Inleiding

Bij het vak CPL01 leer je om programma's te schrijven in de taal C. Deze handleiding beschrijft hoe je Code::Blocks kunt gebruiken om C programma's te ontwikkelen en te debuggen (logische fouten op te sporen).

Downloaden en installeren

Je kunt Code::Blocks gratis downloaden[1]. De installatie wijst zichzelf.

Eenvoudig C programma compileren en uitvoeren met Code::Blocks.

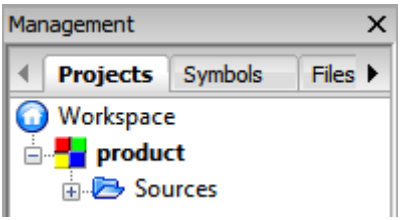
Start Code::Blocks en kies menu File › New › Project... . Dubbelklik op "Console application". Selecteer "C" en klik op Next > . Vervolgens moet je een naam voor het project kiezen, bijvoorbeeld "product". Indien gewenst kun je ook de folder waar het project opgeslagen wordt aanpassen. Klik vervolgens op Next > en tot slot op Finish .

Code::Blocks heeft nu een project met een programma aangemaakt maar opent de source code van het programma, vreemd genoeg, niet automatisch. Klik in het "Management" window op het plusje voor "Sources", zie figuur 1. Dubbelklik vervolgens op "main.c" om de source code van het programma te openen.

Je kunt dit programma compileren (vertalen naar machinecode) en starten:

- door op het knopje "Build and run"  te klikken of
- door het menu Build › Build and run te kiezen of
- door op functietoets F9 te drukken.

[1] https://sourceforge.net/projects/codeblocks/files/Binaries/16.01/Windows/codeblocks-16.01mingw-setup.exe/download



Figuur 1: Het "Management" window nadat het project "product" is aangemaakt.

Als het goed is, verschijnt er een console window met de tekst "Hello world!". Klik op Enter om dit window te sluiten.

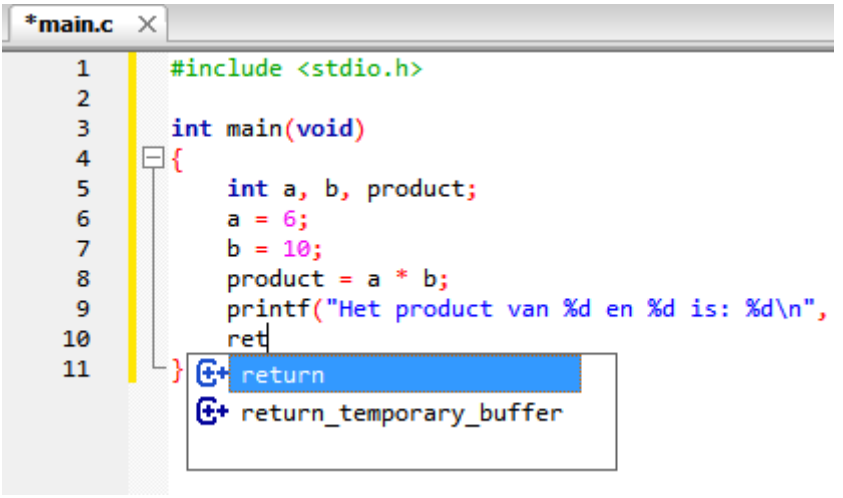
Maak nu het Edit window leeg door op Ctrl + A en vervolgens op Delete te drukken. Type vervolgens het programma dat gegeven is in listing 1 over in het Edit window.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a, b, product;
6      a = 6;
7      b = 10;
8      product = a * b;
9      printf("Het product van %d en %d is: %d\n", a, b, ↩
        ↪ product);
10     return 0;
11 }
```

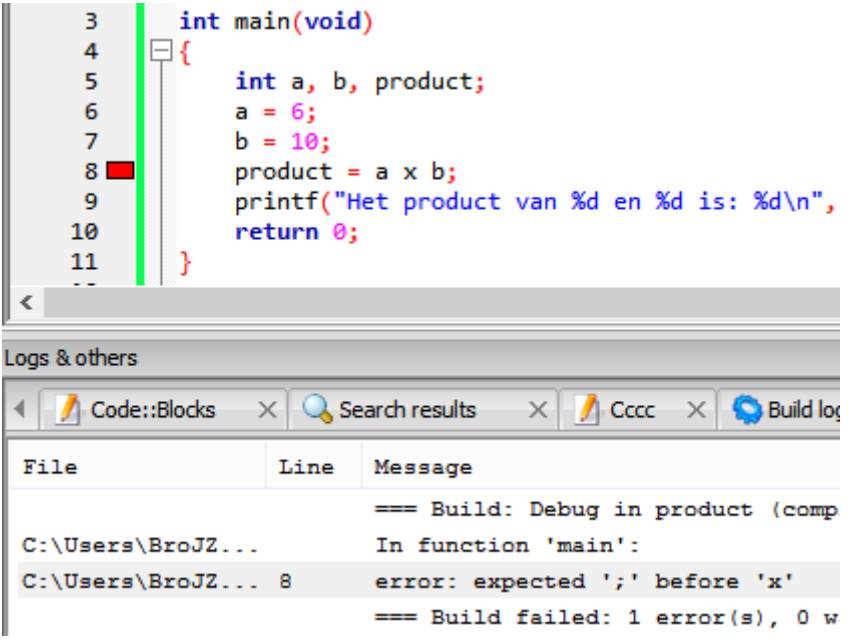
Listing 1: Dit programma berekent het product van 6 en 10.

Merk op dat Code::Blocks je helpt bij het typen. Als je bijvoorbeeld `ret` typt dan geeft Code::Blocks een aantal suggesties, zie figuur 2. Als Code::Blocks dat niet automatisch doet, kun je om suggesties vragen door op Ctrl + Space te drukken.

Compileer en run het programma nadat je het hebt overgetypt. Als je een typefout hebt gemaakt kun je een compilatiefout krijgen, zie bijvoorbeeld figuur 3. Kun je zien welke typefout ik daar gemaakt heb?



Figuur 2: Code::Blocks geeft suggesties tijdens het typen.



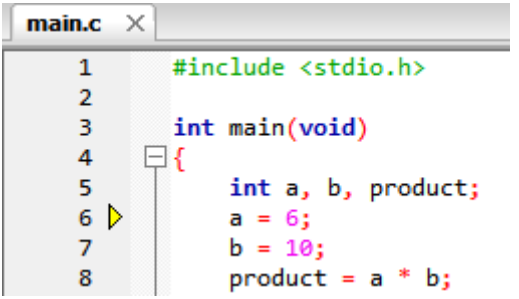
Figuur 3: Dit programma bevat een zogenoemde syntaxisfout.

Programma stap voor stap uitvoeren

Je kunt het programma ook stap voor stap uitvoeren. Dat is erg handig als je een logische fout in je programma wilt opsporen.

Met de functietoets Shift + F7 of door op het knopje “Step into”  te klikken, kun je in het programma stappen.

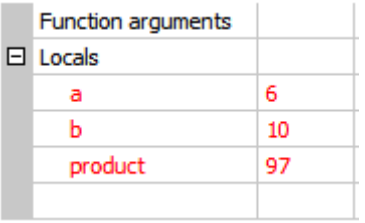
Als je op Shift + F7 drukt stopt het uitvoeren bij de eerste instructie in de functie `main`. Een geel pijltje geeft aan waar de uitvoering gestopt is, zie figuur 4.



Figuur 4: Het uitvoeren van het progamma is gestopt bij de gele pijl.

Met de functietoets F7 of door op het knopje “Next line”  te klikken kun je stap voor stap (regel voor regel) door het programma lopen.

Druk tweemaal op F7 zodat de gele pijl na regelnummer 8 staat. Je kunt nu de waarde van de variabelen bekijken door de menu optie Debug > Debugging windows > Watches te kiezen. Er verschijnt een window waarin je de waarde van de verschillende variabelen kunt zien, zie figuur 5. Je ziet dat de variabelen a en b respectievelijk de waarde `6` en `10` hebben. De variabele `product` heeft nog een willekeurige waarde. Als je nogmaals op F7 drukt, zie je dat de variabele `product` de waarde `60` krijgt. Je kunt het programma verder laten runnen door op de knopje “Debug / Continue”  te klikken.



Figuur 5: Het Watch window net voordat de instructie `product = a * b` wordt uitgevoerd.

Als je (later) een eigen functie of hebt geschreven en je wilt dat deze functie ook stap voor stap wordt uitgevoerd dan moet je op Shift + F7 (Step Into) drukken op het moment dat deze functie wordt aangeroepen.

Als er logische fouten in je programma zitten dan is het soms erg lastig om die fouten te vinden. We kunnen de debugger gebruiken om te kijken of het programma doet wat we verwachten. Hieronder staan een aantal eenvoudige programma's. Deze programma's bevatten logische fouten, die niet door de compiler worden gemeld. Het lijkt dus alsof het programma prima werkt (volgens de compiler), maar de uitvoer van de programma's is niet juist. Door te debuggen kun je ontdekken in welk statement een logische fout zich heeft verstopt.

Voorbeeld 1 – Bol

Om dit voorbeeld te begrijpen moet je bekend zijn met floating point variabelen in C.

Kees wil een programma schrijven dat de inhoud van een bol berekent als de straal van deze bol bekend is. Hij weet dat de inhoud van een bol met vergelijking (1) berekend kan worden.

$$inhoud = \frac{4 \cdot \pi \cdot straal^3}{3} \tag{1}$$

Zijn programma `bol.c` is te vinden in listing 2.

```c
#include <stdio.h>

int main(void)
{
    double inhoud;
    int straal = 2;
    double vier_pi = (4 * 3,1415926535897932);

    printf("Debug Demonstratie Code::Blocks\n");

    inhoud = (straal * straal * straal * vier_pi) / 3;
    printf("Een bol met een straal van %d cm heeft een ↩
     ↪ inhoud van %f cm^3", straal, inhoud);
    return 0;
}
```

Listing 2: Het programma dat Kees heeft geschreven om de inhoud van een bol met een straal van 2 cm te berekenen.

Kees compileert het programma en het blijkt dat er geen syntaxisfouten in zitten. Als hij het programma uitvoert blijkt echter dat de inhoud van de bol niet juist berekend is. De uitvoer van het programma is gegeven in figuur 6.

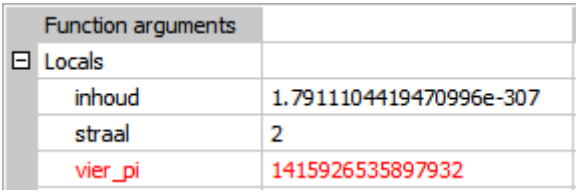
```
Debug Demonstratie Code::Blocks
Een bol met een straal van 2 cm heeft een inhoud van 37.333333 cm^3
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Figuur 6: De uitvoer van het programma uit listing 2.

De juiste inhoud van een bol van 2 cm is echter 33.510322 cm^3. We gaan nu uitzoeken waar deze fout zit.

Met Shift + F7 stappen we in het programma en met F7 lopen we stap voor stap door het programma heen. We openen het Watch window en controleren de waarde van alle variabelen na elke stap. Na twee stappen is de waarde van `vier_pi` berekent, zie figuur 7.

Je ziet nu dat de variabele `vier_pi` de waarde 1415926535897932 heeft gekregen. Dat klopt natuurlijk niet.



Figuur 7: Het Watch window net voordat de instructie `printf(...)` wordt uitgevoerd.

Er zit dus een fout in de regel:

```
    double vier_pi = (4 * 3,1415926535897932);
```

Zie jij de fout?

In C moet de floating point constante 3,1415926535897932 ingetypt worden als 3.1415926535897932.

Na deze correctie blijkt het programma correct te werken, zie figuur 8.

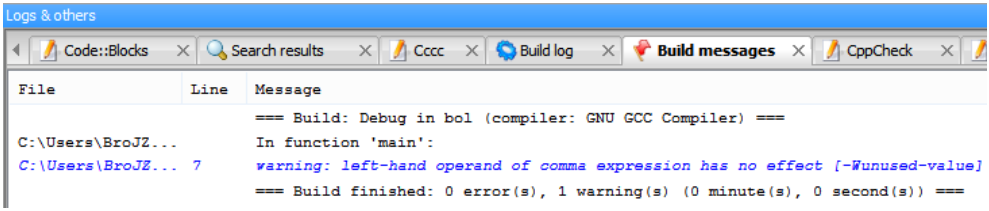
```
Debug Demonstratie Code::Blocks
Een bol met een straal van 2 cm heeft een inhoud van 33.510322 cm^3
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Figuur 8: De uitvoer van het programma uit listing 2 nadat de fout gecorrigeerd is.

Kees had zich overigens de moeite van het debuggen kunnen besparen als hij beter had opgelet. De compiler geeft namelijk bij het compileren een waarschuwing (Engels: warning). Deze warning is te vinden in het “Logs & others” window onder de tab “Build messages”, zie figuur 9.

Standaard wordt door Code::Blocks bij het compileren de optie `-Wall` gebruikt waarmee vele waarschuwingen worden gegeven. Het is verstandig om zoveel mogelijk waarschuwingen te laten genereren door de compiler, zie appendix A. Natuurlijk moet je er dan bij het compileren wel op letten of er warnings gegeven worden.

Kees heeft het programma nu aangepast zoals gegeven in listing 3, zie `bolCorrect`.



Figuur 9: De warning die de compiler geeft bij het vertalen van het programma uit listing 2.

```c
#include <stdio.h>

int main(void)
{
    int straal = 2;
    double vier_pi = 4 * 3.1415926535897932;

    printf("Debug Demonstratie Code::Blocks\n");

    double inhoud = (straal * straal * straal * vier_pi) / 3;
    printf("Een bol met een straal van %d cm heeft een ↩
     ↪ inhoud van %f cm^3", straal, inhoud);
    return 0;
}
```

Listing 3: Het gecorrigeerde programma van Kees.

Bij het compileren krijgt hij bij regel 10 de warning: “warning: ISO C90 forbids mixed declarations and code [-Wpedantic]”. De compiler gebruikt namelijk standaard een verouderde versie van C waarbij alle variabelen meteen na een accolade openen gedeclareerd moeten worden. In appendix B wordt uitgelegd hoe je de compiler de meest recente versie van C kan laten gebruiken.

Voorbeeld 2 – Machtverheffen

Om dit voorbeeld te begrijpen moet je bekend zijn met functies in C.

Gegeven is het programma `machtverheffen.c`, zie listing 4.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int xtoy(int x, int y);
6      printf("Debug Demonstratie Code::Blocks\n");
7      int macht = xtoy(7, 8);
8      printf("\n7 tot de macht 8 = %d", macht);
9      return 0;
10 }
11
12 int xtoy(int x, int y)
13 {
14     int answer = x;
15     for(int tel = 0; tel < y; tel++)
16     {
17         answer = answer * x;
18     }
19     return answer;
20 }
```

Listing 4: De functie `xtoy` berekent x^y, waarbij x en y gehele getallen zijn.

Neem dit programma over en voer het uit. In dit programma wordt 7^8 uitgerekend. Zoals iedereen kan nagaan is het juiste antwoord 5764801. Het programma levert echter een verkeerde waarde. Aan jou de opdracht om uit te zoeken waar

deze fout zit. De werkwijze is dezelfde als bij de vorige opdracht. Stap door het programma heen en hou de variabelen in de gaten. Je wilt ook binnenin de functie `xtoy` stappen, gebruik dus Shift+F7 "Step Into" op het moment dat de functie wordt aangeroepen. Je kunt er ook voor zorgen dat het programma pas stopt als de functie `xtoy` wordt aangeroepen. Dit doe je door een zogenoemd "Breakpoint" te plaatsen op de eerste regel van de functie. Dit kun je doen door de cursor op de eerste regel van de functie te plaatsen en vervolgens op functietoets F5 te drukken. Je kunt ook net na het regelnummer klikken met de muis. Er verschijnt een rode punt achter het regelnummer om aan te geven dat er een breakpoint op de regel staat, zie figuur 10.

```
 9         return 0;
10     }
11
12     int xtoy(int x, int y)
13     {
14         int answer = x;
15         for(int tel = 0; tel < y; tel++)
```

Figuur 10: Er staat een breakpoint op regel 12.

Als je het programma nu start door op functietoets F8 te drukken of door op het knopje "Debug / Continue"  te klikken, zal het uitvoeren van het programma onderbroken worden bij het breakpoint.

Kun je de fout vinden en herstellen? Zo niet, mail dan voor een tip: J.Z.M.Broeders@hr.nl.

Voorbeeld 3 – Optellen

Om dit voorbeeld te begrijpen moet je bekend zijn met array variabelen in C. In dit voorbeeld wordt de variabele `tel` in de `for`-loop aangemaakt. Dit is pas sinds C99 toegestaan. In appendix B kun je vinden hoe je de compiler kunt vertellen dat je de meeste recente versie van C gebruikt.

Gegeven is het programma `optellen.c`, zie listing 5.

```c
#include <stdio.h>

int main(void)
{
    int array[] = {3, 5, 6, 9};
    int telOp(int rij[], int aantal);

    printf("Debug Demonstratie Code::Blocks\n");
    int som = telOp(array, sizeof(array)/sizeof(array[0]));
    printf("De som van de 4 getallen is: %d", som);
    return 0;
}

int telOp(int rij[], int aantal)
{
    int answer;
    for (int tel = 0; tel < aantal; tel++)
    {
        answer = answer + rij[tel];
    }
    return answer;
}
```

Listing 5: De functie `telOp` berekent de som van een rij gehele getallen.

De uitvoer van het programma is niet correct (als de uitvoer wel correct is, dan is dat toeval).

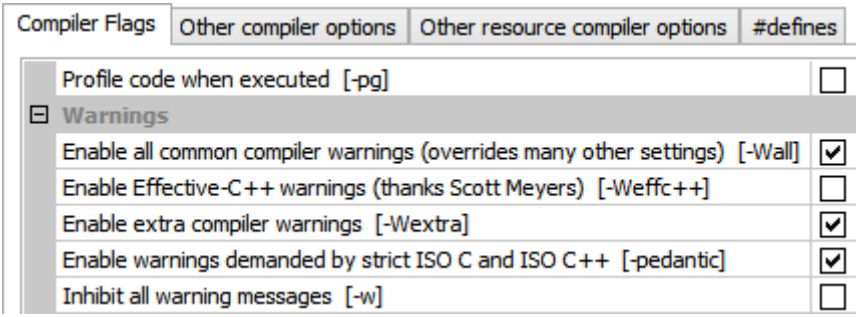
Gebruik de debug technieken die je hebt geleerd om de fout te vinden en te herstellen.

Lukt het niet, mail dan voor een tip: J.Z.M.Broeders@hr.nl.

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 11.

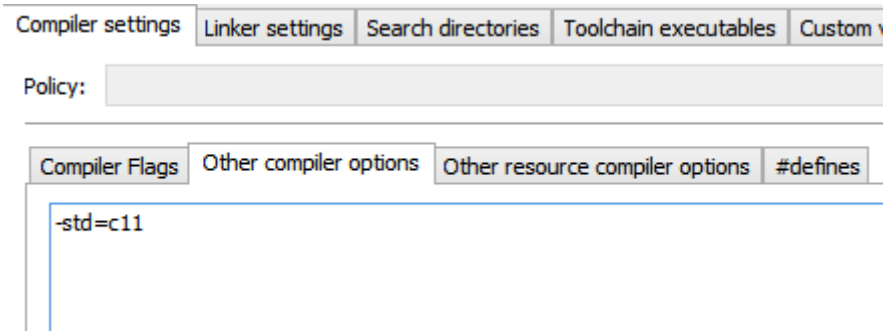


Figuur 11: Vink de aangegeven opties aan om de warnings aan te zetten.

B Compiler configureren voor C11 in Code::Blocks

Om de compiler de C11 standaard[2] 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 12.

[2] 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 12: Specificeer de optie `-std=c11` zodat de compiler C11 gebruikt.