Stap voor stap uitwerking van een eenvoudige C opdracht

Bij het vak CPL01 krijg je regelmatig een opdracht die als volgt begint: “schrijf een programma dat . . . ”. Hoe pak je zoiets, als beginnend programmeur, nu aan? Als voorbeeld neem ik de opdracht: “schrijf een programma dat een geheel getal $0 < n < 7$ inleest en vervolgens de tafels van vermenigvuldiging[1] van 1 t/m n naast elkaar afdrukt”.

Stap voor stap

De methode die ik gebruik om tot een werkend programma te komen dat aan de opdracht voldoet is stapsgewijze verfijning (Engels: stepwise refinement). Ik begin met een eenvoudig programma en breid het programma stap voor stap uit tot ik uiteindelijk het gewenste programma heb. Ik test het programma na elke stap. De bovenstaande opdracht zou ik als volgt aanpakken.

Stap 0: Bezint eer gij begint

De eerste stap heeft nog niets met programmeren te maken. Voordat ik begin te programmeren denk ik eerst over de opdracht na. Snap ik wat ik moet maken? Het helpt vaak als ik mogelijke testgevallen bedenk. Bijvoorbeeld: bij de invoer 4 moet het programma de uitvoer produceren die gegeven is in figuur 1.

Bij het zien van deze gewenste uitvoer realiseer ik mezelf dat het erg belangrijk is dat de regels van de tafels netjes onder elkaar worden afgedrukt. Ik herinner me dat je bij het afdrukken van een geheel getal met behulp van `printf` een veldbreedte kunt opgeven, bijvoorbeeld `printf("3d", getal);` zal de waarde van de variabele `getal` op een veld van 3 karakters breed afdrukken. Als getal de waarde 7 heeft, dan wordt dus afgedrukt `<spatie><spatie>7`.

[1] Als je niet weet wat de tafels van vermenigvuldiging zijn, kijk dan op Wikipedia.

```
 1 x 1 =  1   1 x 2 =  2   1 x 3 =  3   1 x 4 =  4
 2 x 1 =  2   2 x 2 =  4   2 x 3 =  6   2 x 4 =  8
 3 x 1 =  3   3 x 2 =  6   3 x 3 =  9   3 x 4 = 12
 4 x 1 =  4   4 x 2 =  8   4 x 3 = 12   4 x 4 = 16
 5 x 1 =  5   5 x 2 = 10   5 x 3 = 15   5 x 4 = 20
 6 x 1 =  6   6 x 2 = 12   6 x 3 = 18   6 x 4 = 24
 7 x 1 =  7   7 x 2 = 14   7 x 3 = 21   7 x 4 = 28
 8 x 1 =  8   8 x 2 = 16   8 x 3 = 24   8 x 4 = 32
 9 x 1 =  9   9 x 2 = 18   9 x 3 = 27   9 x 4 = 36
10 x 1 = 10  10 x 2 = 20  10 x 3 = 30  10 x 4 = 40
```

Figuur 1: Gewenste uitvoer bij de invoer 4.

Stap 1: Alle begin is ~~moeilijk~~ makkelijk

Elk C-programma heeft hetzelfde begin en einde. Ik heb een bestandje `c.c` gemaakt waarin een C-programma staat dat niets doet. Ik begin met een kopietje van dat bestand en noem dat `tafels_stap1.c`, zie listing 1.

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */

int main(void)
{

    /* Hier komt de code */

    return 0;
}
```

Listing 1: Stap 1: `tafels_stap1.c`.

Stap 2: Invoer

Het is altijd handig om met de invoer te beginnen. Ik begin dus met het eerste deel van de opdracht: “schrijf een programma dat een geheel getal $0 < n < 7$ inleest”. Ik herinner mij dat ik een geheel getal kan inlezen met behulp van `scanf`. Voordat

ik de waarde van n inlees wil ik de gebruiker eerst vertellen wat er van hem/haar verwacht wordt door op het scherm te zetten: `Geef de waarde van n (1..6):`, dat kan ik doen met behulp van `printf`. Om te kijken of het inlezen is gelukt druk ik de waarde van n af. Zie listing 2.

```
1  #include <stdio.h>
2
3  /* Copyright 2016 Hogeschool Rotterdam */
4  /* Dit programma leest een geheel getal 0 < n < 7 en drukt
5     vervolgens de tafels van 1 t/m n naast elkaar af */
6
7  int main(void)
8  {
9      int n;
10
11     printf("Geef de waarde van n (1..6): ");
12     scanf("%d", n);
13
14     printf("Test n = %d", n);
15
16     return 0;
17 }
```

Listing 2: Stap 2a: `tafels_stap2a.c`.

Na elke stap compileer en run ik het programma om te testen of de code die ik heb toegevoegd goed werkt. Het voordeel van deze manier van werken is dat ik fouten meteen ontdek en ze dan ook meteen kan corrigeren.

Als ik het programma uit listing 2 uitvoer verschijnt de mededeling die gegeven is in figuur 2 op mijn scherm. Jij ziet vast meteen wat er mis is, maar ik kon de fout niet zo snel vinden. Het is dan altijd een goed idee om de compiler warnings aan te zetten[2] en het programma opnieuw te compileren.

Als het programma uit listing 2 gecompileerd wordt nadat de warnings aangezet zijn, geeft de compiler twee waarschuwingen, zie figuur 3.

[2] Hoe je de compiler warnings in Code::Blocks kunt aanzetten kun je vinden in appendix A

Figuur 2: Melding die verschijnt na het uitvoeren van het programma uit listing 2.

```
warning line 12: format '\%d' expects argument of type ↩
 ↪ 'int *', but argument 2 has type 'int' [-Wformat=]
warning line 12: 'n' is used uninitialized in this ↩
 ↪ function [-Wuninitialized]
```

Figuur 3: Warnings bij het compileren van het programma uit listing 2.

Er is blijkbaar iets aan de hand met het tweede argument `n` van de `scanf`-functie. Nu zie ik dat ik het `&` teken ben vergeten, dat je altijd voor een variabele moet plaatsen die je met `scanf` wilt inlezen[3].

Het gecorrigeerde programma vind je in listing 3. Dit programma werkt zoals verwacht. Wel valt het me op dat ongeldige waarden voor `n` gewoon worden geaccepteerd door het programma.

Stap 3: Controle op invoer

Ik bedenk me dat het netjes is om te controleren of het ingevoerde getal binnen de gestelde grenzen ligt. Wat moet er gebeuren als de gebruiker een getal invoert dat buiten deze grenzen ligt? Ik besluit om de gebruiker opnieuw te vragen om de

[3] Ik kan nu nog niet uitleggen waarom dit nodig is, maar bij het behandelen van *pointers* kom ik er op terug.

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void)
{
    int n;

    printf("Geef de waarde van n (1..6): ");
    scanf("%d", &n);

    printf("Test n = %d", n);

    return 0;
}
```

Listing 3: Stap 2b: `tafels_stap2b.c`.

waarde in te voeren (zonder foutmelding) en de ingevoerde waarde opnieuw in te lezen. Ik wil dus het programmadeel dat gegeven is in listing 4 laten herhalen totdat een geldige waarde is ingevoerd.

```c
    printf("Geef de waarde van n (1..6): ");
    scanf("%d", n);
```

Listing 4: Programmadeel uit listing 3 dat ik wil laten herhalen.

Ik herinner me uit de les dat er verschillende herhalingsinstructies zijn (`while`, `do while` en `for`). Het bovenstaande programmadeel moet minstens één maal herhaald worden en het is afhankelijk van de gebruiker hoe vaak het herhaald moet worden. Om die reden moet ik (volgens de in les behandelde theorie) kiezen voor een `do while`-lus. Zie listing 5.

Bij het testen probeer ik eerst een aantal ongeldige waarden en constateer dat het programma correct werkt, zie figuur 4. De door mij ingevoerde waarden zijn groen en onderstreept weergegeven.

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void)
{
    int n;

    do
    {
        printf("Geef de waarde van n (1..6): ");
        scanf("%d", &n);
    }
    while(n < 1 || n > 6);

    printf("Test n = %d", n);

    return 0;
}
```

Listing 5: Stap 3: tafels_stap3.c.

```
Geef de waarde van n (1..6): -3
Geef de waarde van n (1..6): 0
Geef de waarde van n (1..6): 8000
Geef de waarde van n (1..6): 7
Geef de waarde van n (1..6): 4
Test n = 4
```

Figuur 4: Uitvoer van het programma uit listing 5.

Stap 4: Eerste regel van de tafels afdrukken

De waarde van n wordt nu dus succesvol ingelezen en op de juiste wijze gecontroleerd. Het tweede deel van de opdracht was: "... en vervolgens de tafels van 1 t/m n naast elkaar afdrukt". Het tweede deel van de opdracht is voor een beginnende programmeur misschien te moeilijk om in één stap uit te voeren. Ik bedenk dat het niet zo moeilijk is om de eerste regel van de tafels af te drukken. Dus bij de invoer 3 moet het programma de uitvoer produceren die gegeven is in figuur 5.

```
1 x 1 =  1   1 x 2 =  2   1 x 3 =  3
```

Figuur 5: Gewenste uitvoer bij de invoer 3 bij stap 4.

Bij de invoer 5 moet het programma de uitvoer produceren die gegeven is in figuur 6.

```
1 x 1 =  1   1 x 2 =  2   1 x 3 =  3   1 x 4 =  4   1 x 5 =  5
```

Figuur 6: Gewenste uitvoer bij de invoer 5 bij stap 4.

Het valt meteen op dat het stukje uitvoer `1 x i = i` wordt herhaald voor $i = 1$ tot en met $i = n$. Hierbij is i het nummer van de tafel. We moeten het programmadeel dat dit stukje print dus n maal herhalen. Het aantal herhalingen is dus bekend als de herhaling begint (want de waarde van n is dan al ingelezen). Omdat het aantal herhalingen bekend is moet ik (volgens de in les behandelde theorie) kiezen voor een `for`-lus. Zie listing 6.

Bij het compileren van het programma uit listing 6 geeft de compiler de error die gegeven is in figuur 7.

```
error line 18: 'for' loop initial declarations are only ↩
 ↪ allowed in C99 mode
```

Figuur 7: Error bij het compileren van het programma uit listing 6.

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void)
{
    int n;

    do
    {
        printf("Geef de waarde van n (1..6): ");
        scanf("%d", &n);
    }
    while(n < 1 || n > 6);

    for (int tafel = 1; tafel < n + 1; tafel = tafel + 1)
    {
        printf(" 1 x %d = %2d ", tafel, 1 * tafel);
    }
    printf("\n");

    return 0;
}
```

Listing 6: Stap 4: tafels_stap4.c.

We moeten blijkbaar nog tegen de compiler vertellen dat we een moderne versie van C willen gebruiken. Als we de compiler vertellen dat we C11 willen gebruiken[4] dan compileert het programma correct.

Bij het testen blijkt dat het programma ook correct werkt, zie figuur 8.

```
Geef de waarde van n (1..6): 4
 1 x 1 =  1  1 x 2 =  2  1 x 3 =  3  1 x 4 =  4
```

Figuur 8: Uitvoer van het programma uit listing 6.

Stap 5: Alle 10 regel van de tafels afdrukken

Als ik alle 10 regels van de tafels wil afdrukken dan moet ik het programmadeel waarin de eerste regel wordt afgedrukt tien maal herhalen. Het programmagedeelte dat herhaald moet worden is gegeven in listing 7.

```
18      for (int tafel = 1; tafel < n + 1; tafel = tafel + 1)
19      {
20          printf(" 1 x %d = %2d ", tafel, 1 * tafel);
21      }
22      printf("\n");
```

Listing 7: Programmadeel uit listing 6 dat ik wil laten herhalen.

De constante 1 die in de eerste regel wordt afgedrukt moet in de tweede regel vervangen worden door de constante 2, in de derde regel door de constante 3 enz. Ik maak een variabele `factor` aan die 'loopt' van 1 t/m 10. Omdat het aantal herhalingen bekend is (tien maal) gebruik ik een `for`-loop. Zie listing 8.

Bij het testen blijkt dat het programma correct werkt, zie figuur 9.

[4] Hoe je dit in Code::Blocks kunt doen kun je vinden in appendix B

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void)
{
    int n;

    do
    {
        printf("Geef de waarde van n (1..6): ");
        scanf("%d", &n);
    }
    while(n < 1 || n > 6);

    for (int factor = 1; factor < 11; factor = factor + 1)
    {
        for (int tafel = 1; tafel < n + 1; tafel = tafel + 1)
        {
            printf("%2d x %d = %2d ", factor, tafel, ↩
             ↪ factor * tafel);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 8: Stap 5: tafels_stap5.c.

```
Geef de waarde van n (1..6): 5
 1 x 1 =  1  1 x 2 =  2  1 x 3 =  3  1 x 4 =  4  1 x 5 =  5
 2 x 1 =  2  2 x 2 =  4  2 x 3 =  6  2 x 4 =  8  2 x 5 = 10
 3 x 1 =  3  3 x 2 =  6  3 x 3 =  9  3 x 4 = 12  3 x 5 = 15
 4 x 1 =  4  4 x 2 =  8  4 x 3 = 12  4 x 4 = 16  4 x 5 = 20
 5 x 1 =  5  5 x 2 = 10  5 x 3 = 15  5 x 4 = 20  5 x 5 = 25
 6 x 1 =  6  6 x 2 = 12  6 x 3 = 18  6 x 4 = 24  6 x 5 = 30
 7 x 1 =  7  7 x 2 = 14  7 x 3 = 21  7 x 4 = 28  7 x 5 = 35
 8 x 1 =  8  8 x 2 = 16  8 x 3 = 24  8 x 4 = 32  8 x 5 = 40
 9 x 1 =  9  9 x 2 = 18  9 x 3 = 27  9 x 4 = 36  9 x 5 = 45
10 x 1 = 10 10 x 2 = 20 10 x 3 = 30 10 x 4 = 40 10 x 5 = 50
```

Figuur 9: Uitvoer van het programma uit listing 8.

Stap 6: Een laatste verbetering

Bij het testen type ik per ongeluk een letter in. Het programma komt nu in een oneinige loop. Een deel van de uitvoer is te zien in figuur 10.

```
Geef de waarde van n (1..6): a
Geef de waarde van n (1..6): Geef de waarde van n (1..6): G
eef de waarde van n (1..6): Geef de waarde van n (1..6): Ge
ef de waarde van n (1..6): Geef de waarde van n (1..6): Gee
f de waarde van n (1..6): Geef de waarde van n (1..6): Geef
 de waarde van n (1..6): Geef de waarde van n (1..6): Geef
de waarde van n (1..6): Geef de waarde van n (1..6): Geef d
e waarde van n (1..6): Geef de waarde van n (1..6): Geef de
```

Figuur 10: Uitvoer van het programma uit listing 8.

Bij navraag blijkt dit een bekend probleem te zijn. Zie: Hoe voorkom ik dat mijn programma rare dingen doet als in plaats van een getal een letter wordt ingetypt?

Na bovenstaande verwijzing bestudeerd te hebben kom ik tot het programma dat gegeven is in listing 9.

Bij het testen blijkt dat het programma correct werkt, ook als je tekst invoert, zie figuur 11.

```c
#include <stdio.h>

/* Copyright 2016 Hogeschool Rotterdam */
/* Dit programma leest een geheel getal 0 < n < 7 en drukt
   vervolgens de tafels van 1 t/m n naast elkaar af */

int main(void)
{
    int n;

    do
    {
        printf("Geef de waarde van n (1..6): ");
        fflush(stdin);
    }
    while (scanf("%d", &n) != 1 || n < 1 || n > 6);

    for (int factor = 1; factor < 11; factor = factor + 1)
    {
        for (int tafel = 1; tafel < n + 1; tafel = tafel + 1)
        {
            printf("%2d x %d = %2d ", factor, tafel, ↩
             ↪ factor * tafel);
        }
        printf("\n");
    }

    return 0;
}
```

Listing 9: Stap 6: tafels_stap6.c.

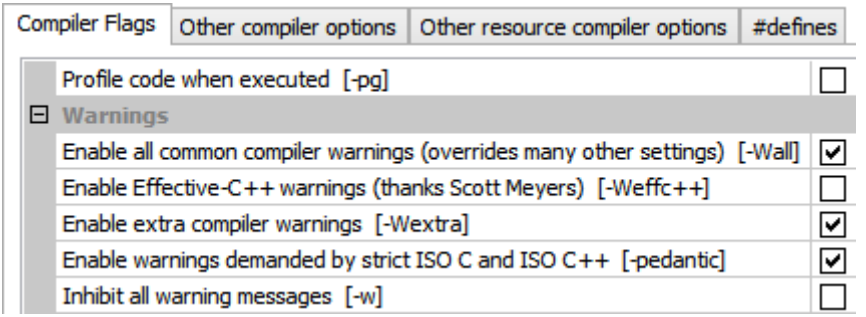
```
Geef de waarde van n (1..6): a
Geef de waarde van n (1..6): Doe maar wat!
Geef de waarde van n (1..6): -3
Geef de waarde van n (1..6): 0
Geef de waarde van n (1..6): 1000
Geef de waarde van n (1..6): 7
Geef de waarde van n (1..6): 4
 1 x 1 =  1  1 x 2 =  2  1 x 3 =  3  1 x 4 =  4
 2 x 1 =  2  2 x 2 =  4  2 x 3 =  6  2 x 4 =  8
 3 x 1 =  3  3 x 2 =  6  3 x 3 =  9  3 x 4 = 12
 4 x 1 =  4  4 x 2 =  8  4 x 3 = 12  4 x 4 = 16
 5 x 1 =  5  5 x 2 = 10  5 x 3 = 15  5 x 4 = 20
 6 x 1 =  6  6 x 2 = 12  6 x 3 = 18  6 x 4 = 24
 7 x 1 =  7  7 x 2 = 14  7 x 3 = 21  7 x 4 = 28
 8 x 1 =  8  8 x 2 = 16  8 x 3 = 24  8 x 4 = 32
 9 x 1 =  9  9 x 2 = 18  9 x 3 = 27  9 x 4 = 36
10 x 1 = 10 10 x 2 = 20 10 x 3 = 30 10 x 4 = 40
```

Figuur 11: Uitvoer van het programma uit listing 9.

Appendices

A Compiler warnings aanzetten in Code::Blocks

Om de compiler warnings aan te zetten kies je voor de menuoptie: Settings › Compiler... . Vink vervolgens de opties `-Wall`, `-Wextra` en `-pedantic` aan, zie figuur 12.

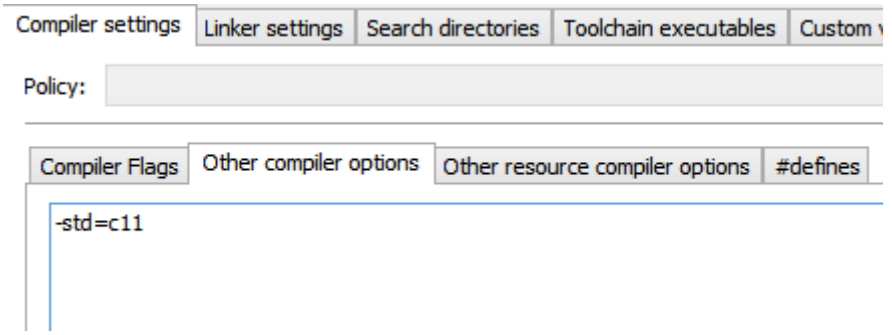


Figuur 12: Vink de aangegeven opties aan om de warnings aan te zetten.

B Compiler configureren voor C11 in Code::Blocks

Om de compiler de C11 standaard[5] te laten gebruiken kies je voor de menuoptie: Settings › Compiler... . Klik vervolgens op het tabblad “Other options” en type daar in: `-std=c11`, zie figuur 13.

[5] C11 is de informele naam voor de ISO/IEC 9899:2011 standaard. Dit is, op het moment van schrijven, de meest recente C standaard. Zie Wikipedia.



Figuur 13: Specificeer de optie `-std=c11` zodat de compiler C11 gebruikt.