Als de functie `som` correct is geïmplementeerd, zal de uitvoer eruit zien zoals gegeven in figuur 1.

```
#include <stdio.h>

typedef struct {
    int teller;
    int noemer;
} Breuk;

int ggd(int n, int m)
{
    if (m == 0)
    {
        return n;
    }
    else
    {
        return ggd(m, n % m);
    }
}

Breuk normaliseer(Breuk b)
{
    int d = ggd(b.teller, b.noemer);
    b.teller /= d;
    b.noemer /= d;
    if (b.noemer < 0)
    {
        b.noemer = -b.noemer;
        b.teller = -b.teller;
    }
    return b;
}

Breuk product(Breuk b1, Breuk b2)
{
    Breuk p;
    p.teller = b1.teller * b2.teller;
```

```
    p.noemer = b1.noemer * b2.noemer;
    return normaliseer(p);
}

int main(void)
{
    Breuk a = {-7, 14}, b = {2, -3};

    Breuk c = som(a, b);
    printf("c = %d/%d\n", c.teller, c.noemer);

    Breuk d = product(a, b);
    printf("d = %d/%d\n", d.teller, d.noemer);

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

Listing 1: Programma waarin met breuken wordt gerekend, zie breuk.c

```
[Cortex_M4_0] c = -7/6
d = 1/3
```

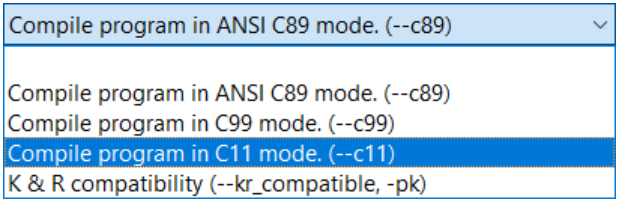
Figuur 1: Correcte uitvoer van het programma van opdracht 2.1.1.

2.1.2 In een (retro) arcadespel[3] willen we de behaalde scores bijhouden. Nadat het spel is afgelopen krijgt de gebruiker de gelegenheid om zijn naam in te voeren. Het spel houdt een lijst bij met de namen en het aantal behaalde punten van de laatste 3 spelers. Een programma dat dit doet is gegeven in listing 2. Maak een project aan zodat je dit programma kunt uitvoeren op de CC3220S LaunchPad. Schrik niet als het programma niet compileert. In enkele **for**-loops in het programma wordt de loop-variabele die alleen lokaal in de loop nodig is, in het **for**-statement gedefinieerd. Dit is niet toegestaan in de (verouderde) C89-standaard maar wel sinds de C99-standaard[4]. Open het menu Project › Properties en kies vervolgen Build › ARM Compiler › Advanced Options › Language Options. Kies vervolgens, in de dropdown box *C Dialect*,

[3] Zie eventueel: https://nl.wikipedia.org/wiki/Arcadespel.

[4] Er zijn verschillende versies van de C-standaard C89, C99, C11 en C18. De laatste versie C18 bevat geen nieuwe features en wordt niet ondersteund door de TI C-compiler. We adviseren je om C11 te gebruiken.

voor *Compile program in C11 mode (–c11)*, zie figuur 2. Als het goed is compileert het programma nu wel zonder fouten.



Figuur 2: Vertel de compiler dat je C11 wilt gebruiken.

In de functie `speelSpel` wordt het spelen van het spel gesimuleerd door een random score te genereren. Op deze wijze kunnen we de code voor het bijhouden van de scores testen. Bestudeer het programma en zorg dat je het volledig begrijpt. Beantwoord daarbij de volgende vragen:

A In de functie `maakLijstLeeg` staat de regel: `lijst[i].naam[ci] = '\0';`. Wat gebeurt er als deze regel uitgevoerd wordt en waarom is dit noodzakelijk om het programma correct te laten werken?

B In de functie `printLijst` wordt de functie `printf` aangeroepen met als eerste argument `"%-*s %5d\n"`. Zoek met behulp van het internet[5] uit wat dit betekent.

C De functie `leesNaam` leest een door de gebruiker ingetypte naam in, in de variabele `s`. Hierbij wordt gebruik gemaakt van twee standaard functies die je misschien nog niet kent: `getchar` en `isprint`. Zoek met behulp van het internet uit wat deze functies doen. Wat gebeurt er als je een hele lange naam intypt? Verklaar waarom dit gebeurt. Wat gebeurt er als je de TAB-toets gebruikt bij het intypen van een naam? Verklaar waarom dit gebeurt.

D De eerste regel van de functie `main` is `srand(time(NULL));`. Zoek met behulp van het internet uit wat de functies `srand` en `time` doen. Hoe verandert het gedrag van het programma als deze regel uit het programma wordt verwijderd?

Vraag indien nodig je practicumdocent om uitleg.

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

[5] Uitgebreide informatie betreffende `printf` kun je vinden op:

- `http://en.cppreference.com/w/c/io/fprintf`;
- `https://en.wikipedia.org/wiki/Printf_format_string`.

```c
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#define AANTAL_SPELERS_IN_LIJST 3
#define MAX_AANTAL_KARAKTERS_IN_NAAM 15

typedef struct {
    char naam[MAX_AANTAL_KARAKTERS_IN_NAAM + 1];
    unsigned int punten;
} Speler;

void maakLijstLeeg(Speler lijst[], size_t aantalInLijst)
{
    for (size_t i = 0; i < aantalInLijst; i++)
    {
        size_t ci;
        for (ci = 0; ci < MAX_AANTAL_KARAKTERS_IN_NAAM; ci++)
        {
            lijst[i].naam[ci] = '-';
        }
        lijst[i].naam[ci] = '\0';
        lijst[i].punten = 0;
    }
}

void printLijst(Speler lijst[], size_t aantalInLijst)
{
    for (size_t i = 0; i < aantalInLijst; i++)
    {
        printf("%-*s %5d\n", MAX_AANTAL_KARAKTERS_IN_NAAM, ↩
 ↪ lijst[i].naam, lijst[i].punten);
    }
}

void zetBovenaanInLijst(Speler s, Speler lijst[], size_t ↩
 ↪ aantalInLijst)
{
    // Alle voorgaande spelers een plaatsje opschuiven:
    for (size_t i = aantalInLijst - 1; i > 0; i--)
```

```
    {
        lijst[i] = lijst[i - 1];
    }
    // De laatste speler komt bovenaan de lijst:
    lijst[0] = s;
}

int speelSpel(void)
{
    // Hier komt  de code die het spel bestuurt.
    // Het aantal behaalde punten wordt als returnwaarde ↩
 ↪ teruggegeven.
    return rand();
}

void leesNaam(char s[])
{
    char karakter;
    size_t i = 0;
    do
    {
        karakter = getchar();
        if (isprint(karakter) && i != ↩
 ↪ MAX_AANTAL_KARAKTERS_IN_NAAM)
        {
            s[i] = karakter;
            i++;
        }
    }
    while (karakter != '\n');
    s[i] = '\0';
}

int main(void)
{
    srand(time(NULL));
    Speler laatsteSpelers[AANTAL_SPELERS_IN_LIJST];
    maakLijstLeeg(laatsteSpelers, AANTAL_SPELERS_IN_LIJST);

    while (1)
    {
```

```
        Speler s;
        s.punten = speelSpel();
        printf("Je hebt %d punten behaald, type nu je naam in: ↩
↪ ", s.punten);
        leesNaam(s.naam);
        zetBovenaanInLijst(s, laatsteSpelers, ↩
↪ AANTAL_SPELERS_IN_LIJST);
        printf("Laatste %d spelers:\n", ↩
↪ AANTAL_SPELERS_IN_LIJST);
        printLijst(laatsteSpelers, AANTAL_SPELERS_IN_LIJST);
    }
    return 0;
}
```

Listing 2: Programma waarin de scores worden bijgehouden, zie high_score.c

2.1.3 Breid nu het programma gegeven in listing 2 uit zodat ook een lijst met high scores wordt bijgehouden (van hoog naar laag geordend). Als er een aantal spelletjes zijn gespeeld, kan de uitvoer eruit zien zoals gegeven in figuur 3.

```
Je hebt 12856 punten behaald, type nu je naam in: Willem-Allexander
Laatste 3 spelers:
Willem-Allexand 12856
MarioBro        12077
Maxima           1890
Beste 5 spelers:
Anne            24311
Willem-Allexand 12856
MarioBro        12077
Theo            10128
Harry            4655
Je hebt 2433 punten behaald, type nu je naam in:
```

Figuur 3: Mogelijke uitvoer van het programma van opdracht 2.1.3.

2.1.4 In listing 3 is een programma gegeven waarin een **struct**-type is gedefinieerd waarin de gegevens van een passieve elektrische component opgeslagen kunnen worden. Dit zou bijvoorbeeld gebruikt kunnen worden om een stuklijst te produceren.

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

```c
typedef struct {
    int type;
    int index;
    double waarde;
} PassieveComponent;

void printComponent(PassieveComponent c)
{
    switch (c.type)
    {
    case 1:
        printf("R%d = %.1E ohm", c.index, c.waarde);
        break;
    case 2:
        printf("C%d = %.1E farad", c.index, c.waarde);
        break;
    case 3:
        printf("L%d = %.1E henry", c.index, c.waarde);
        break;
    }
}

int main(void)
{
    PassieveComponent c[] = {
        {1, 1, 1E6}, {1, 2, 2.7E3}, {2, 1, 1E-9}, {3, 1, ↩
 ↪ 1E-3}, {2, 2, 15E-6}
    };

    printf("\n");
    for (size_t i = 0; i < sizeof c / sizeof c[0]; i++)
    {
        printComponent(c[i]);
        printf("\n");
    }

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

Listing 3: Programma waarin verschillende passieve elektrische componenten gedefinieerd worden, zie component.c

Het programma is echter niet zo goed leesbaar omdat het type van de component (weerstand, condensator of spoel) is gecodeerd met een integer. Het is niet in een oogopslag duidelijk welke componenten gedefinieerd worden op eerste regels van de functie `main`:

```
    PassieveComponent c[] = {
        {1, 1, 1E6}, {1, 2, 2.7E3}, {2, 1, 1E-9}, {3, 1, ↩
↪ 1E-3}, {2, 2, 15E-6}
    };
```

Maak het programma beter leesbaar door een enumeratie (**`enum`**) te gebruiken. De eerste regels van de functie `main` moeten dan als volgt worden aangepast:

```
    PassieveComponent c[] = {
        {R, 1, 1E6}, {R, 2, 2.7E3}, {C, 1, 1E-9}, {L, 1, ↩
↪ 1E-3}, {C, 2, 15E-6}
    };
```