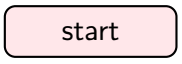
4

Flowcharts

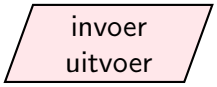
Een *flowchart* (Nederlands: stroomschema) is een grafische weergave van een (deel van een) computerprogramma. Flowcharts gebruiken rechthoeken, ovalen, ruiten en mogelijk andere vormen om het type stap te definiëren om samen met verbindingspijlen de stroom en volgorde te definiëren. Ze kunnen variëren van eenvoudige, met de hand getekende diagrammen tot uitgebreide computergegenereerde diagrammen die meerdere stappen en routes weergeven. Als we alle verschillende vormen van stroomdiagrammen beschouwen, zijn ze een van de meest voorkomende diagrammen op aarde, die door zowel technische als niet-technische mensen gebruikt worden. Ze zijn gerelateerd aan andere populaire diagrammen, zoals Data Flow Diagrams (DFD's) en Unified Modeling Language (UML) activiteitendiagrammen.

Het is belangrijk om een flowchart overzichtelijk te houden. Het heeft geen zin om op een A4'tje bijvoorbeeld 40 symbolen te tekenen. Afhankelijk van de grootte van het programma moeten we stukken code "comprimeren". Zo zouden we de code voor het inlezen van tien elementen van een array als één statement in een flowchart kunnen tekenen met de tekst "lees array in".

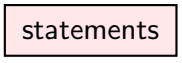
De gebruikte symbolen in een flowchart staan hieronder weergegeven:



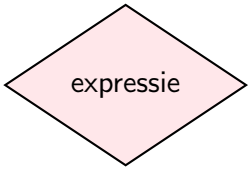
Dit symbool geeft het begin of het einde van de flowchart aan. Naast "start" en "stop" kunnen ook de termen "begin" en "einde" gebruikt worden.



Dit symbool geeft invoer of uitvoer aan, bijvoorbeeld het inlezen van een getal of het afdrukken van tekst.



Dit symbool wordt gebruikt om statements anders dan invoer en uitvoer aan te duiden. Een statement kan bijvoorbeeld een berekening zijn, maar ook een functie-aanroep. Meerdere sequentiële statements mogen samengenomen worden in één symbool.



Dit symbool geeft een beslissing aan. De expressie kan alleen maar “waar” of “niet waar“ opleveren. Een beslissing levert een vertakking op. De vertakkingen kunnen naar links of rechts zijn én naar beneden (dus niet links en rechts). Bij de uitgaande vertakkingen worden de woorden “yes” en “no” geschreven. Naast “yes” en “no” kunnen ook de termen “true” of “T” en “false” of “F” gebruikt worden.

Een pijl geeft een *flow* aan. De pijl begint bij een van de symbolen en eindigt bij een symbool of een andere pijl. Bij een symbool komt altijd maar één pijl aan.

Een connector kan gebruikt worden om een plek aan te geven waar twee pijlpunten samenkomen. Dit symbool wordt niet in dit boek gebruikt.

Voorbeelden van invoer en uitvoer zijn: “lees i” en “print k”. Voorbeelden van statements zijn (geen invoer en uitvoer): “$j = j + 1$” en “$k = 2 * j$”. Voorbeelden van expressies zijn: “$j < 10$” en “$a > 5$ en $a < 25$”.

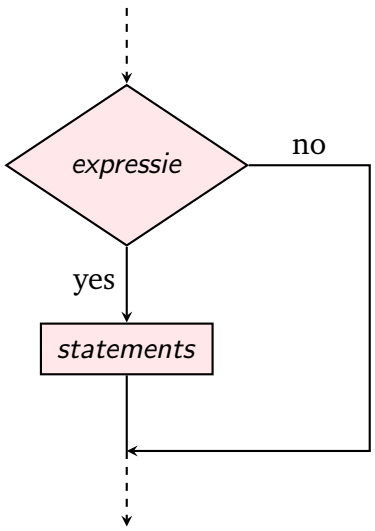
4.1 If-statement

Een if-statement is te modelleren door middel van een beslissing en een statement.

```
1 if (expressie) {
2     statements
3 }
```

Listing 4.1: *if-statement in C*

De flowchart is hieronder weergegeven.



Figuur 4.1: *Flowchart van een if-statement.*

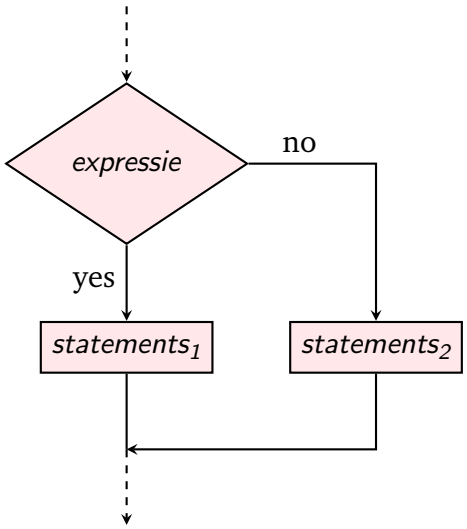
4.2 If-else-statement

Een if-else-statement van de vorm

```
if (expressie) {
    statements_1
else {
    statements_2
}
```

Listing 4.2: *If-else-statement in C.*

is in een flowchart te tekenen als



Figuur 4.2: *Flowchart van een if-else-statement.*

4.3 `while`-lus en `do-while`-lus

Een `while`-lus heeft de gedaante:

```
while (expressie) {
    statements
}
```

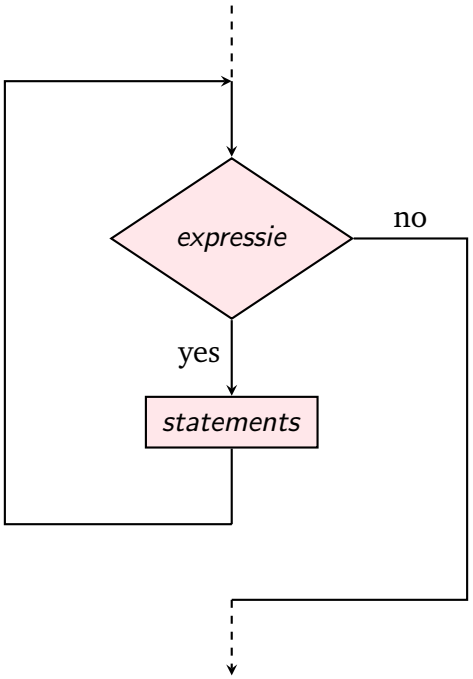
Listing 4.3: *while-lus in C.*

wordt weergegeven als flowchart (figuur 4.3):

De `do-while`-lus heeft in C de volgende gedaante:

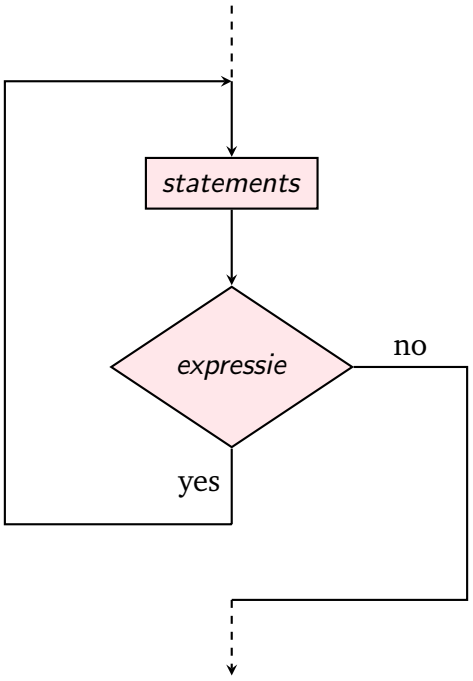
```
do {
    statements
} while (expressie);
```

Listing 4.4: *Do-while-lus in C.*



Figuur 4.3: *Flowchart van een* `while`*-lus.*

De flowchart is te vinden in figuur 4.4. Let erop dat *statements* minstens één keer wordt uitgevoerd, ook al is *expressie* niet waar.



Figuur 4.4: *Flowchart van een* `do-while`*-lus.*

4.4 `for`-lus

Een for-lus van de gedaante:

```
for (statements1; expressie; statements2) {
    statements3
}
```

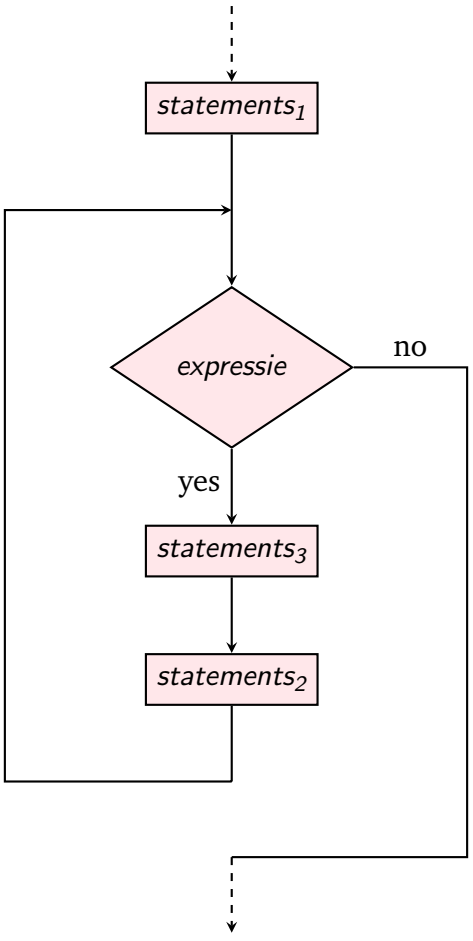
Listing 4.5: *For-lus in C.*

is *semantisch identiek* aan:

```
statements1
while (expressie) {
    statements3
    statements2
}
```

Listing 4.6: *For-lus herschreven als while-lus in C.*

Let erop dat $statements_2$ ná $statements_3$ komt. De flowchart is hieronder te vinden.



Figuur 4.5: *Flowchart van een for-lus.*

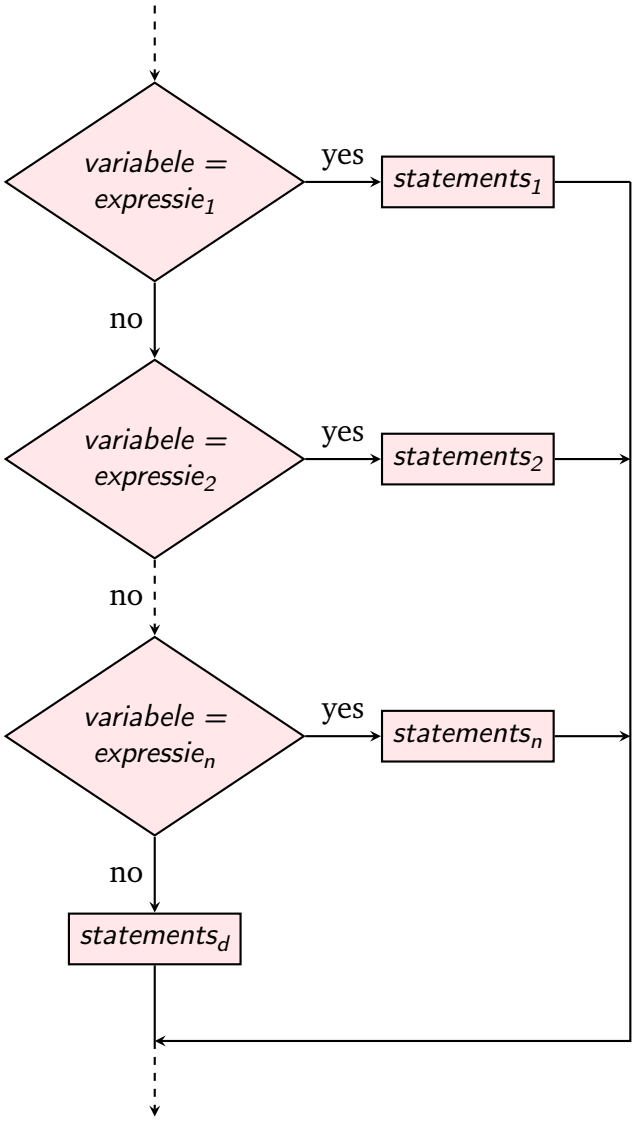
4.5 `switch`-statement

Een `switch`-statement is een meervoudige `if`-statement met steeds dezelfde variabele.

```
switch (variabele) {
    case expressie_1: statements_1
    case expressie_2: statements_2
    ...
    case expressie_n: statements_n
    default : statements_d
}
```

Listing 4.7: *switch-statement in C.*

Merk op dat $expressie_1$, $expressie_2$, ..., $expressie_n$ een constante moeten opleveren. De flowchart is te zien in figuur 4.6.



Figuur 4.6: *Flowchart van een switch-statement.*

4.6 Voorbeeld

We zullen een flowchart presenteren van een kort programma. Hieronder is een een eenvoudig C-programma gegeven dat een variabele k inleest en vervolgens de som bepaalt van alle cijfer tussen 1 en k (inclusief), dus:

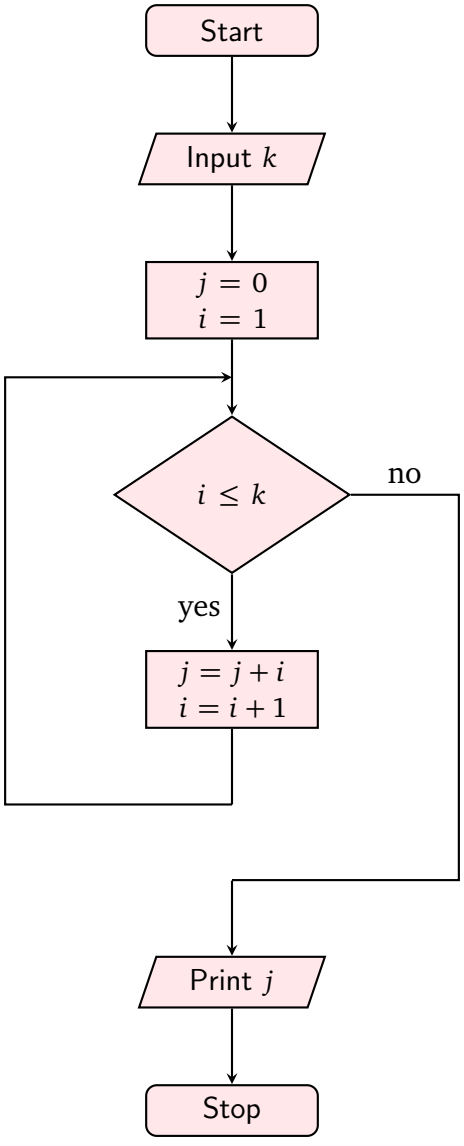
$$j = 1 + 2 + 3 + 4 + \cdots + k \tag{4.1}$$

```c
#include <stdio.h>

int main(void) {

    int k, i, j;

    scanf("%d", &k);

    j=0;

    for (i=1; i<=k; i=i+1) {
        j=j+i;
    }

    printf("%d\n", j);

    return 0;
}
```

Listing 4.8: *Codevoorbeeld in C.*

We hebben hier te maken met het begin en einde van het programma, het inlezen en afdrukken van variabelen en het doorlopen van een for-lus waarbij variabelen worden aangepast. Uiteindelijk komen uit bij het einde van het programma.

In figuur 4.7 is de flowchart te zien.



Figuur 4.7: *Voorbeeld van een flowchart.*

Index