

Gestructureerd programmeren in C

Practicumopdrachten

DE HAAGSE
HOGESCHOOL

Opleiding Elektrotechniek

Versie 1.9999γ
25 juni 2020

Inhoudsopgave

Aanbevolen werkwijze	4
1 Inleiding	5
1.1 De gebruikte C-compiler	5
1.2 Introductie Code::Blocks IDE	5
1.3 Programma wijzigen	5
1.4 Vertalen van het gewijzigde programma	5
1.5 Programma met invoer	6
2 Toekennings- en herhalingsopdrachten (for loop)	7
2.1 Inleiding	7
2.2 Inwendig en uitwendig vermogen	8
2.3 Parallelweerstand	8
2.4 Bazel-probleem	8
2.5 De spaarrekening	8
2.6 De transformator	9
2.7 Faculteit	10
2.8 Afgelegde weg	10
2.9 Benadering sinus	11
2.10 Kerstboom	11
2.11 Maandkalender	12
2.12 Wind-chill temperatuur	12
2.13 Benadering e	12
2.14 Opladen van een condensator	13
2.15 Booglengte van één periode van een sinus	14
3 Beslissingsopdrachten (if, switch)	15
3.1 Inleiding	15
3.2 Even en oneven	15
3.3 Voldoende en onvoldoende	16
3.4 Overwerk	16
3.5 ABC-formule	16
3.6 Simpele rekenmachine	16
3.7 Bepalen schrikkeljaar	17
3.8 Afdrukken binair getal	17
3.9 Windsnelheden	17
3.10 Geschreven letters	18
3.11 Stijgen	18
3.12 Tekens tellen	18
3.13 Dag van de week	18
4 Functies	20
4.1 Inleiding	20
4.2 Catalan-getal (iteratief)	21
4.3 Catalan-getal (recursief)	21
4.4 Vermoeden van Collatz	21
4.5 Digitale techniek	22
4.6 Maximale	23
4.7 Gemiddelde	23
4.8 Kleinste gemene veelvoud	23
4.9 Wortel schieten	24
4.10 Som van de cijfers en modulo 9	25
4.11 Isolated prime	26
4.12 De Ackermann-functie	26

4.13	sinc	26
4.14	Aantal delers van een geheel getal	26
5	Pointers en array's	27
5.1	Inleiding	27
5.2	Omkeren	27
5.3	Voortschrijden	28
5.4	Histogram	28
5.5	Sorteren met "selection sort"	28
5.6	Palindroom	28
5.7	Sorteren met "bubble sort"	29
5.8	Zoeken	29
5.9	Sinustabel	29
5.10	Cesuur aanpassen	29
5.11	Lineaire regressie	30
5.12	Controle identificatiecode van Nederland eurobankbiljet	30
5.13	Doorsnede	30
5.14	Uniek	31
5.15	Run Length encoding	31
5.16	Rolling dice	32
5.17	Overtollige spaties en tabs verwijderen	32
5.18	Vervangen	32
5.19	Onbeperkt	33
5.20	Waar of niet waar	33
6	Eindopdracht	35
6.1	Inleiding	35
6.2	More	35
6.3	Cat	36
6.4	Zinslengte sorteren	36
6.5	Op alfabet sorteren	36
6.6	Regelnummering	36
6.7	Getallen sorteren	36
6.8	Uitgeschreven nummers sorteren	36
6.9	C keywords	36
6.10	Commentaar verwijderen	37
6.11	Letters tellen	37
6.12	XeT	37
A	Installatie Code::Blocks	39
B	Inlezen van een getal met scanf	40
C	Introductie array's en pointers	45
C.1	Array's	45
C.2	Strings	46
C.3	Pointers	47
D	Command line argumenten en bestandsafhandeling in C	49
D.1	Command line argumenten	49
D.2	Bestandsafhandeling in C	50
D.3	Command line argumenten en bestandsafhandeling	51
D.4	Openen van een DOS-box op Windows	52

Dit document is gemaakt door F. Theinert, W. Muhammad, B. Kuiper, M. Schrauwen, H. Broeders en J. op den Brouw. Opmerkingen en suggesties kunnen worden gemaild naar J.E.J.opdenBrouw@hhs.nl

Aanbevolen werkwijze

Beste student,

Inmiddels hebben wij dit vak al een aantal jaar gegeven en gezien wat voor gedrag jullie vertonen tijdens het volgen van deze practica. Volg onderstaande werkwijze als je geïnteresseerd bent in aanzienlijk efficiënter en sneller tot een voldoende eindresultaat te komen. Het is natuurlijk aan jou om deze aanbevolen werkwijze te volgen:

1. Lees goed de opdracht

De meest gemaakte fout is dat de opdracht niet goed wordt gelezen. Stel jezelf na het lezen van de opdracht de vraag wat het programma precies moet kunnen. Wat moet je opleveren?

2. Teken je programma uit met pen en papier

Een andere veel gemaakte fout is dat de student ondoordacht (lees: zonder na te denken) begint met programmeren. STOP! Als je de opdracht hebt begrepen, ga je niet programmeren, maar pak je pen en papier. Als pen en papier voor je ligt, ga je uitschrijven, uitdenken en uittekenen hoe het programma moet zijn opgebouwd. Er zijn verschillende methodieken die een dergelijke werkwijze ondersteunen¹. Dit heeft aanzienlijke voordelen. Je leert namelijk een moeilijk probleem in te delen in gemakkelijkere deelproblemen (analytische werkwijze). Bovendien heb je tijdens het programmeren van je uitgeschreven ontwerp op papier een houvast. Wat niet onvermeld kan blijven, is dat je tijdens het tentamen ook moet programmeren maar dan met pen op papier. Als je van begin af aan tijdens de practica² je opdrachten op papier uitwerkt dan ben je ten tijde van het tentamen aanzienlijk in het voordeel.

3. Gebruik testdata

Zorg dat je tijdens het ontwikkelen van je programma testdata tot je beschikking hebt. Als je op papier het programma hebt uitgetekend dan heb je ook eventueel benodigde berekeningen uitgewerkt. Deze berekeningen hebben een resultaat. Jouw programma moet uiteindelijk bij dezelfde invoer dat resultaat geven. Dit stelt je in staat om je programma continu te testen. Zorg altijd voor meerdere sets met testdata!

4. Werk systematisch

Programmeer stap voor stap de onderdelen die je op papier hebt uitgewerkt. Bedenk ook hoe je elk onderdeel kunt testen en test dit dan ook. Wij hebben vaker gezien dat de enkele student die van nature systematisch te werk gaat eerder klaar is met de opdrachten en tegelijkertijd minder fouten maakt. Bovendien is systematisch werken goed aan te leren.

5. Gebruik de debugger

Als je niet begrijpt waar het programma fout gaat, steek dan niet direct je hand omhoog. Zet in je code op de plek waar het programma vermoedelijk nog correct werkt een *breakpoint*³. Als het programma tijdens het debuggen is gestopt op de plek van de breakpoint, controleer dan goed de inhoud van alle actuele variabelen. Dit is één van de krachtigste manieren om fouten te detecteren en om tot een werkend programma te komen.

6. Gebruik géén goto-statement

Nog een opmerking moet hier gesteld worden: het gebruik van het goto-statement is niet toegestaan, onder het mom van “gotos are the root of all evil”.

7. Gebruik geen spaties en vreemde tekens in bestandsnamen, projectnamen en dergelijke

Windows heeft de onhebbelijke eigenschap om bestandsnamen met spaties toe te staan. Doe het niet! Het leest niet lekker en programma's kunnen niet (goed) functioneren, zeker in combinatie met command line argumenten.

Veel succes!

¹ Dit komt tijdens de colleges nog aan bod.

² En tijdens het maken van huiswerkopgaven.

³ In de les wordt uitgelegd hoe je dit moet doen.

1 Inleiding

De practicumdocent zal tijdens het practicum uit hoofdstukken 2 t/m 6 één opdracht aanwijzen die je moet maken. De opdrachten in hoofdstuk 1 moet je allemaal maken.

1.1 De gebruikte C-compiler

Wij hebben deze opdrachten uitgewerkt met behulp van **Code::Blocks**. Code::Blocks is een gratis compiler suite met geïntegreerde invoer-omgeving (een zogenoemde IDE = Integrated Development Environment). Hoe je Code::Blocks op je laptop kan installeren kun je vinden in bijlage A.

Code::Blocks draait op Windows (7, 8, 8.1, 10), op Linux (Ubuntu, Debian, Fedora Core en CentOS). Er is helaas alleen een oudere versie die op OS-X draait.

Gebruikers van een iMac kunnen ook gebruik maken van **Xcode** die door Apple zelf ontwikkeld is. De functies van Xcode zijn vergelijkbaar met Code::Blocks.

Gebruikers van Windows kunnen ook Microsoft's **Visual Studio Community** installeren. Visual Studio wordt door zeer veel bedrijven gebruikt. De Community editie is gratis te gebruiken maar je moet wel een Microsoft account aanmaken (of hebben).

Die-hard Linux-gebruikers kunnen natuurlijk ook in een terminal de GNU-C-compiler direct gebruiken.

Als je niet de beschikking hebt over een laptop, dan kan je ook online je programma's ontwikkelen. Zie hiervoor de site <http://www.onlinegdb.com/>. Alleen voor de afsluitende opdracht heb je een laptop nodig.

1.2 Introductie Code::Blocks IDE

Zie bijlage A.

1.3 Programma wijzigen

Wijzig de broncode van het programma dat je in de introductie hebt gemaakt als volgt:

```
/* My first program */
#include <stdio.h>

int main(void) {
    int a = 6, b = 10;
    printf("Het product van %d en %d is: %d\n", a, b, a + b);
    printf("\nSluit dit venster door op enter te drukken");
    getchar();
    return 0;
}
```

1.4 Vertalen van het gewijzigde programma

Voordat het gewijzigde programma kan worden uitgevoerd moet het programma opnieuw worden vertaald. Je kunt het programma vertalen door op de groene "play" knop te klikken.

Als je het bovenstaande programma juist hebt ingevoerd dan zal bij het vertalen een fout optreden. De foutmelding verschijnt onderin de ontwikkelomgeving in het window "Build log".

```
D:\PROJECTS\CodeBlocks\opdracht13\main.c:5:5: error: expected ',' or ';' before 'printf'
    printf("Het product van %d en %d is: %d\n", a, b, a + b);
    ^
```

Zie **jij** wat er fout is?

In de programmeertaal C moet elk “commando” worden afgesloten met een puntkomma en dat is vergeten op de regel voorafgaande aan de regel waarop de foutmelding wordt gegeven. Een fout die er voor zorgt dat een programma niet voldoet aan de regels van de programmeertaal wordt een *syntax fout* genoemd.

Corrigeer de fout en compileer (vertaal) het programma opnieuw.

Als je het bovenstaande programma juist hebt ingevoerd (en de syntax fout hebt gecorrigeerd) dan verschijnt een window met de volgende uitvoer:

```
Het product van 6 en 10 is: 16
```

Sluit dit venster door op enter te drukken.

Zie **jij** wat er fout is?

Een fout die er voor zorgt dat de uitvoer van een programma niet correct is wordt een *logische fout* genoemd. Logische fouten zijn moeilijker op te sporen dan syntax fouten omdat de compiler logische fouten niet kan vinden. In de praktijk komt het vaak voor dat een logische fout alleen maar onder bepaalde condities optreedt. Als je zelf een programma hebt geschreven dan ben je dus niet klaar als je programma zonder (syntax) fouten compileert. Je moet het programma daarna zelf nog goed testen om logische fouten te voorkomen!

Corrigeer de fout (er zijn twee mogelijkheden!) en test het programma opnieuw.

1.5 Programma met invoer.

Een integer variabele `a` kan als volgt vanaf het toetsenbord worden ingelezen:

```
scanf("%d", &a);
```

Het zogenoemde *format argument* `"%d"` geeft `scanf` de opdracht om een integer getal in te lezen. Het tweede argument `&a` geeft `scanf` de opdracht om de integer in de variabele `a` op te slaan. Dit wordt in de theorieles nog verder uitgelegd⁴.

Schrijf nu zelf een programma dat twee integers inleest (vanaf het toetsenbord) en de som van deze integers afdrukt.

Noot: er kleven nadelen van het inlezen van getallen met `scanf`. Zie bijlage B.

⁴ Variabele `a` wordt door de computer op een geheugenlocatie opgeslagen. Aan deze geheugenlocatie is een getal gekoppeld, het *adres*. Als `scanf` nu een getal inleest, dan moet dat getal in variabele `a` worden opgeslagen. De `&`-operator geeft het adres van de geheugenlocatie van `a` door aan functie `scanf`.

2 Toekennings- en herhalingsopdrachten (for loop)

2.1 Inleiding

In de opdrachten van dit hoofdstuk wordt gevraagd een programma te schrijven, waarin gebruik wordt gemaakt van lees- en schrijfinstructies, toekenningsopdrachten en een herhalingsopdracht m.b.v. een **for** loop.

Als voorbeeld is onderstaand programma gegeven. Hierin worden o.a. wiskundige functies gebruikt. Deze zijn opgenomen in een zogenoemde mathematische bibliotheek. Enkele voorbeelden van functies die in deze bibliotheek zijn gedefinieerd zijn: sinus, cosinus, de vierkantswortel en machtsverheffen. Om deze standaard functies in jouw programma te kunnen gebruiken, zal je in je programma bovenin de volgende *headerfile* moeten opnemen:

```
#include <math.h>
```

De wiskundige functies die in de mathematische bibliotheek voorkomen staan achterin het boek “De programmeertaal C, vierde editie” in bijlage A7 aangegeven. Je kunt ook online informatie over deze functies vinden op: <http://en.cppreference.com/w/c/numeric/math>

In onderstaand voorbeeldprogramma wordt aan de gebruiker gevraagd een positief getal x en een getal y in te voeren. Het programma berekent dan de vierkantswortel uit x en x tot de macht y . Hiervoor wordt gebruik gemaakt van de functies: `sqrt` en `pow`. De **for** loop zorgt ervoor dat vijf keer achter elkaar de vierkantswortel uit x en x tot de macht y wordt berekend. De eerste keer heeft x de waarde die de gebruiker heeft ingevoerd en de volgende vier keer wordt de waarde van x met 1 opgehoogd.

Maak een project aan en plaats daarin de onderstaande code. Laat zien dat het werkt.

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

int main(void) {
    double x, y, wortel, macht;
    int i;
    printf("Voer een positief getal x in: ");
    scanf("%lf", &x);
    printf("Voer een getal y in: ");
    scanf("%lf", &y);
    for (i = 0; i < 5; i = i + 1) {
        wortel = sqrt(x);
        macht = pow(x, y);
        printf("De vierkantswortel uit %f is: %f\n", x, wortel);
        printf("%f tot de macht %f is: %f\n", x, y, macht);
        x = x + 1;
    }
    fflush(stdin);
    getchar();
    return 0;
}
```



Let op! Je moet één van de onderstaande opdrachten maken. De practicumdocent bepaalt welke opdracht je moet maken.

2.2 Inwendig en uitwendig vermogen

Een spanningsbron kan worden voorgesteld door een ideale spanningsbron met de spanning U_b en een inwendige weerstand R_i . Aan deze bron wordt een uitwendige weerstand R_u geplaatst zodat er een stroom I_b gaat vloeien. Op het moment dat er een stroom vloeit, verbruiken de bron (met de inwendige weerstand) en de uitwendige weerstand vermogen.

Ontwerp een programma dat de gebruiker vraagt de waarden van U_b en R_i in te voeren. Het programma berekent voor de waarden $R_u = R_i$ t/m $R_u = 10 \cdot R_i$ steeds de stroom I_b en de vermogens verbruikt in R_i (de bron) en R_u . Zorg ervoor dat het programma goed omgaat met $R_i \leq 0$ door gebruik te maken van een **do while** loop.

Test het programma met $U_b = 5$ en $R_i = 10$. Bij $R_u = 100$ is $P_{R_i} = 20,66$ mW en $P_{R_u} = 206,61$ mW.

Bereken zelf minimaal één set met testwaardes.

2.3 Parallelweerstanden

De vervangingswaarde van parallel geschakelde weerstanden is uit te rekenen met de formule:

$$\frac{1}{R_p} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$$

waarbij R_p de vervangingswaarde van de parallel geschakelde weerstanden is en R_1, R_2, \dots, R_n de waarden van de parallel geschakelde weerstanden zijn.

Ontwerp een programma dat de gebruiker vraagt om het aantal weerstanden in te voeren (groter dan 0 natuurlijk). Daarna worden een voor een de weerstandswaarden ingevoerd. Vervolgens wordt de berekende vervangingswaarde afgedrukt. Zorg ervoor dat bij het invoeren van de gegevens een informatieve tekst wordt afgedrukt zodat de gebruiker weet wat hij/zij moet doen. Voorbeeld:

```
Geef aantal weerstanden op: 4
Geef waarde weerstand 1: 6
Geef waarde weerstand 2: 2
Geef waarde weerstand 3: 10
Geef waarde weerstand 4: 8
De vervangingswaarde is: 1.121 Ohm
```

2.4 Bazel-probleem

Schrijf een programma dat de uitkomst bepaalt van de onderstaande rij:

$$\sum_{i=1}^n \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2}$$

De uitkomst is dus de som van de inversen van de kwadraten van 1 tot en met n . Aan de gebruiker moet n worden gevraagd. Zorg ervoor dat het ingevoerde getal groter is dan 0 met behulp van een **do while**-lus. Gebruik een **for**-lus om de som te berekenen.

Het is te bewijzen dat als $n \rightarrow \infty$, de uitkomst $\pi^2/6 \approx 1,644934$ bedraagt. Zie <https://nl.wikipedia.org/wiki/Bazel-probleem>.

2.5 De spaarrekening

Wanneer je een geldbedrag op een spaarrekening hebt staan en je neemt de daarover uitgekeerde rente niet op, dan groeit het geldbedrag elk jaar met die rente.

Ontwerp een programma dat de gebruiker vraagt het beginbedrag dat op de spaarrekening staat en de rente in procenten die per jaar rente wordt uitgekeerd, in te voeren. Het bedrag staat 10 jaar op de spaarrekening. Het programma moet telkens het bedrag op de spaarrekening aan het eind van het lopende jaar berekenen, en dit bedrag samen met de rente van het betreffende jaar op het scherm afdrukken. Gebruik hiervoor een **for** loop. Zorg ervoor dat voor het beginbedrag en rente alleen positieve getallen kunnen worden ingevoerd door gebruik te maken van een **do while** loop.

Test het programma met: Beginbedrag = 1000 euro, rente = 5%

```
Geef beginbedrag: 1000
Geef rente in procenten: 5
jaar  rente  nieuw
-----
  1   50.00  1050.00
  2   52.50  1102.50
  3   55.13  1157.63
  4   57.88  1215.51
  5   60.78  1276.28
  6   63.81  1340.10
  7   67.00  1407.10
  8   70.36  1477.46
  9   73.87  1551.33
 10   77.57  1628.89
```

Bereken zelf minimaal nog één set met testwaardes.

2.6 De transformator

De spanningen en wikkelingen van de primaire en de secundaire zijde van een transformator verhouden zich als:

$$\frac{U_p}{U_s} = \frac{N_p}{N_s}$$

Hierin is:

- U_p = de spanning aan de primaire zijde;
- U_s = de spanning aan de secundaire zijde;
- N_p = het aantal wikkelingen aan de primaire zijde;
- N_s = het aantal wikkelingen aan de secundaire zijde.

Ontwerp een programma dat de gebruiker vraagt de primaire spanning (U_p), het aantal wikkelingen aan de primaire zijde (N_p) en de gewenste spanning aan de secundaire zijde (U_s) in te voeren. In het programma moet het aantal wikkelingen aan de primaire zijde (N_p), vanaf de ingelezen waarde, 10 keer achter elkaar met 100 ophogen. Het programma moet telkens voor elke afzonderlijke N_p -waarde, beginnend vanaf de ingelezen N_p -waarde, het aantal benodigde wikkelingen aan de secundaire zijde (N_s) berekenen en deze berekende waarde op het scherm afdrukken. Maak gebruik van een **for** loop.

Test het programma met:

```
Geef primaire spanning Up: 10
Geef primaire wikkelingen Np: 10
Geef secundaire spanning Us: 200
   Np    Ns
-----
   10    200
```

110	2200
210	4200
310	6200
410	8200
510	10200
610	12200
710	14200
810	16200
910	18200
1010	20200

Bereken zelf minimaal nog één set met testwaardes.

2.7 Faculteit

De faculteit van het getal n wordt genoteerd als $n!$ en is gedefinieerd als:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$$

dus voor 5! is dit:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

Dit is echter ook te schrijven als:

$$5! = 5 \cdot (4 \cdot (3 \cdot (2 \cdot 1)))$$

dus eerst $2 \cdot 1$ uitrekenen en daarna $3 \cdot (2 \cdot 1)$ etc.

Ontwerp een programma dat de gebruiker om het gehele getal n ($n \geq 0$) vraagt en de faculteit $n!$ uitrekent op basis van herhaald vermenigvuldigen. Zorg ervoor dat het programma ook tussentijdse berekeningen van $1!$, $2!$, ..., $(n - 1)!$ afdrukt. Zorg ervoor dat het programma goed omgaat met de invoer $n = 0$ en negatieve getallen door gebruik te maken van een **do while** loop.

Bereken zelf minimaal twee sets met testwaardes. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

2.8 Afgelegde weg

Een kleine stalen kogel wordt onder een hoek α weggeschoten met een beginsnelheid v_0 . De hoek ligt tussen 0° en 90° in lijn met het aardoppervlakte (de positieve x-as, zoals hoeken altijd getekend worden). De kogel wisselt geen energie uit met de omgeving, de beweging is dus wrijvingsloos. De kogel is natuurlijk ook onder invloed van de zwaartekracht. Dat houdt in dat de kogel een keer op de grond terecht komt. De hoogte en afgelegde weg als functie van de tijd kunnen beschreven worden met de functies:

$$h(t) = v_0 \cdot \sin(\alpha) \cdot t - \frac{1}{2} \cdot g \cdot t^2 \quad [\text{m}]$$

$$s(t) = v_0 \cdot \cos(\alpha) \cdot t \quad [\text{m}]$$

Hierbij is g de gravitatieversnelling, die is $9,81 \text{ [m/s}^2\text{]}$. Ontwerp een programma dat de gebruiker om de hoek α (in graden, $0^\circ \leq \alpha \leq 90^\circ$) en de beginsnelheid v_0 (in km/h, $v_0 > 0$) vraagt en voor de eerste 10 (gehele) seconden de hoogte en de afgelegde weg afdrukt.

Zorg ervoor dat het programma goed omgaat met de invoer $\alpha < 0^\circ$ en $\alpha > 90^\circ$ en negatieve waarde voor v_0 door gebruik te maken van een **do while** loop.

Test het programma met:

Geef de hoek in graden (0 - 90): 45
Geef de snelheid in km/h (> 0): 200
Hoek in radialen: 0.7853982
Snelheid in m/s: 55.556

Tijd	Hoogte [m]	Afstand [m]
0	0.00	0.00
1	34.38	39.28
2	58.95	78.57
3	73.71	117.85
4	78.65	157.13
5	73.79	196.42
6	59.12	235.70
7	34.64	274.99

Bereken zelf minimaal één set met testwaardes.

2.9 Benadering sinus

De sinus van een hoek wordt benaderd door:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

waarbij x in radialen moet worden opgegeven. De reeks heeft oneindig veel termen, maar een redelijke benadering is al te maken met slechts 5 termen.

Ontwerp een programma dat de gebruiker om een geheel getal n ($n \geq 0$) vraagt, dit getal stelt het aantal uit te rekenen termen voor. Daarna vraagt het programma om de hoek x (in radialen). Het programma rekent de sinus uit op basis van herhaald optellen. Maak gebruik een **for** loop. Zorg ervoor dat het programma goed omgaat met de invoer $n < 0$ door gebruik te maken van een **do while** loop.

Ontwerp een nieuw programma waarbij weer n en x worden gevraagd maar waarbij een lijst van uitgerekende sinuswaarden wordt weergegeven met het aantal termen tussen 0 en n . Hierbij kan je goed zien hoe de waarde van de sinus van een hoek wordt benaderd. Maak gebruik van een **for** loop.

Bereken zelf minimaal twee sets met testwaardes. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

2.10 Kerstboom

Een kerstboom is natuurlijk onmisbaar met kerstmis. Om het aantal verbruikte kerstbomen te minderen moet je een programma schrijven dat een kerstboom met '+'-tekens afdrukt. De gebruiker voert een getal n ($2 < n < 80$) in dat de breedte van de kerstboom voorstelt. Het afdrukken van de kerstboom begint met de top (in het midden) en dijt per afgedrukte regel met twee '+'-tekens uit. De kerstboom moet natuurlijk wel een stam en een 'voet' of standaard hebben. De afmetingen van de stam en de voet zijn (als je het netjes doet) ook afhankelijk van n . Zorg ervoor dat het programma goed omgaat met de invoer $n \leq 2$ en $n \geq 80$ door gebruik te maken van een **do while** loop.

2.11 Maandkalender

Schrijf een programma dat een maandkalender afdruckt. De gebruiker moet het aantal dagen in de maand en de dag van de week waarop de maand begint invoeren. Het aantal dagen in de maand kan minimaal 28 en maximaal 31 zijn. De dag van de week waarop de maand begint wordt ingevoerd als een geheel getal met een minimale waarde van 1 (zondag) en een maximale waarde van 7 (zaterdag). Zorg ervoor dat het programma goed omgaat met de invoer van ongeldige getallen.

Voorbeeld van het uitvoeren van dit programma:

```
Geef het aantal dagen in de maand: 41
Geef het aantal dagen in de maand: 31
Geef de dag van de week waarop de maand begint (1 = zondag ... 7 = zaterdag): 0
Geef de dag van de week waarop de maand begint (1 = zondag ... 7 = zaterdag): 3
```

```
zo ma di wo do vr za
=====
          1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Maak voor het afdrucken van de tabel gebruik van een **for** loop.

2.12 Wind-chill temperatuur

Bij het kouder worden van de dagen vragen veel mensen zich af hoe koud het buiten nu eigenlijk aanvoelt. De gevoelstemperatuur is afhankelijk van de omgevingstemperatuur (uit de zon, uit de wind) en de windsnelheid. De gevoelstemperatuur kan als volgt berekend worden:

$$T_{wc} = 13,12 + 0,6215 \cdot T_{omg} - 11,37 \cdot v^{0,16} + 0,3965 \cdot T_{omg} \cdot v^{0,16}$$

De omgevingstemperatuur T_{omg} is in graden Celsius, de windsnelheid v is in km/h. De formule geldt alleen voor een omgevingstemperatuur lager dan 10 graden Celsius en een windsnelheid groter dan 4,8 km/h.

Ontwerp een programma dat aan de gebruiker een temperatuur en een windsnelheid vraagt en in een tabel de wind-chill temperatuur afdruckt voor de opgegeven temperatuur en de eerste 9 opvolgende temperaturen (door steeds met 1 graad Celsius te verhogen). Dus als een gebruiker als temperatuur -15 opgeeft, moeten de temperaturen van -15 t/m -5 berekend worden.

Zorg ervoor dat de gebruiker alleen temperaturen en windsnelheden invoert die zijn toegestaan (zie eerder in deze opgave).

2.13 Benadering e

De wiskundige constante e wordt benaderd door:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \dots = \sum_{n=0}^{\infty} \frac{1}{n!}$$

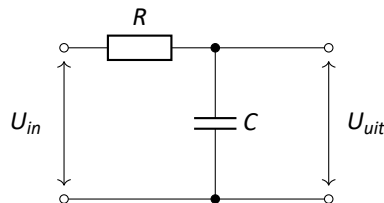
De reeks heeft oneindig veel termen, maar er is al een redelijke benadering te maken met 7 termen. Bij 7 termen is de fout namelijk kleiner dan 0,00005. Ontwerp een programma dat de gebruiker om een geheel getal n ($n > 0$) vraagt, dit getal stelt het aantal uit te rekenen termen voor. Het programma rekent de constante e uit op basis van herhaald optellen. Gebruik hiervoor een **for** loop. Zorg ervoor dat het programma goed omgaat met de invoer $n \leq 0$ door gebruik te maken van een **do while** loop.

Ontwerp een nieuw programma waarbij weer n wordt gevraagd maar waarbij een lijst van uitgerekende benaderingen van de constante e wordt weergegeven met het aantal termen tussen 1 en n . Hierbij kan je goed zien hoe de waarde van de constante e wordt benaderd. Maak gebruik van een **for** loop.

Controleer de werking van het programma met: $e = 2.718281828$

2.14 Opladen van een condensator

In de onderstaande figuur is een schakeling met een weerstand R en een condensator C te zien. Alle spanningen in de schakeling zijn in de beginsituatie 0 V.



Op een bepaald moment wordt op de ingang een constante spanning van 10 V aangeboden. De condensator zal hierdoor langzaam “vollopen” waardoor de spanning over de condensator stijgt. De snelheid waarmee dat gebeurt, is afhankelijk van de waarden van R en C en de tijd t . De uitgangsspanning U_{uit} als functie van de tijd kan als volgt berekend worden:

$$U_{uit}(t) = 10 \cdot (1 - e^{-t/RC}) \quad (\text{e is het grondtal van de natuurlijke logaritme, } e \approx 2,71828)$$

Schrijf een programma dat de uitgangsspanning U_{uit} berekend als functie van R , C en de tijd. Het programma moet de waarden van R en C inlezen (beide moeten groter zijn dan 0) en daarna moet het aantal te berekenen seconden worden ingevoerd (groter dan 0). Het programma berekent vanaf tijdstip 0 op iedere seconde de waarde van de uitgangsspanning. Hieronder zie je een mogelijke uitvoer van het programma.

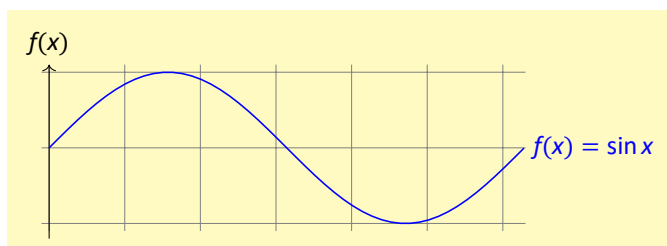
```
Geef de waarde van R (R > 0): 2.5
Geef de waarde van C (C > 0): 1.4
Geef het aantal seconden op (t > 0): 20
tijd spanning
```

```
-----
0  0.00
1  2.49
2  4.35
3  5.76
4  6.81
5  7.60
6  8.20
7  8.65
8  8.98
9  9.24
10 9.43
11 9.57
12 9.68
13 9.76
14 9.82
15 9.86
16 9.90
17 9.92
18 9.94
19 9.96
```

Merk op dat, naar mate de tijd toeneemt, de uitgangsspanning steeds dichterbij 10 V komt, maar nooit helemaal 10 V zal worden.

2.15 Booglengte van één periode van een sinus

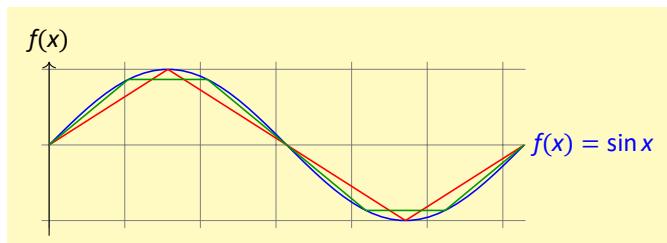
We willen graag de booglengte weten van één periode van een sinus. We gaan uit van de functie $f(x) = \sin x$. Eén periode van de sinus is getekend in de onderstaande figuur.



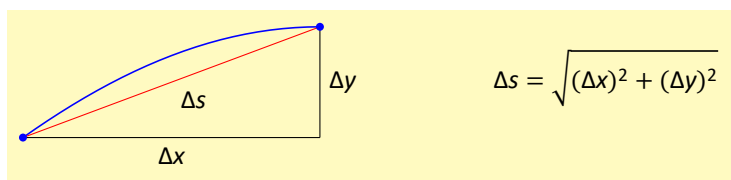
Nu is de booglengte s exact uit te rekenen met de integraal:

$$s = \int_0^{2\pi} \sqrt{1 + \cos^2 x} \, dx$$

Helaas is deze integraal niet analytisch oplosbaar. Er zit niets anders op om de lengte numeriek uit te rekenen. Daartoe hanteren we het volgende. We delen de sinus op (via de x-as) in een aantal stukken. In de onderstaande figuur is te zien dat de sinus in vier stukken is opgedeeld (rode lijn) en in zes stukken (groene lijn). De lengte van de rode en groene lijnen benaderen de booglengte van de sinus aardig.



De lengte van één stuk van de rode lijn is uit te rekenen met behulp van de stelling van Pythagoras, zoals te zien is in de onderstaande figuur. Hierbij staat het teken Δ voor “een stukje lengte”.



Als we nu alle lengtes van de lijnstukken optellen, benaderen we de booglengte van de sinus. Als we nu het aantal stukken steeds groter maken, benaderen we de booglengte van de sinus steeds beter.

Schrijf een programma dat de lengte van één periode van de sinus bepaalt. Het aantal stappen moet worden ingevoerd als een geheel getal n ($n > 1$).

3 Beslissingsopdrachten (if, switch)

3.1 Inleiding

In de opdrachten van dit hoofdstuk wordt gevraagd een programma te schrijven, waarin gebruik wordt gemaakt van inlees- en schrijfinstructies, toekenningsopdrachten, een herhalingsopdracht en één of meer beslissingsopdrachten. Er zijn drie verschillende beslissingsopdrachten: **if**, **if else** en **switch**. De **if** of **if else** kun je gebruiken bij een eenvoudige beslissing, bijvoorbeeld om te beslissen of een getal groter is dan 10. De **switch** gebruik je als je één variabele moet testen op verschillende mogelijke waarden. Hieronder zie je een voorbeeld van het gebruik van een **switch**.

Maak een project aan en plaats daarin de onderstaande code. Laat zien dat het werkt.

```
#include <stdio.h>

int main(void){
    int cijfer;
    char letter;
    do {
        printf("Geef je cijfer: ");
        scanf("%d", &cijfer);
    } while (cijfer < 0 || cijfer > 10);
    switch (cijfer) {
        case 10:
        case 9:
        case 8:
            letter = 'A'; break;
        case 7:
            letter = 'B'; break;
        case 6:
            letter = 'C'; break;
        case 5:
            letter = 'D'; break;
        default:
            letter = 'F'; break;
    }
    printf("Dit komt overeen met een %c.\n", letter);
    fflush(stdin);
    getchar();
    return 0;
}
```



Let op! Je moet één van de onderstaande opdrachten maken. De practicumdocent bepaalt welke opdracht je moet maken.

3.2 Even en oneven

Ontwerp een programma dat de gebruiker vraagt om positieve gehele getallen in te voeren. Dit programma moet net zo lang getallen inlezen totdat het getal 0 wordt ingevoerd. Hierna moet het programma afdrukken hoeveel even en hoeveel oneven getallen door de gebruiker zijn ingevoerd. Als de gebruiker negatieve getallen invoert dan moeten die niet worden meegeteld. Bedenk goed met welke invoerreeksen je dit programma moet testen om de correcte werking aan te tonen. Laat de invoerreeksen die je hebt gebruikt ook aan je practicumdocent zien.

Tip: Een even getal is deelbaar door 2 en een oneven getal is niet deelbaar door 2. Als een getal x deelbaar is door 2 dan is $x \% 2$ gelijk aan 0. Als een getal y niet deelbaar is door 2 dan is $y \% 2$ gelijk aan 1. Zie het boek voor uitleg over de $\%$ -operator.

Verzin zelf een aantal testreeksen. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.3 Voldoende en onvoldoende

Ontwerp een programma dat de gebruiker vraagt om positieve gehele getallen in te voeren met een maximum van 10. Dit programma moet net zo lang getallen inlezen totdat het getal 0 wordt ingevoerd. Hierna moet het programma afdrukken hoeveel voldoende (6...10) en hoeveel onvoldoendes (1...5) door de gebruiker zijn ingevoerd. Als de gebruiker negatieve getallen of getallen groter dan 10 invoert dan moeten die door het programma niet worden meegeteld. Bedenk goed met welke invoerreeksen je dit programma moet testen om de correcte werking aan te tonen. Laat de invoerreeksen die je hebt gebruikt ook aan je practicumdocent zien.

Verzin zelf een aantal testreeksen. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.4 Overwerk

Overwerk kan vervelend zijn maar in veel gevallen levert het extra salaris op. De nominale werkweek in Nederland is 40 uur. Daarboven wordt (vaak) gesproken over overwerk. Tussen 40 uur en 50 uur wordt gesproken van matig overwerk, boven de 50 uur wordt gesproken van veel overwerk.

Schrijf een programma dat het salaris van een werkweek bepaalt. Eerst wordt aan de gebruiker gevraagd wat het standaard tarief is van één uur werk. Alle uren tot 40 uur worden met dit tarief berekend. De uren die tussen 40 uur en 50 uur worden gewerkt leveren 1,5 keer het 40-uurs-tarief op. Uren boven de 50 uur leveren 2 maal het 40-uurs-tarief op. Zorg ervoor dat alleen een positief aantal uren en uurtarief kan worden opgegeven.

3.5 ABC-formule

Schrijf een programma dat de wortels van een kwadratische vergelijking bepaalt. Een kwadratische vergelijking heeft de gedaante:

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

De wortels van de functie, dat zijn de waarden van x waarvoor geldt dat het linkerlid 0 oplevert, zijn te berekenen met de ABC-formule:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{en} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

De term $b^2 - 4ac$ onder het wortelteken staat bekend als de *discriminant*. Deze discriminant moet een waarde hebben die groter is dan of gelijk is aan 0. Een negatieve waarde geeft aan dat er geen oplossingen zijn voor de wortels. Merk op dat als de discriminant 0 is, de waarden van x_1 en x_2 aan elkaar gelijk zijn.

Schrijf een programma dat de gebruiker om de waarden van a , b en c vraagt en de waarden van x_1 en x_2 berekent.

3.6 Simpele rekenmachine

Rekenen doen we al sinds mensenheugenis. Vooral in de elektrotechniek is rekenen onontbeerlijk. We gebruiken daarbij een rekenmachine met veel functies. Schrijf een programma dat een zeer eenvoudige rekenmachine implementeert. De rekenmachine kan slechts optellen, aftrekken, vermenigvuldigen en delen. Het programma vraagt de gebruiker om op één regel de twee getallen, gescheiden door een operator (*, +, - en /), op te geven. Voorbeeld:

Geef twee getallen gescheiden door een operator: $5.2 * 4.5$
De vermenigvuldiging van 5.2 en 4.5 is 23.4

Maak gebruik van het `switch`-statement.

3.7 Bepalen schrikkeljaar

Ontwerp een programma dat de gebruiker vraagt om datums in het formaat *dd mm yyyy* (dag gevolgd door maand gevolgd door jaartal, alle in cijfers) in te voeren. De datum moet opnieuw worden afgedrukt, maar nu moet de maand als woord worden afgedrukt. Daarnaast moet het programma afdrukken welke dag in het jaar het is en of het een schrikkeljaar is. De invoer `24 11 2011` levert dus op: `24 november 2011, dag 328 van het jaar, geen schrikkeljaar`. Er moet uiteraard getest worden of een gebruiker een geldige datum heeft ingevoerd. Het programma herhaalt het omzetten totdat de speciale datum `00 00 0000` ingevoerd wordt. Dan moet het programma stoppen. Maak gebruik van een `switch`-statement. Bedenk goed met welke datums je dit programma moet testen om de correcte werking aan te tonen. Laat de datums die je hebt gebruikt om te testen ook aan je practicumdocent zien.

Het bepalen of een bepaald jaar een schrikkeljaar is kan je vinden op <http://www.daniweb.com/software-development/c/threads/233325>

Vind zelf een aantal testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.8 Afdrukken binair getal

Het afdrukken van een binair getal is in standaard C niet mogelijk. Natuurlijk zijn er uitbreidingen in C-compilers, maar daar kan je niet van uitgaan. Ontwerp een programma dat een geheel getal tussen 0 en 255 omzet naar een 8-bits binair getal. Bedenk goed met welke getallen je dit programma moet testen om de correcte werking aan te tonen. Laat de getallen die je hebt gebruikt om te testen ook aan je practicumdocent zien.

Hint: om te kijken of het hoogste bit (bit 7; we nummeren van 0 t/m 7) een 0 of 1 is, kan je het getal vergelijken met 128, de hoogste macht in een 8-bit binair getal. Is het getal 128 of groter, dan is het meest significante bit 1, anders is het 0. Zo kan je alle bits afgaan. Gebruik een `for`-lus om de oplossing efficiënt te programmeren.

Zorg ervoor dat het aantal af te drukken bits eenvoudig gewijzigd kan worden. Neem in je code bijvoorbeeld de variabele `aantal_bits` op:

```
int aantal_bits = 16;
```

om een ingevoerd getal met 16 bits af te drukken.

Vind zelf een aantal testwaardes.

3.9 Windsnelheden

Varen op de wereldse zeeën is iets wat de Nederlanders al generaties doen. Om een goed beeld te hebben van de kracht van de wind is rond 1805 de zogenaamde Beaufort-schaal ingevoerd (Bf). Tegenwoordig werken we meer met snelheden in km/h. Toch wordt er nog veel met Beaufort gewerkt.

Ontwerp een programma dat de gebruiker vraagt om een windsnelheid in km/h (geheel getal). Het programma moet dan de windsnelheid in Beaufort afdrukken. Het programma moet dit steeds herhalen. Het programma stopt als het getal `-1` wordt ingevoerd als windsnelheid. Bedenk goed met welke getallen je dit programma moet testen om de correcte werking aan te tonen. Laat de getallen die je hebt gebruikt om te testen ook aan je practicumdocent zien.

Je kan een conversietabel vinden op <https://keesvdm.home.xs4all.nl/boot/windkracht.htm>

Hou de testwaarden bij de hand voor tijdens het nakijken van de opdracht.

3.10 Geschreven letters

Ontwerp een programma dat een geheel getal tussen 0 en 100 (inclusief) inleest en dit getal daarna als woord afdruckt. Als voorbeeld: de invoer 21 levert als uitvoer eenentwintig. Het programma blijft dit doen totdat het getal -1 wordt ingevoerd. Ontwerp dit programma zo efficiënt mogelijk. Bedenk goed met welke getallen je dit programma moet testen om de correcte werking aan te tonen. Laat de getallen die je hebt gebruikt om te testen ook aan je practicumdocent zien.

Test het programma met:

Geef getal tussen 0 en 100: 21
Het getal is: eenentwintig

Geef zelf ook een aantal testwaarden. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.11 Stijgen

Ontwerp een programma dat de gebruiker vraagt om positieve gehele getallen in te voeren. Dit programma moet net zo lang getallen inlezen totdat het getal 0 wordt ingevoerd. Op dat moment moet het programma afdrukken hoe vaak de invoer van de gebruiker “gestegen” is bij het invoeren. Als de gebruiker een getal invoert dat groter is dan zijn/haar vorige invoer dan tellen we dat als een stijging. Als de gebruiker negatieve getallen invoert dan moeten deze na het inlezen worden overgeslagen. Bedenk goed met welke invoerreeksen je dit programma moet testen om de correcte werking aan te tonen. Laat de invoerreeksen die je hebt gebruikt ook aan je practicumdocent zien.

Tip: in de volgende tabel zijn enkele invoerreeksen met de gewenste uitvoer gegeven. Test of je programma werkt door de onderstaande reeksen in te voeren.

Invoer	Correcte uitvoer	Verklaring
10 10 20 19 19 21 0	2	Bij het invoeren van 20 na 10 is er een stijging en bij het invoeren van 21 na 19 is er een stijging.
-1 10 -1 10 -1 -1 20 -2 19 19 21 0	2	Als we alle negatieve getallen overslaan dan is deze invoer hetzelfde als de vorige.

Verzin zelf een aantal testreeksen. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.12 Tekens tellen

Ontwerp een programma dat een reeks tekens inleest en de tekens classificeert als letters, cijfers en leestekens. Het aantal ingevoerde letters, cijfers en leestekens moeten worden afgedrukt. Bij letters moet nog onderscheid worden gemaakt tussen klinkers en medeklinkers. Bedenk goed met welke invoerreeksen je dit programma moet testen om de correcte werking aan te tonen. Laat de invoerreeksen die je hebt gebruikt ook aan je practicumdocent zien. Leestekens zijn te vinden op <http://nl.wikipedia.org/wiki/Leesteken>.

Tip: het testen van het type van een teken kan heel eenvoudig met de karakter classificatie functies uit de standaard C library, zie <http://en.wikipedia.org/wiki/Ctype.h>.

Verzin zelf een aantal testreeksen. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

3.13 Dag van de week

Ontwerp een programma dat de gebruiker vraagt om datums in het formaat *dd mm yyyy* (dag gevolgd door maand gevolgd door jaartal, alle in cijfers) in te voeren. Daarna moet het programma de dag van

de week voor deze datum bepalen. Er moet uiteraard getest worden of een gebruiker een geldige datum heeft ingevoerd. Het programma herhaalt het hiervoor beschreven proces totdat de speciale datum 00 00 0000 ingevoerd wordt. Dan moet het programma stoppen. Maak gebruik van een **switch**-statement. Bedenk goed met welke datums je dit programma moet testen om de correcte werking aan te tonen. Laat de datums die je hebt gebruikt om te testen ook aan je practicumdocent zien.

Een algoritme voor het vinden van de weekday van een datum is te vinden op https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week#Sakamoto's_methods

Zoek zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

4 Functies

4.1 Inleiding

Deze opdracht gaat over het gebruik van functies. Veel voorkomende handelingen, zoals het uitrekenen van verschillende waarde van formules of het inlezen van getallen kunnen hiermee efficiënt gecodeerd worden. Daarnaast wordt het gehele programma overzichtelijker. Bepaalde algoritmes, zoals het berekenen van een faculteit of het sorteren van een rij getallen, kunnen elegant worden opgelost door het gebruik van *recursieve functies* (functies die zichzelf aanroepen). De functies moeten grondig getest worden. Dit kan natuurlijk in hetzelfde programma als waar de functie zich in bevindt.

Na de laatste theorieles in week 3 heb je de huiswerkopdracht gekregen om een recursieve functie te schrijven die het n^e getal uit de *fibonacci-rij* berekent. Een mogelijke oplossing is:

```
#include <stdio.h>

int fib(int n) {
    if (n < 2) {
        return n;
    }
    else {
        return fib(n-1) + fib(n-2);
    }
}

int main(void) {
    int getal;

    do {
        printf("Geef een getal groter dan of gelijk aan 0: ");
        scanf("%d", &getal);
    } while (getal < 0);

    printf("fib(%d) is %d\n", getal, fib(getal));

    printf("Druk op enter om het programma af te sluiten.");
    fflush(stdin);
    getchar();
    return 0;
}
```

Maak een project aan en plaats daarin de bovenstaande code. Laat zien dat het werkt.

Let erop dat de functie `fib()` een *stopcriterium* moet hebben, d.w.z. dat het herhaald aanroepen van `fib` \hookrightarrow `()` (recursie) stopt als aan een bepaalde voorwaarde wordt voldaan. In het bovenstaande programma is dat als variabele n kleiner is dan twee. Vervang eens voor de grap de functie `fib()` met:

```
int fib(int n) {
    return fib(n-1) + fib(n-2);
}
```



Let op! Je moet één van de onderstaande opdrachten maken. De practicumdocent bepaalt welke opdracht je moet maken.

4.2 Catalan-getal (iteratief)

De Catalan-getallen zijn natuurlijke getallen die voorkomen in diverse telproblemen. Het n^e Catalan-getal wordt weergegeven als C_n . Bijvoorbeeld: het aantal manieren waarop je n paren haakjes correct kunt opschrijven is gelijk aan C_n . Zie voor verdere informatie: https://en.wikipedia.org/wiki/Catalan_number.

C_n kan berekend worden met behulp van de faculteitsfunctie:

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

- Implementeer een functie `catalan` die de waarde van C_n berekent met behulp van de bovenstaande formule. Maak daarbij gebruik van de in de les behandelde functie `faculteit` die $n!$ berekent.
- Schrijf een testprogramma om deze functie grondig te testen.
- Test tot welke waarde van n de functie `catalan` de waarde van C_n correct berekent.

Test het programma met:

```
0 geeft 1
5 geeft 42
7 geeft 429
9 geeft 4862
10 geeft 16796
```

4.3 Catalan-getal (recursief)

De Catalan-getallen zijn natuurlijke getallen die voorkomen in diverse telproblemen. Het n^e Catalan-getal wordt weergegeven als C_n . Bijvoorbeeld: het aantal manieren waarop je n paren haakjes correct kunt opschrijven is gelijk aan C_n . Zie voor verdere informatie: https://en.wikipedia.org/wiki/Catalan_number.

C_n kan ook recursief berekend worden:

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1} \text{ voor } n \geq 1 \text{ en } C_0 = 1$$

- Implementeer een functie `catalanRecursief` die de waarde van C_n berekent met behulp van de bovenstaande recursieve formule.
- Schrijf een testprogramma om deze functie grondig te testen.
- Test tot welke waarde van n de functie `catalanRecursief` de waarde van C_n correct berekent.

Test het programma met:

```
0 geeft 1
5 geeft 42
7 geeft 429
9 geeft 4862
10 geeft 16796
```

4.4 Vermoeden van Collatz

Het vermoeden van Collatz is één van de bekende onopgeloste vraagstukken in de getaltheorie. Laat n ($n > 0$) een willekeurig geheel getal zijn. Definieer een functie f_n als:

$$f_n = \begin{cases} n/2 & \text{als } n \text{ even is} \\ 3n + 1 & \text{als } n \text{ oneven is} \end{cases}$$

Het vermoeden van Collatz zegt dat als we deze formule *herhaaldelijk* toepassen uiteindelijk altijd de waarde 1 bereikt wordt voor elke beginwaarde van n . Zie voor verdere informatie: https://en.wikipedia.org/wiki/Collatz_conjecture.

- Implementeer een functie `int f(int n)` die de waarde van f_n berekent met behulp van de bovenstaande formule.
- Schrijf een testprogramma om deze functie grondig te testen.
- Ontwerp en implementeer nu een functie `int collatzSteps(int n)` die voor een bepaalde waarde van n bijhoudt hoe vaak de functie `f` aangeroepen wordt voordat de waarde 1 wordt bereikt.
- Schrijf een testprogramma om deze functie grondig te testen.

Bij grote waarden van n kan er een fout optreden bij het berekenen van `collatzSteps` omdat de waarde van f_n niet in een integer variabele past.

Bepaal tot welke waarde van n de functie `collatzSteps` het aantal stappen correct berekent.

Tip: Je kunt de functie `f` aanpassen zodat de waarde -1 wordt teruggegeven als het resultaat niet in een integer past. Bedenk zelf hoe je dit kunt detecteren als gegeven is dat (op een PC) de grootste waarde die in een integer past $2^{31} - 1$ is. Vervolgens kun je dan de functie `collatzSteps` aanpassen zodat de waarde -1 wordt teruggegeven als `f` de waarde -1 teruggeeft. Vervolgens kun je `collatzSteps` aanroepen voor opeenvolgende waarden van n totdat -1 wordt teruggegeven om te achterhalen tot welke waarde van n `collatzSteps` correct werkt.

Een mogelijke uitvoering van het programma zie je hieronder:

```
Geef geheel getal groter dan 1: 6
Stap: 1, n = 3
Stap: 2, n = 10
Stap: 3, n = 5
Stap: 4, n = 16
Stap: 5, n = 8
Stap: 6, n = 4
Stap: 7, n = 2
Stap: 8, n = 1
Aantal stappen voor 6 is 8
```

Bereken zelf ook een aantal testwaardes. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

4.5 Digitale techniek

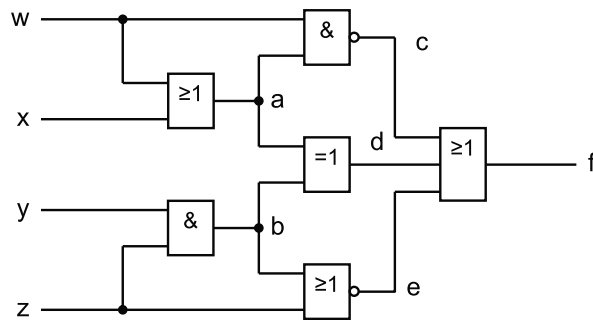
In de digitale wordt gebruik gemaakt van de bekende basispoorten AND, OR, NOT, NAND en NOR. Daarnaast worden ook de veel voorkomende poorten EXOR en EXNOR gebruikt, alhoewel dit geen basispoorten zijn.

Een zekere docent Digitale Techniek wil testen of zijn studenten een digitale schakeling met basispoorten correct kunnen doorrekenen, maar natuurlijk wil hij zelf niet teveel rekenwerk doen. Daarom wil hij een programma schrijven om dit doorrekenwerk te versnellen. Helaas heeft hij door de drukke werkzaamheden geen tijd. Zie figuur 1.

Ontwerp een programma dat, met gebruik van functies voor elke type poort, het hele schema doorrekent en de resultaten netjes als een waarheidstabel afdrukt.

Bedenk zelf handige namen voor de functies van AND, OR ... en zorg ervoor dat 2-input en 3-input (4-input?) poorten eenvoudig uit elkaar te houden zijn.

Zorg ervoor dat er zo efficiënt mogelijk met het opstellen van de functies wordt omgegaan. Bedenk bijvoorbeeld dat een NAND opgebouwd is uit een AND en een NOT. Het is dus niet nodig een volledige functie voor de NAND te maken, want je kunt gebruik maken van de eerder geschreven functies voor AND en NOT.



Figuur 1: Een digitale schakeling.

Alleen de functies voor OR2, AND2 en NOT moeten uitgewerkt worden, de rest van de functies zijn af te leiden uit de eerst genoemde functies.

4.6 Maximale

Het bepalen van een maximum uit een reeks getallen wordt vaak gebruikt. Denk hierbij aan het bepalen van de maximale temperatuur van de dag.

Ontwerp een programma dat aan de gebruiker een vijftal gehele getallen vraagt en het maximum van deze getallen afdrukt. Hiervoor moet een aantal functies geschreven en gebruikt worden:

- Als eerste moet er een functie worden geschreven, genaamd `max2`, die het maximum teruggeeft van twee gehele getallen. De getallen moeten als argumenten aan de functie worden meegegeven.
- Vervolgens moet er een functie `leesGetal` worden geschreven die een geheel getal inleest (netjes eerst tekst afdrukken dat een getal moet worden opgegeven), en dit ingelezen getal teruggeeft. Er mogen alleen gehele getallen worden ingelezen: letters en leestekens moeten worden genegeerd. Zie eventueel bijlage B.
- Natuurlijk kan er een functie worden geschreven die in één keer het maximum van vijf getallen bepaalt maar dat is niet nodig. Je kan het maximum van drie getallen namelijk óók bepalen met behulp van de `max2`-functie: één van de argumenten is dan het maximum van twee getallen! De `max2`-functie wordt dus gebruikt als argument in een (andere) `max2`-functie. Ontwerp een `max5`-functie door alleen maar `max2`-functies te gebruiken.

Verzin zelf meerdere testen. Zorg dat je deze testen bij de hand hebt tijdens het nakijken.

4.7 Gemiddelde

Schrijf een programma dat het gemiddelde van vijf getallen uitrekent.

Er moet een functie `leesGetal` worden geschreven die een geheel getal inleest (netjes eerst tekst afdrukken dat een getal moet worden opgegeven), en dit ingelezen getal teruggeeft. Er mogen alleen getallen worden ingelezen: letters en leestekens moeten worden genegeerd. Zie eventueel bijlage B.

Er moet een functie `gemiddelde` worden geschreven die het gemiddelde van de vijf getallen uitrekent en teruggeeft. Uiteraard moeten de vijf ingevoerde getallen als argumenten worden opgegeven.

Het hoofdprogramma moet de bovenstaande functies gebruiken.

Verzin zelf meerdere testen. Zorg dat je deze testen bij de hand hebt tijdens het nakijken.

4.8 Kleinste gemene veelvoud

Er zijn toepassingen waarbij de *kleinste gemene veelvoud* (kgv) van twee positieve getallen bepaald moet worden, bijvoorbeeld bij het gelijknamig maken van twee breuken. Het kgv van twee getallen is het klein-

ste positieve gehele getal dat een veelvoud is van beide getallen, dus het kleinste positieve gehele getal, waarvan beide getallen deler zijn.

Het kgv van twee getallen a en b kan berekend worden met de formule:

$$\text{kgv}(a, b) = \frac{a \cdot b}{\text{ggd}(a, b)} \quad \text{met } a, b > 0$$

waarbij $\text{ggd}(a, b)$ de *grootste gemene deler* is van a en b . Euclides heeft in 300 v.c. een algoritme beschreven (waarschijnlijk niet door hem bedacht) voor het vinden van de ggd. Het is gebaseerd op het recursief aanroepen van een functie:

$$\text{ggd}(a, b) = \begin{cases} a & \text{als } b = 0 \\ \text{ggd}(b, a \bmod b) & \text{als } b > 0 \end{cases}$$

Schrijf een functie `int kgv(int a, int b)` die de kgv van twee getallen bepaalt. Schrijf verder een functie `int ggd(int a, int b)` die de ggd van twee getallen bepaalt door gebruik te maken van de recursieve identiteit.

Schrijf een programma dat de getallen a en b , beide geheel een groter dan nul, inleest en de kgv berekent en afdruckt met behulp van de bovenstaande geformuleerde functies voor de kgv en ggd. Zorg ervoor dat bij het inlezen van de getallen voor a en b getest wordt of deze getallen groter zijn dan 0.

4.9 Wortel schieten

De Babyloniërs hebben heel lang geleden een methode ontwikkeld voor het vinden van de wortel van een positief getal (of nul, da's niet zo moeilijk). Het is gebaseerd op het herhaald doorrekenen van een eenvoudige functiebeschrijving:

$$\begin{aligned} X_0 &= S \\ x_{n+1} &= \frac{1}{2} \left(x_n + \frac{S}{x_n} \right) \\ \sqrt{S} &= \lim_{n \rightarrow \infty} x_n \end{aligned}$$

Hier staat het volgende:

- Als eerste benadering van de wortel (x_0) neem je het getal (S) waarvan je de wortel wil berekenen.
- Een tweede benadering van de wortel (x_1) kan worden gevonden door toepassing van de middelste formule (die met die $1/2$ erin) door voor n 0 in te vullen ($n + 1$ wordt dus 1).
- Een nieuwe benadering van de wortel kan worden gevonden door n steeds met 1 te verhogen en de middelste formule (die met die $1/2$ erin) herhaald toe te passen. Elke keer als je de middelste formule toepast kom je dichter bij de waarde van de wortel.
- Als je dit oneindig vaak doet, dan heb je de waarde van de wortel exact bepaald. Dat staat in de laatste formule.

Schrijf een programma dat de wortel van een getal groter dan of gelijk aan nul uitrekent. De middelste formule moet als functie worden geïmplementeerd. Er moet een functie `leesGetal` worden geschreven die het getal inleest (netjes eerst tekst afdrukken dat een getal moet worden opgegeven), en dit ingelezen getal teruggeeft. Er mogen alleen getallen worden ingelezen: letters en leestekens moeten worden genegeerd. Zie eventueel bijlage B.

Verzin zelf meerdere testen. Zorg dat je deze testen bij de hand hebt tijdens het nakijken.

4.10 Som van de cijfers en modulo 9

Gegeven is de functie leesCijfer die een integer cijfer inleest.

```
#include <stdio.h>

/* Dit programma demonstreert hoe een
   groot getal cijfer voor cijfer kan
   worden ingelezen */

int leesCijfer(void) {
    switch (getchar()) {
        case '0': return 0;
        case '1': return 1;
        case '2': return 2;
        case '3': return 3;
        case '4': return 4;
        case '5': return 5;
        case '6': return 6;
        case '7': return 7;
        case '8': return 8;
        case '9': return 9;
        default: return -1;
    }
}

int main(void) {
    int cijfer;
    printf("Type een groot getal:\n");
    do {
        cijfer = leesCijfer();
        if (cijfer != -1) {
            printf("%d ", cijfer);
        }
    } while (cijfer != -1);

    return 0;
}
```

Schrijf een functie leesEnBerekenSom die de som van de cijfers van een groot getal bepaalt.

BV. 2348967 -> 2+3+4+8+9+6+7 = 39

Gegeven is dat dit getal zo groot is dat het niet in een integer variabele past. Daarom wordt het getal in de functie leesEnBerekenSom cijfer voor cijfer ingelezen met behulp van de functie leesCijfer die gegeven is. Als de functie leesCijfer geen geldig cijfer in kan lezen dan geeft deze functie de waarde -1 terug.

Schrijf vervolgens de functie leesEnBerekenMod9 die een groot getal inleest en de modulo 9 van dit getal berekent. Gegeven is dat dit getal zo groot is dat het niet in een integer variabele past. De modulo 9 van een getal is gelijk aan de modulo 9 van de som van de cijfers van dat getal. Je kunt dus gebruik maken van de functie leesEnBerekenSom.

Schrijf ook een hoofdprogramma om de functies leesEnBerekenSom en leesEnBerekenMod9 te testen.

Test het programma met:

```
Geef (groot getal): 12345678901234567890
Som is: 90
Mod 9 is: 0
```

4.11 Isolated prime

Een priemgetal is een getal dat slechts twee delers heeft: het getal 1 en het getal zelf. Zo is het getal 11 een priemgetal want het is alleen deelbaar door 1 en door 11.

Een *isolated prime* is een priemgetal dat geflankeerd wordt door twee niet-priemgetallen. Dat wil zeggen dat als p een priemgetal is, dan zijn $p - 2$ en $p + 2$ dat niet. Neem het getal 23. Dit is een isolated prime, want 21 en 25 zijn geen priemgetallen.

Schrijf een programma dat een geheel getal vraagt (getal > 1 , want 0 en 1 zijn geen priemgetallen) en afdrukt of het opgegeven getal een isolated prime is. In het programma moet een functie `int isprime(int n)` komen die bepaalt of getal n een priemgetal is. Daarnaast moet er een functie `int isisolatedprime(int n)` die bepaalt of n een isolated prime is.

4.12 De Ackermann-functie

In de jaren '20 van de vorige eeuw is een functie bedacht door Wilhelm Ackermann. Het is een recursieve functie van twee variabelen m en n (beide ≥ 0). De functie is gedefinieerd als:

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Schrijf een programma dat de getallen m en n , beide geheel een groter dan of gelijk aan nul, inleest en met behulp van een functie de Ackermann-functie berekent en afdrukt met behulp van de bovenstaande recursieve identiteit.

4.13 sinc

In analoge en digitale filters komt veelvuldig de sinc-functie voor. De functie is gedefinieerd als:

$$\text{sinc } x = \frac{\sin x}{x}$$

Schrijf een functie `double sinc(double x)` die sinc-waarde bepaalt van het getal x . Druk de waarden af van x en $\text{sinc}(x)$ van 0,0 t/m 6,0 met een stapgrootte van 0,1.

4.14 Aantal delers van een geheel getal

In de getaltheorie komt een bepaalde klasse van getallen voor met de eigenschap dat zo'n getal alleen deelbaar is 1 en zichzelf. We noemen zo'n getal een *priemgetal*. Priemgetallen spelen een belangrijke rol bij encryptie.

Schrijf een functie `int aantal_delers(int getal)` die van het opgegeven getal het aantal delers bepaalt. Merk op dat 1 ook een deler is. Het getal moet groter zijn dan 1. Schrijf een hoofdprogramma dat de gebruiker vraagt een getal in te voeren (netjes een melding geven wat er gevraagd wordt en testen of het getal groter is dan 1) en het aantal delers afdrukt m.b.v. de functie `aantal_delers`.

5 Pointers en array's

5.1 Inleiding

In de les heb je het één en ander geleerd over pointers en array's. Een pointer is een variabele die als inhoud een adres bevat. Dit adres is dan het adres waar een andere variabele ligt opgeslagen. Zodoende wijst de pointer dus naar de variabele.

Een array is een lijst van variabelen van het zelfde type. Je kan een array indexeren. Dat houdt in dat je een *element* uit de lijst kan gebruiken in je bewerkingen. Hiervoor gebruik je blokhaken. Een korte introductie over array's en pointers is te vinden in bijlage C.

Hieronder is een programma dat een aantal waarden inleest in een array en vervolgens het gemiddelde van deze waarden afdrukt (het is in dit geval eigenlijk niet nodig om de getallen in een array op te slaan). Maak een project aan en plaats daarin de onderstaande code. Laat zien dat het werkt.

```
#include <stdio.h>

#define AANTAL 10

int main(void) {
    double temperatuur[AANTAL];
    int i;
    double temp_acc;

    for (i = 0; i < AANTAL; i = i + 1) {
        do {
            printf("Geef temperatuur %d op: ", i + 1);
            fflush(stdin);
        } while (scanf("%lf", &temperatuur[i]) != 1);
    }

    temp_acc = 0.0;
    for (i = 0; i < AANTAL; i = i + 1) {
        temp_acc = temp_acc + temperatuur[i];
    }

    printf("De gemiddelde temperatuur is %.2f\n\n", temp_acc / AANTAL);

    printf("Druk op enter om het programma af te sluiten.");
    fflush(stdin);
    getchar();
    return 0;
}
```



Let op! Je moet één van de onderstaande opdrachten maken. De practicumdocent bepaalt welke opdracht je moet maken.

5.2 Omkeren

C kent geen primitief type voor een string (primitieve types zijn o.a. `int` en `double`). Een string met karakters in C is niets anders dan een array van `char`'s, afgesloten met een nul-karakter (`'\0'`). Er zijn veel functies beschikbaar voor het manipuleren van strings, zoals het bepalen van de lengte of het kopiëren van strings. Informatie is te vinden op <https://en.wikipedia.org/wiki/String.h>.

Schrijf een programma dat een string inleest van het toetsenbord en deze string achterstevoren afdrukt. Hiervoor moet een functie `reverse` worden geschreven met als parameter de string. Deze functie zorgt

er voor dat de string achterstevoren wordt gezet. Het afdrukken gebeurt in het hoofdprogramma. Het gebruik van de functie `strrev` is niet toegestaan.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.3 Voortschrijden

Het bijhouden van de temperatuur wordt al jaren gedaan. Meestal wordt hierbij dan een gemiddelde temperatuur berekend. Dit is eenvoudig, zie opgave 5.1. Maar er is nog een soort gemiddelde: het voortschrijdend gemiddelde. Dit wordt bijvoorbeeld gebruikt bij beurskoersen⁵.

Schrijf een programma dat steeds het voortschrijdend gemiddelde bepaalt van de laatste 10 koersstanden. Na elke berekening moet een nieuwe koerswaarde (> 0) worden ingelezen. Gebruik een array om de koersstanden op te slaan.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.4 Histogram

Schrijf een programma dat de gehele getallen 0 t/m 10 inleest en bijhoudt hoe vaak elk getal is ingevoerd. Getallen kleiner dan -1 of groter dan 10 moeten worden genegeerd. Het getal -1 sluit de invoerprocedure af. Daarna drukt het programma de aantallen af in de vorm van een histogram⁶. Dit kan bijvoorbeeld elegant met (een rij van) sterretjes. Let erop dat de uitvoer geschaald moet worden; het getal dat het meest is ingevoerd moet met 30 sterretjes worden afgedrukt. Het afdrukken mag horizontaal of verticaal.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.5 Sorteren met “selection sort”

Schrijf een programma dat tien gehele getallen in een array inleest en deze getallen oplopend sorteert. Dat kan als volgt: zoek het minimum van alle getallen in de array en verwissel dit getal met het eerste getal in de array. Zoek vervolgens het minimum van de getallen met index 1 t/m 9 in de array en verwissel dit getal met het getal met de index 1 (het tweede getal) in de array. Zoek vervolgens het minimum van de getallen met index 2 t/m 9 in de array en verwissel dit getal met het getal met de index 2 in de array. Dit moet je herhalen totdat de gehele array gesorteerd is.

NB: Dit algoritme wordt *selection sort* genoemd, het is een leuke programmeeroefening maar een erg inefficiënte sorteermethode.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.6 Palindroom

C kent geen primitief type voor een string (primitieve types zijn o.a. `int` en `double`). Een string met karakters is in C niets anders dan een array van `char`'s, afgesloten met een nul-karakter (`'\0'`). Er zijn veel functies beschikbaar voor het manipuleren van strings, zoals het bepalen van de lengte of het kopiëren van strings. Informatie is te vinden op <https://en.wikipedia.org/wiki/String.h>.

Schrijf een programma dat een string inleest van het toetsenbord en bepaalt of deze string een palindroom⁷ is. Hiervoor moet een functie `is_palindroom` worden geschreven met als argument de string. De functie geeft een 1 terug als de string een palindroom is, anders geeft de functie een 0 terug. Het afdrukken gebeurt in het hoofdprogramma. Het gebruik van de functie `strrev` is niet toegestaan.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

⁵ Zie https://nl.wikipedia.org/wiki/Voortschrijdend_gemiddelde

⁶ Zie <https://en.wikipedia.org/wiki/Histogram>

⁷ Zie <http://nl.wikipedia.org/wiki/Palindroom>

5.7 Sorteren met “bubble sort”

Schrijf een programma dat tien gehele getallen in een array inleest en deze getallen oplopend sorteert. Dat kan als volgt: als het eerste getal in de array groter is dan het tweede getal, dan verwissel je die twee. Dit kan dan ook voor het tweede en derde getal etc. dus voor de hele array. Dit moet je herhalen totdat er geen verwisselingen meer nodig zijn, dan is de array gesorteerd.

NB: Dit algoritme wordt *bubble sort* genoemd, het is een leuke programmeeroefening maar een erg inefficiënte sorteer methode.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.8 Zoeken

Schrijf een programma dat een string inleest en bepaalt of dit woord in een gesorteerde lijst met 127 woorden voorkomt. Hiervoor moet je het ingevoerde woord vergelijken met de woorden in de lijst. Merk op dat je met maximaal zeven vergelijk-opdrachten kan bepalen of het ingevoerde woord in de gesorteerde lijst voorkomt of niet. Als je het woord wat je zoekt namelijk vergelijkt met het woord met index 63 in de lijst dan kun je het meteen vinden (als het woord dat je zoekt gelijk is aan het woord dat op index 63 staat) of weet je of het woord gezocht moet worden in de lijst van index 0 t/m 62 (als het woord dat je zoekt kleiner is dan het woord dat op index 63 staat) of dat het woord gezocht moet worden in de lijst van index 64 t/m 126 (als het woord dat je zoekt groter is dan het woord dat op index 63 staat). Op deze manier halveer je steeds het deel van de lijst dat je nog moet doorzoeken. Na 7 keer halveren heb je de gehele lijst van $127 = 2^7 - 1$ woorden doorzocht. Je programma mag dus niet meer dan zeven vergelijk-opdrachten uitvoeren. Een programma waarin een lijst met 127 gesorteerde meisjesnamen staat kun je vinden op: <http://bd.eduweb.hhs.nl/gesprg/progs/namenlijst.c>.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.9 Sinustabel

Het uitrekenen van een sinus van een hoek kost veel tijd. Slimmer is om een tabel aan te leggen van sinuswaarden bij verschillende hoeken. Nadeel is dat niet van elke willekeurige hoek de sinus kan worden opgevraagd.

Schrijf een programma dat een array vult met de sinuswaarden van 0 tot 90 in hele graden. Daarna vraagt het programma om een geheel getal tussen 0 en 360 en drukt de bijbehorende sinuswaarde met behulp van de tabel af. Dit doet het programma totdat -1 wordt ingevoerd.

Noot: deze oplossing wordt ook gebruikt bij het ontwerpen van digitale systemen. Het opzoeken van waarden is vele malen sneller dan het uitrekenen ervan.

5.10 Cesuur aanpassen

Bij sommige opleidingen wordt de cesuur van een toets aangepast zodat het percentage voldoende op of boven 70% ligt. Als blijkt dat een toets slecht is gemaakt dan moeten er punten worden bijgeteld zodanig dat het percentage voldoende op of boven 70% ligt. Is de toets goed gemaakt, dan moeten er geen punten worden bijgeteld.

Schrijf een programma dat een aantal resultaten (tussen 0,0 en 10,0) inleest. Dit aantal moet door de gebruiker worden opgegeven. Het minimum aantal is 10 en het maximum aantal is 100. Daarna moet bepaald worden of het aantal voldoende onder of boven de 70% ligt. Als het percentage gelijk is aan of groter is dan 70%, dan hoeft er geen bijtelling plaats te vinden. Als het percentage te laag is, dan moet er wel bijtelling plaats vinden. Dit kan door steeds 0,1 bij de resultaten op te tellen en daarna te bepalen of de 70%-grens al bereikt is. Let erop dat resultaten nooit hoger kunnen zijn dan 10,0.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.11 Lineaire regressie

In de statistiek wordt gebruikt gemaakt van lineaire regressie. Stel, je meet de waarden van spanning en stroom bij een bepaalde weerstand. In principe is dit een perfecte rechte lijn aangezien de wet van Ohm een lineair verband weergeeft tussen spanning en stroom. Door meetfouten en onnauwkeurigheden in de schakeling liggen de meetgegevens niet op een perfecte lijn, ze liggen “rond” of “in de buurt” van die lijn. Met behulp van statistiek kan je de rechte lijn vinden die het best past bij de meetgegevens.

De gebruiker moet een aantal x-y paren invoeren. Dit aantal wordt gevraagd aan de gebruiker en ligt tussen 2 en 100. Daarna moeten de x-y paren worden gevraagd. Vervolgens berekent het programma de rechte lijn die het best past bij de meetgegevens.

Een rechte lijn wordt gegeven door $y = \beta \cdot x + \alpha$. Om de parameters te vinden moet je de volgende berekeningen doen:

$$\beta = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$
$$\alpha = \bar{y} - \beta \bar{x}$$

Hierin zijn \bar{x} en \bar{y} de gemiddelden van de x-gegevens resp. de y-gegevens.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.12 Controle identificatiecode van Nederland eurobankbiljet

Schrijf een functie `int controleerNedEuroID(char id[])` die de identificatiecode van een Nederlands eurobankbiljet controleert. De parameter `id` is een C-string (een array van karakters afgesloten met een nul-karakter). De identificatiecode van een Nederlands eurobankbiljet bestaat uit de letter P gevolgd door 11 cijfers. De som van deze 11 cijfers modulo 9 moet 1 zijn. Zie eventueel: https://nl.wikipedia.org/wiki/Eurobankbiljetten#Nationale_identificatiecodes. Als de ingevoerde identificatiecode geldig is dan moet de functie `controleerNedEuroID` de waarde 1 teruggeven en anders moet deze functie de waarde 0 teruggeven.

Schrijf een hoofdprogramma om de functie `controleerNedEuroID` grondig te testen.

Test het programma met:

```
int res = controleerNedEuroID("P12345678901");
```

Na het uitvoeren moet de variabele `res` de waarde 0 hebben.

Bepaal zelf ook een aantal testwaardes.

5.13 Doorsnede

Schrijf een functie `int doorsnede(int a[], int aantal_a, int b[], int aantal_b)` die de *doorsnede*⁸ van de array's `a` en `b` in de array `a` plaatst. De parameter `aantal_a` bevat het aantal elementen in de array `a` en de parameter `aantal_b` bevat het aantal elementen in de array `b`. Je mag ervan uit gaan dat een integer hooguit eenmaal in elke array kan voorkomen. De functie moet het aantal getallen in de doorsnede teruggeven als return waarde.

Test de functie met het onderstaande hoofdprogramma:

⁸ Zie [https://nl.wikipedia.org/wiki/Doorsnede_\(verzamelingenleer\)](https://nl.wikipedia.org/wiki/Doorsnede_(verzamelingenleer))

```

int rij1[] = { 23, 3, 56, 6, 5 };
int rij2[] = { 12, 6, 3, 18, 7, 10, 56, 11 };
int aantal_doorsnede = doorsnede(rij1, sizeof rij1 / sizeof rij1[0], rij2,
    ↪ sizeof rij2 / sizeof rij2[0]);
int i;

for (i = 0; i < aantal_doorsnede; i = i + 1) {
    printf("%d ", rij1[i]);
}

```

moet de volgende uitvoer geven: 3 6 56

Bepaal zelf ook een aantal testwaardes.

5.14 Uniek

Schrijf een functie `int uniek(int a[], int aantal_a)` die alle getallen die meer dan één keer in de array `a` voorkomen verwijderd uit deze array. De parameter `aantal_a` bevat het aantal elementen in de array `a`. De functie moet het aantal unieke getallen in array `a` teruggeven als return waarde.

Schrijf ook een hoofdprogramma om de functie `uniek` grondig te testen.

Test het programma met:

De code:

```

int rij[] = { 23, 3, 56, 3, 23, 5, 12, 6, 3, 18, 7, 10, 56, 11 };
int aantal_uniek = uniek(rij, sizeof rij / sizeof rij[0]);
int i;

for (i = 0; i < aantal_uniek; i = i + 1) {
    printf("%d ", rij[i]);
}

```

kan de volgende uitvoer geven: 5 12 6 18 7 10 11

Bereken zelf ook een aantal testwaardes.

5.15 Run Length encoding

Run Length Encoding is een van de oudste compressiemethoden. Een reeks identieke, naast elkaar geplaatste karakters (een Run) wordt vervangen door het aantal karakters (Length) en het karakter.

Zo wordt de invoer van de string:

aaaabbbbbccccccdaaaa

gecodeerd als:

4a6b8c1d4a

Merk op dat de uitvoer groter wordt als de karakters steeds afwisselen. Zo wordt de string `abcd` gecodeerd als `1a1b1c1d`.

Schrijf een programma dat een string inleest en deze string RLE-gecodeerd in een andere string plaatst. Gebruik hiervoor de functie:

```
double rlecompress(char *invoerstring, char *rlestring).
```

Zorg ervoor dat de string `rlestring` groot genoeg is om de gecomprimeerde string op te slaan. De return-waarde is de compressieratio: de lengte van `rlestring` gedeeld door de lengte van `invoerstring`.

Verzin zelf testdata. Zorg dat je deze testdata bij de hand hebt tijdens het nakijken.

5.16 Rolling dice

Het gooien van dobbelstenen doen we al sinds mensenheugenis. Er zijn veel spelletjes waarbij twee dobbelstenen in één worp worden gegooid. Maar hoe vaak wordt nu 7 als het aantal ogen gegooid? En hoe vaak wordt nu 2 als het aantal ogen gegooid? Dat willen we uitzoeken.

Schrijf een programma dat de gebruiker vraagt hoe vaak de twee dobbelstenen gegooid moeten worden (`getal > 2`). Maak een functie `roll_dice()` die het gooien van één dobbelsteen simuleert. Met deze functie kan je het aantal gegooide ogen van twee dobbelstenen simuleren. Houd in een array bij hoe vaak een bepaald aantal ogen wordt gegooid. Druk aan het einde van de simulatie van alle mogelijke combinaties het gegooide aantal ogen af.

Bij deze opdracht moet je gebruik maken van de *pseudo random generator* in C. De term “pseudo” geeft aan dat het hier om een random generator gaat niet geheel random is (maar goed genoeg voor ons experiment). Om te beginnen moet je een *seed* zetten. Dat is het startpunt van onze reeks. Daarna kan je via de functie `rand()` een random getal opvragen. Zie voor meer informatie: http://www.gnu.org/software/libc/manual/html_node/ISO-Random.html#ISO-Random.

5.17 Overtollige spaties en tabs verwijderen

Bij het *parsen* (ontleden) van een string is het soms nodig om overtollige spaties en tabs te verwijderen. Met het verwijderen wordt bedoeld dat een reeks van aaneengesloten spaties en/of tabs worden vervangen door één spatie. Spaties aan het begin en einde van een string moeten in het geheel verwijderd worden. Dit wordt *trimming* genoemd. Hieronder zie je een voorbeeld waarbij spaties zijn vervangen door het `_`-symbool en tabs zijn vervangen door het `__`-symbool.

```
__Hallo, __dit__is__een__string.__
```

Deze string wordt omgezet in

```
Hallo,_dit_is_een_string.
```

Het verwijderen moet in de ingevoerde string gebeuren, er mag geen nieuwe string aangemaakt worden. Ga ervan uit dat de ingevoerde string niet langer is dan 100 karakters.

5.18 Vervangen

Voor het vervangen van een woord in een string is in C geen functie beschikbaar. Die zal je dus zelf moeten schrijven. Schrijf een functie:

```
int stringreplace(char *str, char *what, char *with)
```

die in de string `str` zoekt naar het woord `what` en vervangt door het woord `with`. De return-waarde is het aantal vervangingen en kan natuurlijk ook 0 zijn. Schrijf een hoofdprogramma om de functie te testen. Noot: let erop dat de lengte van `with` groter kan zijn dan de lengte van `what`. Zorg ervoor dat de ruimte van de string `str` groot genoeg is.

5.19 Onbeperkt

Het inlezen van een regel tekst van het toetsenbord is niet echt lastig. Het wordt pas problematisch als er meer tekens worden ingelezen dan er in de string kunnen worden ondergebracht. Meestal declareren we een string met 100 karakters en hopen maar dat er niet meer karakters worden ingevoerd.

We kunnen dit oplossen door on-the-fly dynamisch geheugen te alloceren. Een eenvoudig voorbeeld van dynamische geheugenallocatie is hieronder gegeven:

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

int main(void) {

    char *ptr = malloc(1024); /* Allocate 1024 bytes */

    if (ptr == NULL) {
        printf("Oops, no memory!\n");
        exit(-2);
    }

    printf("Allocated 1024 bytes at address %p\n", ptr);

    free(ptr); /* Free memory */

    return 0;
}
```

Merk op dat het stuk geheugen dat beschikbaar is gesteld niet is geïnitieerd (bijvoorbeeld met null-bytes).

We hebben nu een stuk geheugen van 1024 bytes. Dat kan op een gegeven moment vol zijn. Er moet dan een stuk geheugen worden gealloceerd dat groter is, bijvoorbeeld 2048 bytes. Dat kan met de functie `realloc()`:

```
char *ptrnew;

ptrnew = realloc(ptr, 2048);
```

Let erop dat `ptrnew` en `ptr` naar twee verschillende adressen *kunnen* wijzen. Als het eerste geheugengebied niet kan worden uitgebreid dan wordt er een nieuw geheugengebied gealloceerd. De functie `realloc()` kopieert dan de 1024 bytes van de eerste `alloc()` naar het nieuwe geheugengebied en geeft de pointer `ptr` vrij d.m.v. `free(ptr)`. Je kan meer vinden over `malloc` en `realloc` op <http://www.cplusplus.com/reference/cstdlib/malloc/>.

Schrijf een functie `char *getoneline(int chunksize)` dat een regel tekst van het toetsenbord inleest en in een stuk dynamisch gealloceerd geheugen plaatst. De parameter `chunksize` is het aantal bytes waarmee het geheugen moet worden uitgebreid als een eerder gealloceerd stuk geheugen vol is. Nadat de regel is ingelezen (end-of-line-karakter wordt niet opgeslagen) geeft de functie de pointer terug naar het gealloceerde geheugen. Test je functie met een klein programma en varieer de waarde van `chunksize`.

5.20 Waar of niet waar

Gegeven is de volgende bewering:

Deze zin bevat 1 keer een 0, 1 keer een 1, 1 keer een 2, 1 keer een 3, 1 keer een 4, 1 keer een 5, 1 keer een 6, 1 keer een 7, 1 keer een 8 en 1 keer een 9.

We bekijken **alleen** de cijfers en getallen. Natuurlijk is de bewering niet waar, er zijn immers 11 enen. Toch is er een zin mogelijk waarbij het aantal cijfers en de getallen kloppen. Schrijf een programma dat vanuit een beginsituatie (stel alle aantallen 1) iteratief naar een mogelijke oplossing zoekt.

Hint: maak gebruik van twee array's. Array `int a[10]` bevat het aantal cijfers in getalvorm, dus element `a[0]` bevat het aantal nullen in getalvorm. In array `int b[10]` houd je bij hoe vaak een cijfer voorkomt in de getallen van array a. Dus `b[1]` zou "10" kunnen bevatten omdat er in de getallen van array a 10 keer een 1 voorkomt. Ga ervan uit dat een cijfer niet meer dan 99 keer voorkomt. Bedenk ook dat het aantal voorkomingen van een cijfer minstens 1 is (kijk hierbij naar de zin).

6 Eindopdracht

6.1 Inleiding

Bij de eindopdracht gaan we gebruik maken van argumenten op de command-line en openen, lezen, schrijven en sluiten van bestanden op de disk. Een volledige introductie over deze onderwerpen is te vinden op BlackBoard. Hieronder wordt slechts het afdrucken van de argumenten op de command-line besproken.

Hieronder zie je een voorbeeld van een programma dat de opgegeven argumenten op het beeldscherm afdrukt. Het programma drukt eerst het aantal opgegeven argumenten af (via de variabele `argc`) en drukt daarna een voor een de argumenten af (via de variabele `argv`). Kopieer de code in een nieuw Code::Blocks-project en start het programma vanaf de command-line.

```
/* myprog.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    int i;

    printf("\nAantal argumenten: %d\n\n", argc);
    for (i = 0; i<argc; i++) {
        printf("Argument %d: %s\n", i, argv[i]);
    }
    return 0;
}
```

Als je dit programma draait vanuit een DOS-box⁹, dan zie je het volgende op het scherm.

```
D:\PROJECTS\CodeBlocks\myprog\bin\Debug>myprog.exe one two
```

```
Aantal argumenten: 3
```

```
Argument 0: myprog.exe
Argument 1: one
Argument 2: two
```

```
D:\PROJECTS\CodeBlocks\myprog\bin\Debug>
```



LEES DE VOETNOOT AAN HET EINDE VAN DIT HOOFDSTUK!.



Let op! Je moet één van de onderstaande opdrachten maken. De practicumdocent bepaalt welke opdracht je moet maken.

6.2 More

Schrijf een programma dat de inhoud van een tekstbestand met twintig regels tegelijk op het scherm laat zien. De naam van het tekstbestand wordt als argument op de command-line*) meegegeven. Na een druk op de enter-toets worden de volgende twintig regels getoond. Lever een tekstbestand mee van minimaal 100 regels. Bedenk dat een bestand niet altijd een veelvoud van twintig regels heeft.

⁹ Zie https://en.wikipedia.org/wiki/Win32_console

6.3 Cat

Schrijf een programma dat de inhoud van een aantal tekstbestanden achter elkaar op het scherm afdrukt. De namen van de tekstbestanden worden als argumenten op de command-line*) meegegeven. Het aantal bestandsnamen kan variëren, er wordt dus geen vast aantal namen opgegeven. Als geen bestandsnaam wordt opgegeven, moet het toetsenbord als invoer dienen (er kunnen dan geen bestanden geopend worden want er zijn geen namen opgegeven). Als het eerste argument `-h` is, moet er een helptekst worden afgedrukt en moet het programma stoppen. Er worden dan geen bestanden afgedrukt.

6.4 Zinslengte sorteren

Schrijf een programma dat de regels in een tekstbestand inleest en op lengte sorteert. Het resultaat schrijf je naar het scherm en naar een uitvoerbestand. De namen van de tekstbestanden moeten als argumenten op de command-line*) meegegeven worden.

6.5 Op alfabet sorteren

Schrijf een programma dat de woorden in een tekstbestand op alfabet sorteert. Het resultaat schrijf je naar het scherm en naar een uitvoerbestand. De namen van de tekstbestanden moeten als argumenten op de command-line*) meegegeven worden.

6.6 Regelnummering

Schrijf een programma dat een tekstbestand inleest (bijv. een C-file) en hier regelnummers aan toevoegt en naar een uitvoertekstbestand schrijft. Lege regels worden niet genummerd, maar worden wel naar het uitvoerbestand geschreven. De regelnummers komen aan de linkerkant te staan. Gebruik een invoerbestand van minimaal 100 regels tekst en lever dit bestand ook in. De namen van de bestanden moeten als argumenten op de command-line*) meegegeven worden.

6.7 Getallen sorteren

Schrijf een programma dat twee tekstbestanden met ongesorteerde integer getallen samenvoegt tot een derde gesorteerd bestand met integers. Alle drie bestandsnamen worden als argumenten op de command-line*) meegegeven.

6.8 Uitgeschreven nummers sorteren

In een tekstbestand staat een willekeurige rij met nummers van nul tot en met twintig. De nummers staan echter niet op volgorde en zijn languit (dus als woorden) geschreven. Sorteert alle woorden (nummers) op numerieke volgorde van hoog naar laag en schrijf het resultaat (aantal keer) naar een nieuw bestand. Het resulterende bestand kan er dus zo uitzien:

```
9 Twintig
15 Negentien
3 Achttien
6 Zeventien
11 Zestien
8 Vijftien
```

De namen van de tekstbestanden moeten als argumenten op de command-line*) meegegeven worden.

6.9 C keywords

Schrijf een programma dat een tekstbestand met C-programmacode inleest. Het programma moet vervolgens tellen hoe vaak een keyword van de programmeertaal C voorkomt. Vervolgens moet het resultaat

weggeschreven worden naar een ander bestand. De namen van de bestanden moeten als argumenten op de command-line*) meegegeven worden. Het resulterende bestand kan er dus zo uitzien:

```
2 double
12 for
5 if
8 int
6 while
```

6.10 Commentaar verwijderen

Schrijf een programma dat een tekstbestand met C-programmacode inleest en de programmacode zonder commentaren naar een ander bestand schrijft. Filter zowel blokcommentaren `/*...*/` als lijncommentaren `//...` uit. De namen van de bestanden moeten als argumenten op de command-line*) meegegeven worden.

6.11 Letters tellen

Schrijf een programma dat een tekstbestand inleest en bepaalt hoe vaak elke letter voorkomt. Lege regels, cijfers en leestekens mogen worden genegeerd. Hoofdletters en bijbehorende kleine letters moeten als identiek beschouwd worden. De resultaten moeten naast absolute aantallen ook als percentages van het totaal aantal letters in een tekstbestand worden weggeschreven. De namen van de tekstbestanden moeten als argumenten*) op de command-line meegegeven worden. Maak je code efficiënt door gebruik te maken van array's.

6.12 XeT

Schrijf een programma dat een tekstbestand inleest. In het tekstbestand zitten speciale commando's verborgen. Deze zijn:

- Als het commando `\spaces<getal>` wordt gevonden, dan moet dit commando vervangen worden door een aantal spaties dat is opgegeven met het getal `<getal>`. De tekens `<` en `>` zijn geen onderdeel van het commando. Merk op dat tussen `\spaces` en het getal geen spaties staan. Na het getal kan een spatie staan. Als er geen getal wordt gevonden, moet een foutmelding op het scherm volgen.
- Als het commando `\lines<getal>` wordt gevonden, dan moet dit commando vervangen worden door een aantal lege regels dat is opgegeven met het getal `<getal>`. De tekens `<` en `>` zijn geen onderdeel van het commando. Merk op dat tussen `\lines` en het getal geen spaties staan. Na het getal kan een spatie staan. Als er geen getal wordt gevonden, moet een foutmelding op het scherm volgen.
- Als het commando `\nobreak` wordt gevonden, dan moet het end-of-line-karakter aan het einde van de huidige ingelezen regel verwijderd worden. Het end-of-line-karakter wordt dus niet afgedrukt. Merk op dat na `\nobreak` altijd een spatie of een end-of-line-karakter moet volgen anders is het commando `\nobreak` niet te detecteren.

De aangepaste tekst moet naar een bestand geschreven worden. Eventuele foutmeldingen moeten naar het scherm worden geschreven. Ga ervan uit dat een ingelezen regel uit niet meer dan 100 karakters bestaat.

*) Argumenten op de command-line

Met argumenten op de command-line wordt het volgende bedoeld:

- Open een command-box (Windows: Windowstoets-R en type `cmd.exe`, OS-X: start terminal; Linux: start terminal).

- Navigeer naar de map waar de *executable* (= uitvoerbaar programma) staat. Bij Code:Blocks iets van C:\Users\...\Debug\bin. Bij Linux/Mac: effe uitzoeken.
- Start de executable met naam van executable.exe filename1.txt filename2.txt <enter> (of zoveel argumenten als noodzakelijk).
- Het programma start nu met de twee argumenten.

Het is dus **niet** de bedoeling dat na het starten van het programma de gebruiker om twee bestandsnamen wordt gevraagd of dat de bestandsnamen hard gecodeerd zijn in het programma. Verder moet getest worden of de bestanden wel geopend kunnen worden. Zo niet, dan moet dat aan de gebruiker worden medegedeeld. Als een gebruiker je programma start met teveel of te weinig argumenten, dan moet je dat aan de gebruiker mededelen. Ga er niet vanuit dat de gebruiker de inhoud van de bestanden zo formatteert dat het direct in je programma werkt. Je moet dus de ingelezen data wel controleren of het voldoet aan het formaat dat je kan verwerken (denk bijvoorbeeld aan letters i.p.v. getallen). Maak je programma flexibel. Dus niet voorschrijven dat je precies 10 getallen moet invoeren (maar om het eenvoudig te houden: niet meer dan 1000) of dat een regel precies 20 karakters moet hebben. Bedenk ook wat er moet gebeuren als een invoerbestand leeg is.

Zie bijlage [D](#) voor meer informatie en voorbeelden.

A Installatie Code::Blocks

In deze bijlage zijn instructies te vinden voor het installeren en gebruiken van Code::Blocks 17.12. De instructies zijn als volgt:

- Download Code::Blocks van <http://sourceforge.net/projects/codeblocks/files/Binaries/17.12/Windows/codeblocks-17.12mingw-setup.exe> (let erop dat in de naam de term mingw staat)
- Installeer vervolgens Code::Blocks door de gedownloade executable te runnen. (Bij installatie continu op next drukken)
- Bij het opstarten van Code::Blocks wordt er gevraagd naar de compiler die gebruikt zal gaan worden. Als het goed is, is er al één geselecteerd en hoeft je alleen op OK te drukken.
- Nu kun je een project aanmaken en code gaan compileren en runnen:
 - a. File->New->Project
 - b. Selecteer Console application + GO
 - c. Next
 - d. Selecteer C (!!!NIET!!! C++)
 - e. Project Title = helloworld
 - f. Folder to create project in: Selecteer een map die je later nog terug kan vinden...
 - g. Next
 - h. Finish (vinkjes aan, GNU GCC compiler geselecteerd)
 - i. Links in de projectomgeving is nu de file main.c te vinden met code:

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

int main(void)
{
    printf("Hello world\n");
    return 0;
}
```

- j. Klik op Build (Ctrl+F9) (het tandwiel symbool bovenaan)
- k. Klik op Run (Ctrl+F10) (het groene play symbool bovenaan)

B Inlezen van een getal met scanf

De functie `scanf` is een veelzijdige functie voor het inlezen van variabelen. Zo kan je een getal, een letter of een string inlezen. Maar hoe weet `scanf` nu wat er ingelezen moet worden? Dat geef je op met een *format string*. Een enkel geheel getal kan je inlezen met:

```
scanf("%d", &getal);
```

De format string `"%d"` vertelt `scanf` dat een geheel getal moet worden ingelezen. Het is ook mogelijk om meerdere getallen in te lezen:

```
scanf("%d %d %d", &getala, &getalb, &getalc);
```

Bij het invoeren moeten de getallen gescheiden worden door één of meerdere spaties. Het aantal spaties maakt dus niet uit; deze worden overgeslagen. Als je in C programma een getal inleest met `scanf` dan gebeuren er vreemde dingen als je in plaats van een getal letters intypt. Bijvoorbeeld:

```
#include <stdio.h>

int main(void) {
    int getal = -12345;
    printf("Geef een geheel getal: ");
    scanf("%d", &getal);
    printf("Het getal is %d.\n", getal);
    printf("Druk op de Enter toets om dit window te sluiten.");
    getchar();
    return 0;
}
```

Als je dit programma uitvoert en als invoer het woord `Hallo` intypt dan verschijnt de volgende uitvoer:

```
Geef een geheel getal: Hallo
Het getal is -12345.
Druk op de Enter toets om dit window te sluiten.
```

Het blijkt dat de waarde van variabele `getal` niet is veranderd. Het eerste teken dat `scanf` tegenkomt is de `H` en dat is geen cijfer, dus wordt er gestopt met inlezen van het toetsenbord. Het wordt echt problematisch als er een getal ingelezen moet worden dat aan een voorwaarde moet voldoen:

```
int main(void) {
    int getal = -12345;

    do {
        printf("Geef een geheel getal groter dan 0: ");
        scanf("%d", &getal);
    } while (getal < 1);

    printf("Het getal is %d.\n", getal);
    printf("Druk op de Enter toets om dit window te sluiten.");
    getchar();
    return 0;
}
```

De uitvoer is nu:

```
Geef een geheel getal groter dan 0: Hallo
Geef een geheel getal groter dan 0: Geef een geheel getal groter dan 0: Geef een
geheel getal groter dan 0: Geef een geheel getal groter dan 0: Geef een geheel
getal groter dan 0: Geef een geheel getal groter dan 0: Geef een geheel getal gr
oter dan 0: Geef een geheel getal groter dan 0: Geef een geheel getal groter dan
0: Geef een geheel getal groter dan 0: Geef een geheel getal groter dan 0: Geef
een geheel getal groter dan 0: Geef een geheel getal groter dan 0: Geef een geh
eel getal groter dan 0: Geef een geheel getal groter dan 0: Geef een geheel geta
enzovoorts
```

Het programma gaat nu als een razende te werk en de enige manier om het te stoppen is het programma af te sluiten. Maar hoe komt dat nou?

Als `scanf` voor de eerste keer iets inleest dan zijn er nog geen karakters vanaf het toetsenbord ingevoerd. Daarom vraagt `scanf` aan het operating system (Windows, Linux, OS-X) om een karakter. Het operating system weet dat er geen karakters beschikbaar zijn en gaat via interne routines karakters opvragen. Er wordt echter niet één karakter opgevraagd maar een hele reeks die afgesloten moet worden met een <enter>-toets. Je moet als gebruiker dus je invoer altijd afsluiten met een <enter>-toets, ook als je via `scanf` maar één karakter inleest. De ingelezen karakters worden ergens in het geheugen opgeslagen. Dit wordt *buffering* genoemd en de geheugenruimte wordt *invoerbuffer* genoemd. Er zijn nu karakters beschikbaar en het eerste ingevoerde karakter wordt aan `scanf` gegeven.

Het eerste karakter is de letter H en dat is geen cijfers. Dus stopt `scanf` direct met het inlezen van karakters (die dus alleen cijfers mogen zijn). Het karakter H blijft hierbij in de invoerbuffer staan. De H wordt dus niet verwijderd.

Omdat `scanf` geen cijfers heeft kunnen inlezen, wordt de opgegeven variabele niet veranderd en blijft zijn originele waarde behouden. Daarom drukt het eerste programma in deze bijlage het getal `-12345` af. Het tweede programma blijft in een `do while`-lus steeds een getal inlezen als het getal kleiner is dan `-1`. Aangezien `scanf` geen cijfers inleest en de opgegeven variabele niet aanpast, blijft de waarde `-12345` behouden en dat is kleiner dan `-1`. Dus wordt de lus nog een keer uitgevoerd. De H staat nog steeds in de invoerbuffer en lukt het `scanf` niet om een getal in te lezen.

Dit kun je vrij eenvoudig voorkomen door de returnwaarde van de functie `scanf` te testen. Deze functie geeft het aantal variabelen terug dat succesvol is geconverteerd en ingelezen (dat kan dus ook 0 zijn), anders geeft de functie de integer waarde EOF terug. Je kan dat als volgt doen:

```
#include <stdio.h>

int main(void) {
    int getal = -12345;
    int ret;

    printf("Geef een geheel getal groter dan 0: ");
    ret = scanf("%d", &getal);

    if (ret == 1) {
        printf("Het getal is %d.\n", getal);
    } else {
        printf("Geen getal ingevoerd!\n");
    }

    printf("Druk op de Enter toets om dit window te sluiten.");
    getchar();
    return 0;
}
```

De uitvoer wordt dan:

```
Geef een geheel getal groter dan 0: Hallo
Geen getal ingevoerd!
Druk op de Enter toets om dit window te sluiten.
```

Er is nog één probleem. De karakters van Hallo staan nog steeds in de invoerbuffer. Die zullen we er een voor een uit moeten halen. Dat kan met het onderstaande programma:

```
#include <stdio.h>

int main(void) {
    int getal, ret;
    do {
        printf("Geef een geheel getal: ");
        ret=scanf("%d", &getal);
        if (ret == 0) {
            printf("Dat was geen getal!\n");
            printf("Maar het karakter %c.\n", getchar());
        }
        else if (ret == EOF) {
            printf("Er is een fout opgetreden bij het lezen!\n");
        }
    } while (ret != 1);
    printf("Het getal is %d.\n", getal);
    printf("Druk op de Enter toets om dit window te sluiten.");
    getchar();
    return 0;
}
```

De functie getchar leest één karakter uit de invoerbuffer. Daarna proberen we scanf opnieuw. Dat mislukt telkens als er een letter gevonden wordt. Als je nu Hallo invoert, krijg je de volgende uitvoer:

```
Geef een geheel getal: Hallo
Dat was geen getal!
Maar het karakter H.
Geef een geheel getal: Dat was geen getal!
Maar het karakter a.
Geef een geheel getal: Dat was geen getal!
Maar het karakter l.
Geef een geheel getal: Dat was geen getal!
Maar het karakter l.
Geef een geheel getal: Dat was geen getal!
Maar het karakter o.
Geef een geheel getal:
```

Natuurlijk willen niet steeds dat bij elk karakter een melding op het scherm wordt afgedrukt. We kunnen een aantal printf-regels verwijderen maar niet de regel waarin de gebruiker wordt gevraagd om een geheel getal in te voeren. Anders weet de gebruiker niet wat hij/zij moet doen. Helaas wordt deze regel herhaaldelijk afgedrukt. We zullen op een iets andere manier de karakters moeten inlezen.

Gelukkig zorgt het operating system ervoor dat ook het end-of-line-karakter in de invoerbuffer terecht komt. We kunnen dus hierop testen. Hoe dat moet, is te zien in de onderstaande code:

```
do { ch = getchar(); } while (ch != '\n' && ch != EOF);
```

We lezen een karakter van de invoerbuffer en dat doen we zolang dat karakter ongelijk is aan `\n` (end-of-line-karakter) en ongelijk is aan EOF (end-of-file).

Omdat we vaker gebruik willen maken van het legen van de invoerbuffer plaatsen we de code een functie. We kunnen nu aan de gebruiker vragen om een getal in te voeren:

```
#include <stdio.h>

void purge_stdin(void) {
    int ch;
    do {
        ch = getchar();
    } while (ch != '\n' && ch != EOF);
}

int main(void) {
    int getal, ret;
    do {
        printf("Geef een geheel getal: ");
        ret=scanf("%d", &getal);
        if (ret == 0) {
            purge_stdin();
        }
    } while (ret != 1);
    printf("Het getal is %d.\n", getal);
    printf("Druk op de Enter toets om dit window te sluiten.");
    purge_stdin();
    getchar();
    return 0;
}
```

Er is nog een andere manier om de invoerbuffer te legen. We kunnen de invoerbuffer legen met de functie `fflush` (file flush):

```
fflush(stdin);
```

Als parameter van `fflush` wordt `stdin` opgegeven. Dat staat voor *standard input* en daar wordt in de regel het toetsenbord mee bedoeld¹⁰. Daarnaast kennen we nog `stdout` (*standard output*, het beeldscherm) en `stderr` (*standard error*, meestal ook het beeldscherm).

Deze manier werkt echter niet met alle C-compilers en operating systems. Dat heeft te maken met de definitie van de functie `fflush`. De functie `fflush` is bedoeld om uitvoerbuffers te legen. Als de uitvoer naar het beeldscherm is, wordt de buffer naar het beeldscherm geschreven. Als de uitvoer naar een bestand is, wordt de buffer naar het bestand geschreven. De C-standaard schrijft echter alleen voor dat *flushen* van een uitvoerbuffer gedefinieerd is, niet van een invoerbuffer. Niet zo gek eigenlijk, want wat wordt er nou bedoeld met flushen van de invoerbuffer. Flushen van het toetsenbord is het nog te begrijpen maar flushen van een invoerbestand niet. Moeten we dan helemaal tot einde van het bestand flushen? Of alleen maar de bijbehorende buffer? Dat levert een onvoorspelbaar programma op want we weten immers niet hoeveel karakters in de buffer staan.

Toch zijn er wel C-implementaties die het flushen van een invoerbuffer uitvoeren, bijvoorbeeld de GNU C-compiler op Linux en MinGW op Windows (zit o.a. in Code::Blocks).

Hieronder is de code te vinden van het flushen van de invoerbuffer van het toetsenbord m.b.v. `fflush`:

¹⁰ Op de bekende operating systems is het mogelijk om de inhoud van bestanden door te geven aan de *standard input*. Het programma krijgt dan data uit een bestand i.p.v. het toetsenbord. Dat wordt *redirection* genoemd.

```
/* Please note: might not works on all Operating Systems. */
#include <stdio.h>

int main(void) {
    int getal, ret;
    do {
        printf("Geef een geheel getal: ");
        ret=scanf("%d", &getal);
        if (ret == 0) {
            fflush(stdin);
        }
    } while (ret != 1);
    printf("Het getal is %d.\n", getal);
    printf("Druk op de Enter toets om dit window te sluiten.");
    fflush(stdin);
    getchar();
    return 0;
}
```

C Introductie array's en pointers

Soms is het gebruik van elementaire datatypen niet handig bij het ontwerpen van een programma. We nemen als voorbeeld een programma dat vijf getallen inleest en het gemiddelde van deze getallen afdrukt. We gebruiken vijf integer variabelen voor de in te voeren getallen en een integer variabele voor het berekenen van het gemiddelde:

```
#include <stdio.h>

int main(void) {

    int a, b, c, d, e;
    printf("Geef getal: ");
    scanf("%d", &a);
    printf("Geef getal: ");
    scanf("%d", &b);
    printf("Geef getal: ");
    scanf("%d", &c);
    printf("Geef getal: ");
    scanf("%d", &d);
    printf("Geef getal: ");
    scanf("%d", &e);

    gem = (a+b+c+d+e)/5;

    printf("Gemiddelde is: %d\n", gem);
    return 0;
}
```

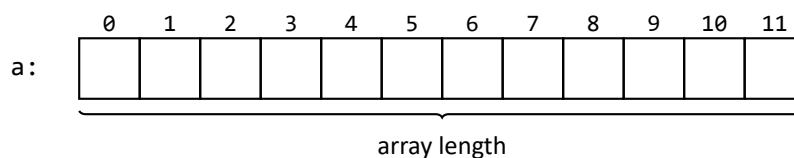
We kunnen hierbij twee opmerkingen plaatsen. De code voor het inlezen van de vijf getallen lijkt sterk op elkaar. Het programma wordt onpraktisch als we het gemiddelde van 100 getallen willen berekenen.

C.1 Array's

We kunnen bij het ontwerpen van het programma gebruik maken van *array's*. Een array wordt een *samen-gesteld datatype* genoemd. Hieronder volgt de declaratie van een array van twaalf integers:

```
int a[12];
```

Hierin is *a* de naam van de array en 12 het aantal *elementen* van de array. Dit wordt de lengte van de array genoemd. De inhoud van de afzonderlijke elementen is niet gedefinieerd (behalve als de array globaal wordt gedeclareerd, dan is de inhoud 0). Een array is uit te beelden als een rechthoek met hokjes zoals hieronder is te zien.



Te zien is dat de array begint bij element 0 en eindigt bij element 11. In C begint een array altijd bij element 0. Het laatste of hoogste element is één minder dan de lengte van de array. In het voorbeeld is het laatste element dus 11.

We kunnen de inhoud van een array tijdens runtime specificeren met een zogenoemde *array initializers*. Bij een volledige lijst met initializers mag de lengte weggelaten worden. Een voorbeeld:

```
int a[] = {9, 5, 13, 19, 12, 8, 7, 4, 21, 2, 6, 3};
```

De lengte van de array komt overeen met het aantal initializers. Om een array met nullen te initialiseren, kunnen we gebruik maken van een gedeeltelijke initialisatie:

```
int a[12] = { 0 };
```

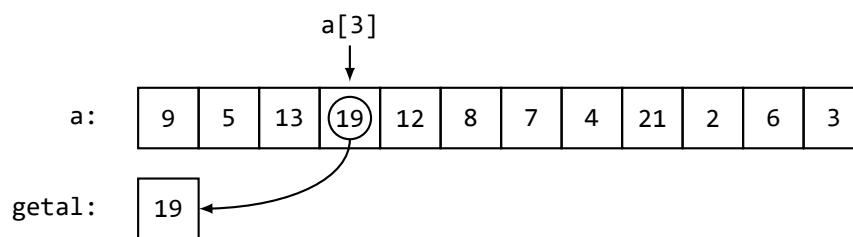
Element 0 wordt expliciet met 0 geïnitieerd. De overige elementen worden, conform de C-standaard, altijd met 0 geïnitieerd, ook als de eerste initializer ongelijk aan 0 is.

Om een element uit een array te gebruiken, moeten de naam van de array en het elementnummer (tussen blokhaken) worden opgegeven. Het onderstaande voorbeeld kopieert element 3 uit de array in variabele `getal`:

```
int a[] = {9, 5, 13, 19, 12, 8, 7, 4, 21, 2, 6, 3};
int getal;

getal = a[3];
```

In de onderstaande figuur is de werking van het programma uitgebeeld.



Met behulp van een array is het makkelijk om de som te bepalen van een groot aantal getallen:

```
...
int a[] = {9, 5, 13, 19, 12, 8, 7, 4, 21, 2, 6, 3};
int som = 0;
...
for (i=0; i<12; i=i+1) {
    som = som + a[i];
}
...
```

De lengte (of grootte) van een array is te bepalen met behulp van de keyword `sizeof`. Let daarbij op dat `sizeof` de grootte in *bytes* berekend. Om het aantal elementen van de array te bepalen, moet de grootte in bytes nog worden gedeeld door de grootte van één element:

```
array_size = sizeof a / sizeof a[0];
```

C.2 Strings

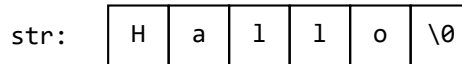
Een string in C is niets anders dan een array van `char`'s, afgesloten met een zogenoemde *null-byte* (`\0`). Zo maakt de onderstaande declaratie de array `str` aan met een grootte van 6.

```
char str[] = "Hallo";
```

Dit is equivalent met:

```
char str[] = { 'H', 'a', 'l', 'l', 'o', '\0' };
```

Een uitbeelding van de string is hieronder te zien:



Let erop dat de array dus één element groter is dan het aantal karakters in de array. Dit is een veel gemaakte fout bij het afhandelen van strings (een zogenoemde *off-by-one error*). Omdat het null-byte het einde van de string aangeeft, kunnen we er gebruik van maken, bijvoorbeeld voor het bepalen van de lengte van de string. Hieronder zie je een mogelijke implementatie van de functie `strlen`:

```
int stringlength(char str[]) { /* use another function name to prevent */
                               /* clash with strlen */
    int count = 0;

    while (str[count] != '\0') {
        count++;
    }

    return count;
}
```

C.3 Pointers

Een *pointer* is een variabele waarvan de inhoud het *adres* is van een (andere) variabele in het computergeheugen. Hieronder is te zien dat de pointer `pa` wijst naar de variabele `a`.



We kunnen dit als volgt in code programmeren:

```
int a; /* an integer */
int *pa; /* a pointer to an integer */

a = 3; /* set integer a to 3 */
pa = &a; /* let pa point to a */
```

We kunnen nu door middel van de pointer een variabele aanpassen:

```
a = a + 1; /* increment integer variabele a with 1 */
*pa = *pa + 1; /* increment variabele through pointer */
a = *pa + 1; /* also works */
*pa = a + 1; /* also works */
```

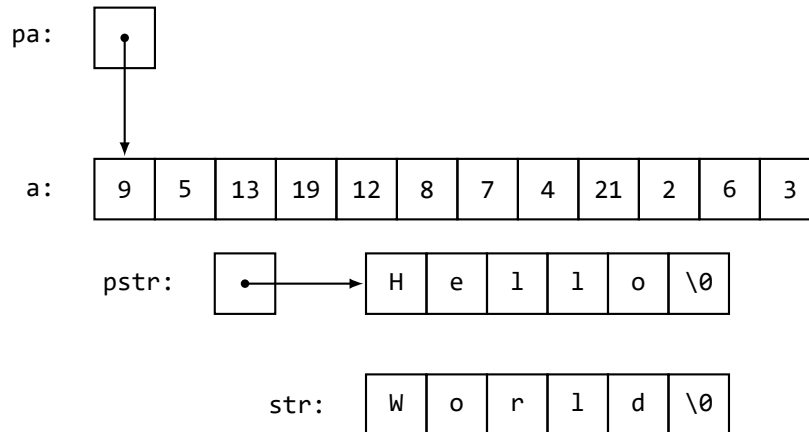
Array's en pointers hebben een sterke relatie. De *naam van een array* is een *synoniem* van een pointer naar het eerste element van een array:

Pointers kunnen worden aangepast. De naam van een array kan *niet* worden aangepast. We demonstreren dit aan de hand van twee strings in het computergeheugen. De pointer `pstr` wijst naar een string in het geheugen. De identifier `str` is de naam van een array.

We kunnen `pstr` dus wel aanpassen en `str` niet:

```
int a[12] = {9,5,13,19,12,8,7,4,21,2,6,3};
int *pa;
```

```
pa = a; /* of: pa = &a[0]; */
```



```
char *pstr = "Hello"; /* pstr: pointer to first address of array */
char str[] = "World"; /* str: alias for first address of array */
```

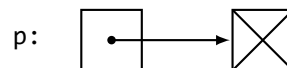
```
pstr = str; /* pstr points to first address of str */
str = pstr; /* ILLEGAL: str cannot be assigned! */
```

Een speciaal geval van een pointer-adres is de zogenoemde NULL-pointer. De inhoud van de pointer is dan 0 (alle bits van de pointer-variabele zijn 0). In principe is er geen verschil met elke andere inhoud van de pointer, maar een NULL-pointer wordt gebruikt om aan te geven dat een pointer nergens naartoe wijst.

```
/* #define NULL ((void*)0) */
```

```
char *p = NULL;
```

Een NULL-pointer wordt als volgt weergegeven:



Een NULL-pointer mag niet gebruikt worden om een variabele te benaderen. Dit wordt een *NULL-pointer dereference*¹¹ genoemd en leidt in veel gevallen tot een crash van het programma.

```
/* #define NULL ((void*)0) */
```

```
char *p = NULL;
*p = *p + 1; /* Oops!!! */
```

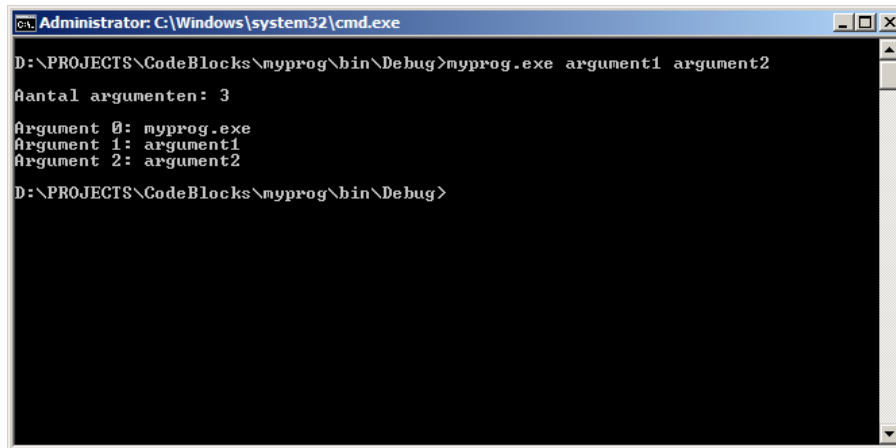
¹¹ Zie https://en.wikipedia.org/wiki/Null_pointer#Null_Dereferencing

D Command line argumenten en bestandsafhandeling in C

In besturingssystemen die C ondersteunen zoals Windows, Linux en Mac OS X, is het mogelijk om een C-programma *command line argumenten* mee te geven en bestanden te manipuleren. In deze bijlage doen we dat in een zogenoemde DOS-box onder Windows.

D.1 Command line argumenten

In figuur 2 is te zien dat het programma `myprog.exe` gestart wordt met de argumenten `argument1` en `argument2`. Het programma drukt eenvoudigweg alle argumenten op het beeldscherm af. Te zien is dat ook de programmaam als argument wordt meegegeven.



Figuur 2: Voorbeeld van een commando met argumenten.

Elk C-programma krijgt per definitie de twee parameters `argc` en `argv` mee, die aan `main()` worden meegegeven. Dit is te zien in listing 1.

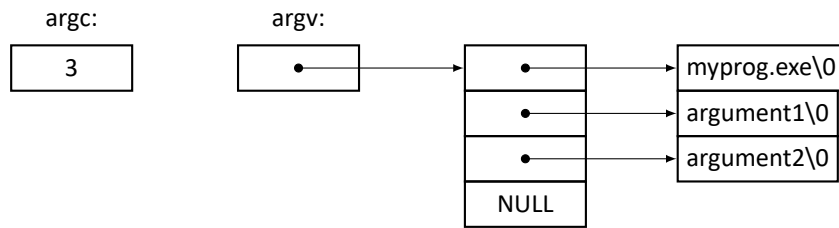
```
int main(int argc, char *argv[]) {  
    /* rest of the code */  
    return 0;  
}
```

Listing 1: Declaratie van de command line parameters.

De integer `argc` (**argument count**) geeft aan hoeveel parameters aan het C-programma zijn meegegeven. De pointer `*argv[]` (**argument vector**) is een pointer naar een lijst van pointers naar strings. Elke string bevat een argument. Per definitie wijst `argv[0]` naar een string waarin de programmaam vermeld staat. Dat houdt in dat `argc` dus minstens 1 is. Er zijn dan geen optionele argumenten meegegeven.

In het voorbeeldprogramma is `argc` dus 3 en zijn `argv[0]`, `argv[1]` en `argv[2]` pointers naar respectievelijk `myprog.exe`, `argument1` en `argument2`. In figuur 3 is een uitbeelding van de variabelen `argc` en `argv` te zien. De strings worden, zoals gebruikelijk in C, afgesloten met een `\0`-karakter. De C-standaard schrijft voor dat de lijst van strings wordt afgesloten met een `null`-adres.

Het programma `myprog.exe` is te zien in listing 2. Het programma drukt eerst de variabele `argc` af. Met behulp van een `for`-lus worden de argumenten één voor één afgedrukt. Merk op dat `argv[i]` een pointer is naar het *i*^e argument. We kunnen `argv[i]` dus direct gebruiken voor het afdrukken van de bijbehorende string.



Figuur 3: Uitbeelding van de variabelen argc en argv.

```

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    int i;

    printf("\nAantal argumenten: %d\n\n", argc);
    for (i = 0; i < argc; i++) {
        printf("Argument %d: %s\n", i, argv[i]);
    }
    return 0;
}
  
```

Listing 2: Het programma myprog.exe.

D.2 Bestandsafhandeling in C

In C is het mogelijk om met bestanden op de harde schijf (of USB-stick of SD-card) te werken. We spreken dan over File I/O.

In listing 3 is de code te zien om een bestand te schrijven. We zullen er stap voor stap doorheen lopen. In regel 6 wordt de pointer `outfile` naar een object `FILE` gedeclareerd. Met behulp van de pointer kunnen we onder andere lezen van een bestand en schrijven naar een bestand. In regel 8 wordt het bestand `C:\temp\testbestand.txt` geopend voor schrijven. Dat is te zien aan de tweede argument `"w"`. Merk op dat in de bestandsnaam gebruik is gemaakt van een dubbele backslash omdat een backslash door C wordt gebruikt voor afprintbare karakters zoals `\n`. De functie `fopen` geeft een adres terug die wordt toegekend aan de pointer `outfile`. Als om één of andere reden het bestand niet geopend kan worden, geeft de functie `fopen` een null-pointer terug. We moeten hier natuurlijk op testen. Dat is te zien in regel 10. We drukken dan een verklarende tekst af (regel 11). Als het bestand voor schrijven geopend kan worden, dan schrijven we een regel naar dat bestand. Dat is te zien regel 13. De functie `fprintf` lijkt veel op de bekende `printf`-functie maar heeft één extra parameter, namelijk de pointer `outfile`. In regel 14 wordt het bestand met behulp van de functie `fclose` gesloten.

Listing 4 laat de code zien om een bestand te lezen. Er wordt weer gebruik gemaakt van een `FILE`-pointer, in dit geval de pointer `infile`. Het genoemde bestand wordt weer geopend met de functie `fopen` maar ditmaal met als tweede argument `"r"` wat staat voor lezen uit een bestand. Mocht om één of andere reden het bestand niet geopend kunnen worden, dan geeft `fopen` een null-pointer terug. Hier testen we op in regel 11.

In de regels 14 t/m 16 worden de karakters één voor één ingelezen en afgedrukt op het beeldscherm. We zien hier een typische C-`while`-constructie. Er wordt een karakter ingelezen door middel van de functie `fgetc` en toegekend aan de variabele `ch`. Tegelijkertijd wordt getest of het ingelezen karakter gelijk is aan EOF wat staat voor end-of-file. Zolang dat niet het geval is wordt het karakter afgedrukt met de functie `printf`.

```

#include <stdio.h>
#include <stdlib.h>

int main() {

    FILE *outfile;

    outfile = fopen("C:\\temp\\testbestand.txt", "w");

    if (outfile == NULL) {
        printf("Bestand kan niet geopend worden.\n");
    } else {
        fprintf(outfile, "Een stukje tekst\n");
        fclose(outfile);
    }

    return 0;
}

```

Listing 3: Voorbeeld schrijven naar een bestand.

```

#include <stdio.h>
#include <stdlib.h>

int main() {

    FILE *infile;
    int ch;

    infile = fopen("C:\\temp\\testbestand.txt", "r");

    if (infile == NULL) {
        printf("Kan bestand niet openen.");
    } else {
        while ((ch = fgetc(infile)) != EOF) {
            printf("%c", ch);
        }

        fclose(infile);

        return 0;
    }
}

```

Listing 4: Voorbeeld lezen uit een bestand.

D.3 Command line argumenten en bestandsafhandeling

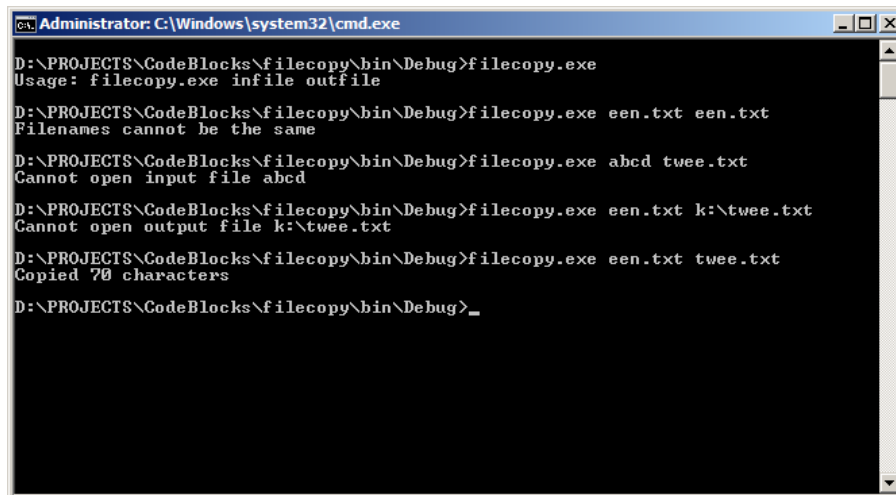
Het laatste voorbeeld betreft het kopiëren van een bestand. We gebruiken de command line argumenten om de bestandsnamen door te geven en bestandsafhandeling om het daadwerkelijke kopiëren te realiseren. Het programma is te vinden in listing 5 op pagina 53.

We declareren eerst de pointers voor de twee bestanden, dit is te zien in regel 8. In regel 10 declareren we twee integers, één voor de in te lezen karakters en één voor het bijhouden hoeveel karakters er zijn gekopieerd. In de regels 14 t/m 17 testen we of het aantal opgegeven argumenten drie is. Zo niet, dan drukken we een helptekst af om de gebruiker te informeren hoe het commando moet worden gebruikt. Tevens wordt het programma afgesloten met een **return**-statement. Deze geeft de waarde `-1` terug aan het besturingssysteem, een gebruikelijke techniek in Unix-achtige omgevingen om foutmeldingen door te geven.

In de regels 20 t/m 23 wordt getest of de namen van de twee bestanden identiek zijn. Dat mag natuurlijk niet anders wordt het leesbestand overschreven. We geven een passende foutmelding en verlaten het programma middels een `return`-statement met de waarde `-2`.

In de regels 26 t/m 30 wordt het leesbestand geopend. We hebben dit al eerder besproken. In de regels 33 t/m 38 wordt het schrijfbestand geopend. Ook dit is al eerder besproken. Het daadwerkelijke kopiëren gebeurt in de regels 41 t/m 44. Tevens wordt bijgehouden hoeveel karakters er zijn gekopieerd. Dit aantal wordt afgedrukt in regel 46.

We ronden het programma af door het leesbestand en schrijfbestand af te sluiten. Een aantal voorbeelden van het gebruik van het programma is te zien in figuur 4.



```
Administrator: C:\Windows\system32\cmd.exe
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>filecopy.exe
Usage: filecopy.exe infile outfile
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>filecopy.exe een.txt een.txt
Filenames cannot be the same
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>filecopy.exe abcd twee.txt
Cannot open input file abcd
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>filecopy.exe een.txt k:\twee.txt
Cannot open output file k:\twee.txt
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>filecopy.exe een.txt twee.txt
Copied 70 characters
D:\PROJECTS\CodeBlocks\filecopy\bin\Debug>_
```

Figuur 4: Voorbeelden van het kopieerprogramma.

D.4 Openen van een DOS-box op Windows

Een DOS-box kan je openen via de Search-box (in het voorbeeld onder Windows 10). Klik op de Search-box en vul in het kader de tekst `cmd.exe` in. Druk op de enter-toets en een DOS-box wordt geopend. Zie figuur 5.



Figuur 5: Openen van een DOS-box.

Vervolgens kun je navigeren naar de *executable* (uitvoerbaar programma) middels het commando `cd` (change directory). In het voorbeeld van figuur 4 doe je als volgt:

```
cd \Users\JouwNaam\Documents\CodeBlocks\filecopy\bin\Debug
```

Vul voor JouwNaam je gebruikersnaam in waarmee je inlogt. We gaan er hier trouwens vanuit dat je de programma's in de map CodeBlocks zet die in de map Documents staat.

Als je je programma's op een andere schijf geplaatst hebt, moet je eerst van schijf veranderen. Stel dat je je programma's op de D:-schijf hebt geplaatst, dan verander je van schijf met het commando:

```
D:
```

Daarna kan je naar je map navigeren, bijvoorbeeld:

```
cd \PROJECTS\CodeBlocks\filecopy\bin\Debug
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char *argv[]) {
6
7      /* The input and output file handlers */
8      FILE *infile, *outfile;
9
10     /* Character to read and character count */
11     int ch, count = 0;
12
13     /* We need exactly 3 arguments */
14     if (argc != 3) {
15         printf("Usage: %s infile outfile\n", argv[0]);
16         return -1;
17     }
18
19     /* Test if input file and output file have the same name */
20     if (strcmp(argv[1], argv[2]) == 0) {
21         printf("Filenames cannot be the same\n");
22         return -2;
23     }
24
25     /* Open the input file, exit if error */
26     infile = fopen(argv[1], "r");
27     if (infile == NULL) {
28         printf("Cannot open input file %s\n", argv[1]);
29         return -3;
30     }
31
32     /* Open the output file, exit if error */
33     outfile = fopen(argv[2], "w");
34     if (outfile == NULL) {
35         printf("Cannot open output file %s\n", argv[2]);
36         fclose(infile);
37         return -4;
38     }
39
40     /* Copy characters from input file to output file */
41     while ((ch = fgetc(infile)) != EOF) {
42         count++;
43         fprintf(outfile, "%c", ch);
44     }
45
46     printf("Copied %d characters\n", count);
47
48     fclose(infile);
49     fclose(outfile);
50
51     return 0;
52 }

```

Listing 5: Programma om een bestand te kopiëren.