UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

LABO
Programmeren in C en C++
Oefeningenbundel

Leen Brouns

Helga Naessens

Wim Van den Breen

Opleiding Industrieel Ingenieur Informatica / Elektronica / Automatisering

september 2017

Doelstelling van de labo's C en C++

Met het aanbieden van de labo's C en C++ willen we inhoudelijk volgende zaken op een rijtje krijgen:

1. **inzicht in het verschil tussen C /C++ en Java**
   Zodat je op het juiste moment de juiste redenering en syntax gebruikt. Daarom zullen we je (voor dit ene vak) ook aanmoedigen om af en toe zonder IDE te werken: veel oefenen en zélf de code intikken laat je bewuster omgaan met de verschillen tussen Java en C /C++ .

2. **een reflexmatige voorkeur voor leesbare en efficiënte code**
   Je krijgt in deze cursus heel wat programmeeropgaven. Uiteraard gaan we er vanuit dat afgewerkte code de gevraagde output levert. Dat is echter nog maar de startvoorwaarde. Daarnaast is ook leesbaarheid en efficiëntie zeer belangrijk. Tip: vergelijk onze oplossingen altijd kritisch met je eigen code. Zit er een verschil in leesbaarheid en/of efficiëntie? Leer daarvan — of laat ons weten wat beter kan in de voorbeeldoplossing.

3. **een zekere vlotheid in je basishandelingen**
   Bepaalde programmaonderdelen, zoals aanbieden van een keuzemenu, (handmatig) zoeken in een tabel, bewerkingen op cijfers van gehele getallen,... komen dikwijls voor. Toch zeker als je met low-level-talen werkt. Even dikwijls zien we echter teveel ballast verschijnen in de geproduceerde code: teveel hulpvariabelen, overbodige testen, storende bewerkingen tussendoor. We geven je enkele stijltips, en sommen op welke basishandelingen je vlot uit je mouw moet kunnen schudden. (Dit staat uiteraard los van de gebruikte programmeertaal, maar we hebben hier de gelegenheid enkele van onze aandachtspunten aan jullie voor te stellen.)

4. **een goed begrip van het hot topic in C(++) : pointers**
   Dit komt jullie nog van pas in andere vakken (o.a. netwerkprogrammatie), daarom schakelen we in de oefeningen vrij snel pointers in.

5. **feeling met de 'low level' features in C**
   Hier denken we onder andere aan het gebruik van `char*` in plaats van `string`. Even doorbijten in het begin — het is écht niet gedateerd, en nog altijd nuttig.

6. **een gepast gebruik van de techniek van het recursief programmeren**
   (zowel in interface als functie-opbouw), met afweging van de voor- en nadelen ten opzichte van niet-recursief programmeren.

7. **een eerste overwinning op C++11**
   zodat jullie zelf verder kunnen in de *New Standard War*, die in *C++Lands* wordt uitgevochten. Zie voor de roadmap hiervan `http://fearlesscoder.blogspot.be/2012/01/c11-lands.html`.

8. ...

Meer algemeen streven we naar

1. **een denk- in plaats van tik-reflex**
   Vooral bij het gebruik van pointers, gelinkte lijsten, boomstructuren, pointers naar pointers,... is het van belang dat je de zaken al bij de eerste poging juist op papier zet. (Jawel, PAPIER, uitvinding van voor onze jaartelling, maar sedertdien o zo geduldig.) Begin je gehaast aan een kladpoging op je scherm, en sla je de bal of de pointer mis, dan kan elke kleine aanpassing (poging tot verbetering) je verder af drijven van het juiste pad. Dit straatje zonder eind kan zeer frustrerend werken, en 'tabula rasa' maken is dan dikwijls het enige aangewezen hulpmiddel.
   Is de oefening wat groter, en wil je aan alles tegelijk beginnen? Maak eerst een gerangschikt (!) to-do-lijstje dat je tijdens de loop van de opdracht kan afvinken. Dat vraagt soms dat je procedures of functies voorlopig 'schetst' (bvb. een hardgecodeerde waarde laat teruggeven, in afwachting van betere implementatie).

2. **propere manieren wat communicatie betreft**
   Hulp nodig tijdens een labo? Let op hoe je die vraagt. De zinsnede *Meneer/mevrouw, het werkt niet!* is geen vraag, maar een vaststelling waarop je lesgever enkel instemmend kan knikken. Zorg voor

   - een duidelijke omschrijving van de context
   - een stukje code
   - het probleem dat zich voordoet (compileerfout, error at runtime, onverwachte resultaten,...)
   - wat je eventueel zelf al probeerde

   We helpen je graag bij het juist formuleren van je vraag — dikwijls vind je hierdoor ook zelf al de oplossing.

3. **een goede inschatting van je eigen vorderingen en mogelijkheden**
   Voor de reguliere bachelorstudenten onder jullie: allicht hebben jullie nog een pak programmeerervaring op te doen. Op twee fronten tegelijk: Java én C(++) . Probeer gelijke tred te houden in beide vakken, en een speekmedaille af en toe kan alleen maar deugd doen!
   Voor de schakelstudenten: dit jaar wordt er veel van jullie verwacht — veel algemene vakken die na lange tijd weer op de agenda staan, ernstig voorbereid naar alle labo's komen, eventuele leemtes tussen vooropleiding en nieuwe leerstof zelf opvullen. Je zou voor minder het overzicht verliezen, of jezelf foutief inschatten. Ook hier: houd de pas erin, en wees fier op elk bereikt deelresultaat.
   Voor beide groepen: jullie krijgen alvast enkele praktische hulpmiddelen — om te beginnen elke week twee volledig uitgesponnen theorielessen, in het labo een opeenvolging opdrachten van opbouwende moeilijkheidsgraad, geregelde feedback van de aanwezige docenten, oplossingen die je kritisch naast je eigen code dient te leggen. Doe je voordeel met de aangeboden hulp om zo vlot mogelijk de eindmeet te halen.

4. **een zicht op jullie kijk op de zaak**
   Aarzel dus niet om op- of aanmerkingen door te geven. IEDEREEN moet mee zijn met deze leerstof!
   Voor de schakelstudenten: ben je niet mee met deze leerstof, dan is er geen beginnen aan in het tweede semester (labo bij de cursus Algoritmen I).
   Voor de bachelorstudenten: jullie hebben bovendien nog een 'gap' van 12 maanden te overbruggen voor jullie aan de cursus Algoritmen I beginnen. Zorg dus dat de leerstof goed verankerd is!

Veel succes!

Software

Voor dit labo kan je kiezen: lokaal werken (dat kan op eigen laptop of op de labotoestellen in lokalen B2.031 en B2.035) of via Athena. Zoek zelf uit wat voor jou het snelst/handigst werkt.

Via Athena

Lees aandachtig hoe je Athena best gebruikt op `www.helpdesk.ugent.be/athena/gebruik.php`. Zo kan je software gebruiken zonder die zelf te installeren.

Je beschikt over een Netwerkshare die je via Athena overal kan raadplegen (lees het onderdeel `Centrale Schijfruimte`). Nadat Athena opgestart is kan je met `File Explorer` een verkenner openen vanuit Athena, waar je ook toegang hebt tot de lokale computer. Op je centrale schijfruimte (of netwerkdrive, `H`-drive) staat alles veilig, er worden backups gemaakt, en je kan het van overal terug bereiken.

Toepassingen die je in Athena start werken veel sneller als je de bestanden ook opslaat op de netwerkdrive.

Open in het labo Athena zodat je alles kan uitproberen; voeg `Dev-C++` en `Command Shell` (beide onder `Academic/Development` te vinden) alvast toe aan `MyAthena`.

Maak op je `H`-drive een map aan voor deze cursus, en een submap voor de eerste reeks. Dan kan je starten. Gezien we in deze cursus vooral focussen op korte oefeningen, is het niet nodig om telkens een nieuw project aan te maken. Allicht heb je aan aparte bestanden genoeg — in `Dev-C++` raden we het gebruik van projecten trouwens af!

Lokaal

Wie kiest om lokaal te werken op de labotoestellen in B2.031 en B2.035, moet weten dat de U-drive in onbruik is, en je dus de UGent-cloud moet mounten (icoon met huisje op je bureaublad) zodat je op je H-drive kan werken. Het mounten laat je toe om de lokaal geïnstalleerde software te gebruiken (snelkoppeling Dev-C++ op je bureaublad).

Wie op een eigen Windows-toestel werkt, downloadt de laatste versie van Orwell Dev-Cpp op `http://sourceforge.net/projects/orwelldevcpp/`.

Instellingen Dev-C++

Om je bladschikking in `Dev-C++` deftig te krijgen, pas je volgende zaken aan: onder `Tools - Editor Options - tabblad General` vink je (`SmartTabs` uit en) `Use Tab Character` aan.

Je moet ook de juiste compileropties aan- of afvinken. Ga daarvoor in de menubalk naar `Tools - Compiler Options`. Voor C -programma's komt er in het tabblad `General` onder de compileropties de optie `-pedantic` terwijl de linkeropties uitgevinkt moeten worden. Voor C++ -programma's komt er onder de compileropties de optie `-std=c++11` (met een kleine c) en de linkeropties vink je aan.

Nog een weetje: via Athena staan de instellingen bij de start misschien op `qwerty`-klavier. Verander dit met `Alt-Shift`.

REEKS 1

Kennismaking met C
main(), tabellen en eenvoudige functies / procedures

Oefening 1

Schrijf een programma dat volgende tekst op het scherm brengt (delay bij uitschrijven van de verschillende getallen is niet nodig). (En wat als we laten aftellen vanaf 100? Heb je de output `10 9 8 7 6 5 4 3 2 1` hardgecodeerd? Niet doen!)

```
Hello world!
10 9 8 7 6 5 4 3 2 1
START
```

Oefening 2

Schrijf een programma dat alle (gehele) getallen van 0 tot en met 64 uitschrijft. Per regel komt zowel octale, decimale, als hexadecimale voorstelling van één getal. Zorg ervoor dat de getallen rechts gealligneerd zijn, dus zo:

```
  0    0    0
  1    1    1
  2    2    2
  3    3    3
  4    4    4
  5    5    5
  6    6    6
  7    7    7
 10    8    8
 11    9    9
 12   10    a
 13   11    b
 14   12    c
 15   13    d
 16   14    e
 17   15    f
 20   16   10
 21   17   11
 ...
 77   63   3f
100   64   40
```

Oefening 3

Gegeven volgend programmafragment, als oplossing voor oefening 1. Dit levert de gevraagde output. Het levert echter op een examen geen punten op. Waarom niet?

```
    int i;
    for(i=10; i>0; i--){
        if(i==10){
            printf("Hello world!\n");
        }
        printf("%d ",i);
        if(i==1){
            printf("\nSTART");
        }
    }
```

Oefening 4

Gegeven de opgave *schrijf alle machten van 2 (beginnend bij $2^0 = 1$), kleiner dan 10.000 uit.* Onderstaande code is foutief. Geef aan hoe je op zicht ziet dat er iets loos is.

```
#include <stdio.h>
int main(){
    int macht = 1;
    while(macht < 10000){
        macht *= 2;
        printf("%d ",macht);
    }
    return 0;
}
```

Oefening 5

Deze code levert, in tegenstelling tot vorige oefening, wél de gevraagde output. Toch levert deze amper meer punten op dan de vorige oplossing. Waarom?

```
#include <stdio.h>
int main(){
    int macht = 1;
    int i;
    for(i=0; i<20; i++){
        printf("%d ",macht);
        macht *= 2;
        if(macht > 10000){
            break;
        }
    }
    return 0;
}
```

Oefening 6

Bij het berekenen van de sinus van een gegeven hoek, gebruikt je computer of zakrekenmachine onderstaande reeksontwikkeling.

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

1. Schrijf een programma dat de sinus berekent van 0.23 radialen. Opgelet: we kijken naar efficiëntie van je berekeningen! Vergelijk dit met het resultaat van de ingebouwde sinusfunctie uit de bibliotheek `math.h`.
2. Pas het programma aan. In plaats van de sinus van 0.23 radialen, bereken je de sinus van een getal dat (door het programma) willekeurig gekozen wordt in het interval $[-3.200, 3.200[$. Het getal bevat dus drie beduidende decimalen. Gebruik de methode `rand`; voor meer informatie zoek je online.
3. De derde versie die je schrijft, vraagt aan de gebruiker een (reëel) getal, waarna er opnieuw twee sinuswaarden berekend worden: via de zelfgeschreven reeksontwikkeling en via de ingebouwde sinusfunctie van `math.h`.
4. Na theorieles 2 kan je de berekening van de reeksontwikkeling in een functie verpakken; geef deze de naam `mijn_sinus`. Pas ook het hoofdprogramma aan.

Het efficiënt implementeren van een reeksontwikkeling is niet evident. Vergelijk bijvoorbeeld het aantal berekeningen dat jouw code uitvoert met het aantal berekeningen van de voorbeeldoplossing. Heb je graag nog wat extra oefeningen op reeksontwikkelingen, dan vind je er op `https://nl.wikipedia.org/wiki/Taylorreeks` nog een hele resem (zie 'Ontwikkelingen'), die bovendien eenvoudig te controleren zijn. Zo kan je bijvoorbeeld de uitkomst die jouw zelfgeprogrammeerde reeksontwikkeling

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

van `ln(1+x)` geeft, vergelijken met de uitkomst van `ln(1+x)`. Kies in dit geval `x` dan wel tussen -1 en 1 zoals de voorwaarde eist, en zoek uit hoe `ln` in C-taal geschreven wordt.

Oefening 7

Schrijf een programma dat aan de gebruiker vijf positieve gehele getallen vraagt. Geeft de gebruiker echter een negatief getal in, dan stopt het programma met getallen opvragen. Nadien schrijft het programma uit of de gebruiker inderdaad vijf positieve gehele getallen opgaf. Tot slot wordt de som van de ingegeven positieve gehele getallen uitgeschreven (ook als er niet genoeg ingegeven werden). Test uit. Wat met de getallenreeks `1 2 3 4 -5`?

Vanaf de volgende oefeningen heb je theorieles 2 nodig: er wordt gebruik gemaakt van functies (zowel implementatie als oproepen van functies) en van recursie.

Oefening 8

1. Schrijf een functie `cijfersom(x)` die van een gegeven geheel getal `x` de som van de cijfers berekent. Zo is `cijfersom(12345)` gelijk aan 15. Doe dit zonder bewerkingen op karaktertekens, gebruik enkel het type `int` (en wat wiskunde uit de lagere graad).
2. Gebruik deze functie in de functie `cijfersom_herhaald(x)`; deze blijft de som van de cijfers berekenen tot een getal kleiner dan 10 bekomen wordt. Zo is `cijfersom_herhaald(12345)` gelijk aan 6.
3. Schrijf een hoofdprogramma dat 10 strikt positieve gehele getallen inleest. (Indien een getal niet aan de voorwaarden voldoet, dan blijft het programma vragen naar een strikt positief geheel getal tot het gegeven wordt.) Bij elk positief geheel getal dat wordt ingegeven, wordt meteen geantwoord met de cijfersom.

4. Maak een recursieve versie `cijfersom_rec(x)` die hetzelfde doet als `cijfersom_herhaald(x)`. Test uit in het hoofdprogramma dat je als volgt aanpast: beantwoord elke input van de gebruiker met drie zaken, te weten (1) de cijfersom berekend met `cijfersom_herhaald(x)`, (2) de cijfersom berekend met `cijfersom_rec(x)` en (3) de uitkomst (ok / niet ok) van de vergelijking van deze beide cijfersommen.

Oefening 9

Schrijf een functie `faculteit(x)` die de faculteit van een gegeven geheel getal `x` berekent.
Schrijf een procedure `schrijf_faculteit(x)` die de faculteit van een gegeven geheel getal `x` uitschrijft.
Roep deze procedure op voor het getal 5. Daarna doe je dit voor alle getallen van 0 (0!=1) tot en met 40. Wat merk je? (Probleem dat zich stelt hoef je niet op te lossen, enkel te constateren.)

Oefening 10

Om de grootste gemende deler (ggd) van twee getallen te berekenen, werd je allicht aangeleerd om de twee getallen eerst te ontbinden in priemfactoren, om dan de gemeenschappelijke factoren te ontdekken. Het kan ook anders, met het algoritme van Euclides dat gebruik maakt van volgende gelijkheid:

```
ggd(a , b) = ggd(b , a mod b);
```

De uitdrukking `a mod b` lees je als 'a modulo b' en stelt de rest van `a` bij deling door `b` voor. In C (en C++) gebruik je de notatie `%` in plaats van `mod`. Het algoritme van Euclides vervangt de getallen `a` en `b` (herhaaldelijk) door de getallen `b` en `a mod b`. Indien `a>b`, zullen `b` en `a mod b` kleiner zijn dan `a` en `b` - en dus werd het probleem vereenvoudigd. Dit vervangen stopt van zodra een van de getallen 0 is: `ggd(a,0) = a`.

Schrijf een recursieve functie `ggd(a,b)` die de grootste gemene deler van twee gehele getallen berekent. Test uit met

```
ggd(-6,-8)==2
ggd(24,18)==6
ggd( 0,-5)==5
ggd(6,-35)==1
```

Oefening 11

Wat doet deze code? Verklaar. Na theorieles 3 kan je de code zo omvormen, dat ze ook doet wat ze belooft.

```
void wissel(int a, int b){
     int hulp;
     printf("  Bij start van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
     hulp = a;
     a = b;
     b = hulp;
     printf("  Op het einde van de wisselprocedure hebben we a=%d en b=%d.\n",a,b);
}

int main(){
    int x, y;
    x = 5;
    y = 10;

    printf("Eerst hebben we x=%d en y=%d.\n",x,y);
    wissel(x,y);
    printf("Na de wissel hebben we x=%d en y=%d.\n",x,y);

    return 0;
}
```

Deze vraag beantwoord je eerst ZONDER de code uit te proberen. Daarna kan je jouw antwoord controleren, door de code te copiëren, compileren en uit te voeren. Omdat het copiëren vanuit een .pdf-bestand de kantlijnen niet behoudt, kan je copiëren vanuit een andere bron. (Zeker nuttig als we je later langere code geven!) Op Minerva vind je een map met .tex-bestanden. Daar haal je het juiste bestand op (`opg_18_11_wissel.tex`, waarbij `18` staat voor jaargang 2017-2018 en `11` het nummer van de oefening is), en kopieer je het programma (te vinden tussen `\begin{verbatim}` en `\end{verbatim}`) in een .c-bestand.

REEKS 2

Arrays

Gebruik van de commandolijn om een programma op te roepen

Eerst even wat uitleg over het runnen van je programma. Zolang je in een IDE (Integrated Development Environment) werkt, kan je een sneltoets gebruiken om je programma te laten lopen. Van zodra het programma getest en goedgekeurd is, heb je de broncode echter niet meer nodig. Enkel het programma zelf (`.exe`-extensie voor windows) en een venster met opdrachtprompt (command prompt window), volstaan. Probeer dit uit.

- Open een venster met command prompt (bvb. via de Windows startknop, tik onderaan bij 'zoek' de letters 'cmd').
- In dat venster navigeer je naar de schijf waarop je werk staat, bvb de H:-drive: tik in `H:` .
- Dan vraag je de inhoud van deze map op met het commando `dir`.
- Het commando `cd` *mapnaam* navigeert naar de map met de naam *mapnaam*.
- Het commando `cd ..` navigeert naar de bovenliggende map.
- Zodra je in de map bent waar het `.exe`-bestand staat, tik je de naam van dit bestand in. Dat kan zónder extensie, of met extensie `.exe`.
- Het programma loopt nu - voortijdig afsluiten kan met `Ctrl-C`.
- Dit opdrachtpromptvenster laat je constant open staan, zodat je niet telkens opnieuw moet navigeren.
- De pijltjestoets omhoog roept het vorige commando weer op.
- De tab-toets vervolledigt een bestands- of mapnaam, voor zover éénduidig.

Gebruik van input redirection

Je kan op de commandolijn ook aangeven dat elke input niet vanop het scherm komt (het scherm waarin je bezig bent met de commandolijn), maar dat je elke input uit een bepaald bestand moet nemen. Je doet dit door op de commandolijn eerst een `<`-teken te tikken, gevolgd door (spatie en) bestandsnaam. Let op: hiervoor hoef je helemaal niet te weten hoe je in C (of C++) een (tekst)bestand inleest (dat komt ook pas later in de theorie aan bod). In de code van je programma staat gewoon `scanf(...)`, maar de input komt niet van scherm (de standaardinput) maar uit het gegeven bestand. Hetzelfde kan je doen met output redirection: elk `printf`-commando stuurt de informatie niet naar het scherm, maar naar het opgegeven bestand. Nu gebruik je op de commandolijn eerst het `>`-teken gevolgd door (spatie en) bestandsnaam.

Een voorbeeld: als je op de commandolijn

```
keerom < verhaaltje.txt > omgekeerd.txt
```

intikt, dan roep je het programma `keerom.exe` op. De programmacode in het bestand `keerom.c` zal de inlees- en uitschrijfopdrachten `scanf` en `printf` gebruiken, maar elke `scanf`-opdracht haalt input uit het bestand `verhaaltje.txt` en elke `printf`-opdracht schrijft output weg naar het bestand `omgekeerd.txt`.

Oefening 12

Schrijf voor het eerste deel van deze oefening ENKEL een hoofdprogramma, geen functies of procedures. (Houd je hier aan, of je mist de clou van de oefening.)

1. Maak in het hoofdprogramma een array aan waarin elk element een karakter is. Vul deze array bij declaratie op als volgt:

   ```
   letters [] = {'p','o','r','e','o','i','f','o','i','e','c','i','i',':',
                 'a','-','t','('};
   ```

   Bepaal de lengte van de array zonder te tellen (zie theorie)! Schrijf nu (in het hoofdprogramma) alle karakters uit die op een even positie in de array staan.
2. Schrijf nu een procedure `schrijf_even_posities(array)` die alle karakters uitschrijft die op een even positie in de gegeven array staan. Roep deze procedure op in het hoofdprogramma en test uit. Wat merk je, en wat is daar de reden van? Los op door een extra parameter te voorzien.

Oefening 13

Schrijf een functie met naam `index_van(...)`, die de (kleinste) index teruggeeft van de plaats waarop een gegeven reëel getal in een gegeven array van reële getallen gevonden wordt. Indien het getal niet aanwezig is, wordt er -1 teruggegeven. Bepaal zelf aantal en aard van de parameters.

Wat verandert er als de gegeven array met zekerheid geordend is? (Niet uitwerken, wel netjes formuleren.)

Oefening 14

Schrijf een procedure die alle elementen van een gegeven array met lettertekens, één plaats naar links schuift. Het eerste karakter komt achteraan. Roep deze procedure drie keer aan op de array

```
char rij[] = {'s','a','p','a','p','p','e','l'};
```

Aangezien je de array verschillende keren moet uitschrijven, voorzie je hiervoor uiteraard een procedure.

Voor de snelle geest: argumenteer waarom de voorgestelde oplossing niet altijd even efficiënt is, en welk (ander) nadeel een efficiëntere methode dan weer heeft.

Oefening 15

Schrijf een programma dat 20 gehele getallen genereert tussen 100 en 120 (grenzen inbegrepen) en deze ook op het scherm toont. Daarna worden alle getallen die niet gekozen werden, in stijgende volgorde uitgeschreven. Kijk kritisch na!

Heb je de grenzen en het aantal te genereren getallen in constanten bewaard?

Oefening 16

Schrijf een programma dat tekstuele input van de gebruiker, afgesloten met een `$`-teken, verwerkt. Van elke letter wordt bijgehouden hoe dikwijls hij voorkomt. Hierbij worden hoofdletters meegerekend bij de overeenkomstige kleine letters. Daarna wordt er een histogram van gemaakt, en dit op twee manieren.

- Indien de gebruiker de tekst

  ```
  Als je tevreden bent met weinig, bezit je veel. $
  ```

  intikt, dan komt er als output

  ```
  a:   *
  b:   **
  c:
  d:   *
  e:   ***********
  ...
  z:   *
  ```

- Voor diezelfde input wordt het histogram daarna ook als volgt voorgesteld:

  ```
      e
      e
      e
      e
      e
      e
      e
      e              t
      e   i    n     t
   b  e   ij l n     t v
  ab de g ij lmn   rst vw  z
  ```

Voorwaarden waaraan je oplossing moet voldoen:

- Je schrijft en gebruikt drie procedures: één die de tekst inleest/verwerkt, één die een horizontaal staafdiagram tekent en één die een vertikaal staafdiagram tekent.
- Je gebruikt, buiten `scanf` en `printf`, geen bibliotheekfuncties. Zelf extra hulpfuncties schrijven kan wel.

Laat je programma vanaf de command line draaien, waarbij je input redirection gebruikt om de input te halen uit het bestand `gandhi.txt`. In welke taal is de tekst geschreven - kan je dat afleiden uit de frequentie van de gebruikte letters?

Oefening 17

Keer terug naar oefening 11. Hier werd een `wissel`-procedure gegeven, die echter niet deed wat ze beloofde. Pas de code aan (haal eerst het bestand `opg_18_11_wissel.tex` af van Minerva (map met LaTeX-bestanden (extensie `.tex`)), zodat je geen code hoeft over te tikken). Aan het hoofdprogramma verander je niets of zo weinig mogelijk.

Oefening 18

- Schrijf een procedure `zoek_extremen(...)` die op zoek gaat naar het minimum én maximum in een array van gehele getallen. Het type en de aard van de parameters bepaal je zelf. In het hoofdprogramma zorg je voor een hardgecodeerde array, en je schrijft het minimum en maximum uit. Test kritisch!
- Schrijf een recursieve versie van deze procedure.

REEKS 3

pointers, functiepointers en C-strings

Oefening 19

Schrap in het rood de regels code die syntactisch niet correct zijn. Schrap in het blauw de bewerkingen die wel kloppen qua syntax, maar in het verloop van het programma zinloos en/of gevaarlijk zijn. Wat zit er uiteindelijk in de variabelen? Beantwoord alles zonder computer.

```
main(){
    int i=7, j;
    double d;
    int *ip, *jp, *tp;
    double *dp;
    const int * p1;
    int * const p2 = &i;
    int t[25];
    int k;
    for(k = 0; k < 25; k++){
        t[k] = 10*k;
    }

    /* 1*/ i = j;
    /* 2*/ jp = &i;
    /* 3*/ j = *jp;
    /* 4*/ *ip = i;
    /* 5*/ ip = jp;
    /* 6*/ &i = ip;

    /* 7*/ (*ip)++;
    /* 8*/ *ip *=i;
    /* 9*/ ip++;
    /*10*/ tp = t+2;
    /*11*/ j = &t[5] - tp;
    /*12*/ t++;

    /*13*/ (*t)++;
    /*14*/ *tp--;
    /*15*/ j = (*tp)++;
    /*16*/ i = *tp++;
    /*17*/ p1 = ip;
    /*18*/ jp = p1;

    /*19*/ (*p1)--;
    /*20*/ dp = &i;
    /*21*/ dp = ip;
    /*22*/ jp = p2;
    /*23*/ p2 = p1;
    /*24*/ *p2 += i;
}
```

Oefening 20

Wat wordt er uitgeschreven? Uiteraard beantwoord je deze vraag zonder computer.
Bijvraag: hoe zou je de hardgecodeerde bovengrenzen van de drie for-lussen kunnen vervangen door zelf de grootte van de array's te bepalen?

```
#include <stdio.h>
int main(){
    int t[6] = {0,10,20,30,40,50};
    int* pt[3];

    int i;
    for(i=0; i<3; i++){
        pt[i] = &t[2*i];
    }

    pt[1]++;
    pt[2] = pt[1];
    *pt[1] += 1;
    *pt[2] *= 2;
    int ** ppt = &pt[0];
    (*ppt)++;
    **ppt += 1;

    for(i=0; i<6; i++){
        printf("%d ",t[i]);
    }
    printf("\n");
    for(i=0; i<3; i++){
        printf("%d ",*pt[i]);
    }
    printf("\n");
    return 0;
}
```

Oefening 21

Zorg dat je oefening 13 onder de knie hebt, voor je hieraan begint.

1. Schrijf een functie met naam `plaats_van(...)`, die de (meest linkse) plaats teruggeeft waarop een gegeven reëel getal in een gegeven array van reële getallen gevonden wordt. Bepaal zelf aantal en type van de parameters. Het returntype van deze functie is een pointer (naar een niet-constante)! Indien het getal niet aanwezig is, wordt de nullpointer teruggegeven. Gebruik verplicht schuivende pointers in plaats van indexering om de array te doorlopen.
2. Schrijf een hoofdprogramma om je functie uit te testen. Probeer alle randgevallen: zoek het eerste getal, het laatste getal, en een getal dat niet in de array voorkomt.
3. Schrijf een procedure met naam `plaats_ptr_op_getal(...)`, die een gegeven pointer verplaatst naar de plaats waarop een gegeven reëel getal in een gegeven array van reële getallen gevonden wordt. Indien het getal niet aanwezig is, wordt de gegeven pointer de nullpointer. Gebruik de functie `plaats_van` *niet*; gebruik schuivende pointers in plaats van indexering.
4. Breid je hoofdprogramma uit: verander één van de getallen uit de gegeven array in zijn tweevoud, door eerst de procedure `plaats_ptr_op_getal` op te roepen en dan het gevonden getal (in de array!) te verdubbelen. Schrijf ter controle de hele array uit. (Wat zou er gebeuren als je een getal dat niet aanwezig is in de array wil verdubbelen? Voorziet jouw code dat er dan een foutboodschap komt in plaats van een crash?)
5. Schrijf een functie met naam `plaats_in_geordende_array_van(...)`, die hetzelfde doet als de functie `plaats_van` maar die er vanuit gaat dat de meegegeven array niet-dalend gesorteerd is. (De functie test niet of dit ook effectief het geval is; het is de verantwoordelijkheid van de gebruiker om deze functie enkel voor niet-dalend gesorteerde arrays op te roepen!) Uiteraard zorg je voor een efficiënte oplossing!

Oefening 22

Gegeven onderstaand hoofdprogramma, en de functies `som`, `product` en `verschil`. Je ziet dat de derde tabel wordt ingevuld aan de hand van de twee eerste tabellen. Afhankelijk van de laatste parameter van de procedure `vul_tabel`, bevat de derde tabel de som, respectievelijk product of verschil van de overeenkomstige elementen uit de eerste twee tabellen.

```
#include <stdio.h>
#define AANTAL 5
int som(int a, int b){
    return a+b;
}
int product(int a, int b){
    return a*b;
}
int verschil(int a, int b){
    return a-b;
}
void schrijf(const int * t, int aantal){
    int i;
    for(i=0; i<aantal; i++){
        printf("%i ",t[i]);
    }
    printf("\n");
}
```

```
int main(){
    int a[AANTAL];
    int b[AANTAL];
    int c[AANTAL];
    int i;
    for(i=0; i<AANTAL; i++){
        a[i] = 10*i;
        b[i] = i;
    }

    vul_tabel(a,b,c,AANTAL,som);
    schrijf(c,AANTAL);

    vul_tabel(a,b,c,AANTAL,product);
    schrijf(c,AANTAL);

    vul_tabel(a,b,c,AANTAL,verschil);
    schrijf(c,AANTAL);
    return 0;
}
```

Als output zal er dus verschijnen:

```
0 11 22 33 44
0 10 40 90 160
0 9 18 27 36
```

Schrijf de procedure `vul_tabel(...)`. De laatste parameter is een functie. Zet de procedure `vul_tabel` ná je `main`-functie. Dan moet je `vul_tabel` vooraf declareren. Laat in die parameterlijst de benamingen van de parameters zelf weg, schrijf enkel de types neer.

Oefening 23

Schrijf een procedure `my_toupper(s)` die de eerste letter van de gegeven c-string `s` omzet naar een hoofdletter, en de andere letters naar een kleine letter. Cijfertekens en leestekens worden niet beïnvloed. Je mag er vanuit gaan dat het eerste teken een letter is. Gebruik geen functies uit de cstring-bibliotheek, maar doe de aanpassingen zelf. (Tip: als een karakterteken tussen 'a' en 'z' ligt, zorg je dat zijn ASCII-waarde verhoogd/verlaagd wordt met het gewenste getal.) Loop de c-string af met schuivende pointers. Test uit met een hoofdprogramma waarin je het woord `snEEuwwITJE<3!!` laat omzetten naar `Sneeuwwitje<3!!`. Daarna test je ook uit met een woord dat je inleest van het toetsenbord.

Oefening 24

Gegeven onderstaand hoofdprogramma. Schrijf de nodige code om dit te laten werken; gebruik overal **verplicht schuivende pointers**.

1. de procedure `verzetNaarEersteHoofdletter(p)` verzet de gegeven pointer `p` zodat hij wijst naar de eerste hoofdletter die vanaf zijn huidige positie te vinden is. Indien er geen hoofdletters meer volgen, staat `p` op het einde van de c-string.
2. de functie `pointerNaarEersteKleineLetter(p)` geeft een pointer terug naar de eerste kleine letter die te vinden is vanaf de huidige positie van de pointer `p`. (Ook hier: voorzie de situatie waarbij er geen kleine letters meer volgen.)

3. de procedure `schrijf(begin,eind)` schrijft de letters uit die te vinden zijn tussen de plaats waar de pointers `begin` en `eind` naar wijzen. (Laatste grens niet inbegrepen; je mag er vanuit gaan dat beide pointers in dezelfde c-string wijzen, en dat `begin` niet na `eind` komt.)

```
int main(){
    const char zus1[50] = "sneeuwWITje";
    const char zus2[50] = "rozeROOD";
    const char* begin;
    const char* eind;
    begin = zus1;
    verzetNaarEersteHoofdletter(&begin);
    eind = pointerNaarEersteKleineLetter(begin);
    schrijf(begin,eind);   /* schrijft 'WIT' uit */
    printf("\n");
    begin = zus2;
    verzetNaarEersteHoofdletter(&begin);
    eind = pointerNaarEersteKleineLetter(begin);
    schrijf(begin,eind);   /* schrijft 'ROOD' uit */
    return 0;
}
```

Oefening 25

Schrijf een procure `wis(s)` die uit de string `s` alle tekens wist die geen kleine letter of geen white space zijn. Je mag gebruik maken van de functies `islower(char)` en `isspace(char)`. Test uit met de string `"8d'a7!<t-)>+. -)4h&!e9)b*( )j'(e)!4\n8g|'92o!43e5d/.' 2 3g*(e('d22a'(a25n'(".` Test ook uit met een zinnetje dat je inleest (gebruik spaties in je input).

Oefening 26

Deze oefening geldt als extra uitdaging. Te maken als je (redelijk) vlot door alle oefeningen heen fietste.

Schrijf een procedure `pivoteer(begin,na_einde,pivot)`. De drie parameters zijn pointers naar karakters in dezelfde array. De pointer `pivot` wijst naar een element tussen de elementen waar de pointers `begin` en `na_einde` naar wijzen. De procedure wisselt de elementen symmetrisch rond de pivot. Ligt de pivot niet netjes in het midden tussen de grenzen `begin` en `na_einde`, dan wordt het wisselen beperkt. Een voorbeeld: indien de elementen vanaf `begin` tot net voor `na_einde` gelijk zijn aan `a b c d e f g h` en `pivot` wijst naar de letter `c`, dan wordt dit rijtje na afloop van de procedure `e d c b a f g h`.

Schrijf en procedure `schrijf(begin,na_einde)`. De twee parameters zijn pointers naar karakters in dezelfde array. De procedure schrijft alle elementen in de array uit, te beginnen bij `begin` en eindigend net voor `na_einde`. Gebruik schuivende pointers.

Controleer met het hoofdprogramma hieronder. Is de uitvoer van je programma geen leesbare uitspraak, dan schort er nog wat aan.

```
main(){
    char tekst[] = {'b','d','?','z','g','o','e','z','e','b',
                    ' ','d','i','g','!','h','o','s','v'};
    pivoteer(tekst+7,tekst+12,tekst+9);
    schrijf(tekst+4,tekst+15);
}
```

REEKS 4

Structs, malloc

Oefening 27

Schrijf een programma dat je bewaart onder de naam `begroet.c`. Op de commandolijn zal je, na de naam van het programma, een aantal persoonsnamen opgeven. Het programma antwoordt dan met een groet aan alle vermelde personen. Elke naam moet met een hoofdletter starten, zoals in oefening 23 (hergebruik code). Voorbeeld: als er op de commandolijn

`begroet oriana thomas han`

ingegeven wordt, komt er als reply

```
Dag Oriana!
Dag Thomas!
Dag Han!
```

Worden er geen namen meegegeven, dan komt er `Dag allemaal!`.

Oefening 28

Werk verder op vorige oefening (maak een kopie van je code, en haal weg wat je niet nodig hebt). Indien het programma op de commandolijn wordt opgeroepen zonder parameters, dan komt er nog altijd `Dag allemaal!`. In het andere geval begroet je enkel de persoon waarvan de naam alfabetisch het eerst komt. Dat zou in het voorbeeld van vorige opgave dus `Han` zijn. Schrijf daartoe een hulpfunctie `alfab_kleinste(voornamen,n)` die de c-string teruggeeft die in de array van c-strings (met naam `voornamen` en lengte `n`) alfabetisch eerst staat.

Oefening 29

Gegeven onderstaande code, waar een struct met naam `persoon` gedeclareerd wordt. Elke persoon onthoudt zijn naam (maximum 80 letters) en zijn leeftijd (geheel getal). Declareer een array van 6 personen. Vul hun namen en leeftijden in; die vind je (in volgorde) in de arrays `namen` en `leeftijden`. Schrijf daarna 6 regels uit van de vorm "`Evi is 21 jaar oud.`".

```
#include <stdio.h>
#include <string.h>
#define AANTAL 6
#define LENGTE ... /* aan te vullen */

typedef struct{
char naam[LENGTE];
int leeftijd;
}persoon;

int main(){
    int i;
    char * namen[AANTAL] = {"Evi","Jaro","Timen","Youri","Ashaf","Jennifer"};
    int leeftijden[AANTAL] = {21,30,18,14,22,19};

    /* ... */

    return 0;
}
```

Oefening 30

Ga voort op vorige oefening. Daar heb je ondertussen drie arrays, met elementen van type `char*`, type `int` en type `persoon`. We willen de elementen van deze arrays uitschrijven, gescheiden door een leesteken naar keuze. Omdat de arrays elk van een ander type zijn, zouden we drie keer bijna dezelfde code moeten schrijven: een lus om de elementen van een array te overlopen en die uit te schrijven. Dat dubbel werk kunnen we vermijden.

Ga daarvoor als volgt te werk:

1. Schrijf drie procedures `schrijf_cstring`, `schrijf_int` en `schrijf_persoon` die alledrie één parameter van pointertype meekrijgen en één element uitschrijven. (De persoon met naam `Evi` en leeftijd `21` wordt uitgeschreven als `Evi (21)`.) Test uit.
2. Schrijf de procedure

   ```
   void schrijf_array(const void * array, int aantal, int grootte, char tussenteken,
                      void (*schrijf)(const void*))
   ```

   die de eerste `aantal` elementen uit de array `array` uitschrijft. Elk koppel elementen wordt gescheiden door het opgegeven `tussenteken`. De parameter `grootte` bevat de grootte van het type dat uitgeschreven moet worden. De parameter `schrijf` is een pointer naar een passende uitschrijfprocedure.

Schrijf een hoofdprogramma dat volgende output produceert:

```
21+30+18+14+22+19

Evi;Jaro;Timen;Youri;Ashaf;Jennifer

Evi (21)
Jaro (30)
Timen (18)
Youri (14)
Ashaf (22)
Jennifer (19)
```

Belangrijke voetnoot met het oog op de labotest. Zorg dat je deze oefening helemaal binnenste buiten draait en jezelf grondig bevraagt over het hoe en waarom van welke parametertypes, sterretjes, haakjes,... Een dergelijke vraag komt dikwijls op testen voor, waarbij je soms ook zelf de hoofding van de procedure moet schrijven.

Oefening 31

1. Schrijf een functie `lees()` die een lijn inleest vanop het klavier, en een nieuwe c-string teruggeeft. Je weet niet hoe lang de tekst is, maar na oproep van de functie neemt de tekst niet meer geheugenplaats in dan strikt noodzakelijk. Indien de tekst langer is dan 1000 karaktertekens, breek je het af na het 1000^e. Test dit uit door de constante 1000 aan te passen - uiteraard.
2. Test je functie uit in het onderstaande (half-afgewerkte) hoofdprogramma. Vervolledig het hoofdprogramma.

```
int main(){
    int i;
```

```
    for(i=0; i<5; i++){
        char * tekst = lees();
        printf("Ik las in %s.\n",tekst);
    }
    return 0;
}
```

Oefening 32

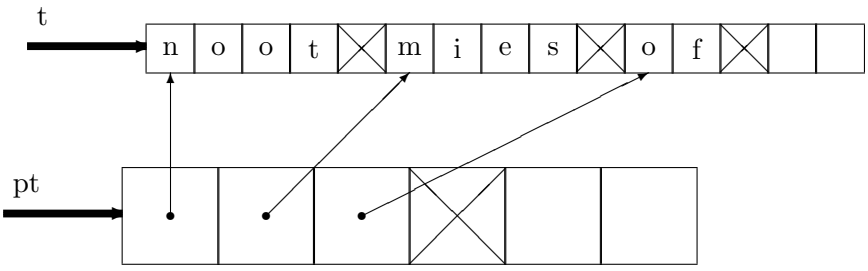
Werk verder op vorige oefening (maak een kopie).

1. Schrijf een functie `lees_meerdere()` die maximaal 6 lijnen tekst inleest vanop het klavier, tot de gebruiker de regel `STOP` intikt. Alle lijnen tekst (behalve `STOP`) worden teruggegeven. Merk op: de functie vraagt hoogstens `MAXAANTAL=6` lijnen tekst aan de gebruiker.
   Voorzie net voldoende plaats voor elke tekst, en net voldoende plaats voor alle teksten (dat zijn er maximaal 6). Na de laatste tekst bewaar je een nullpointer.
2. Test je functie uit in een hoofdprogramma: lees in en schrijf alles weer uit. Uiteraard laat je geen 1000 karaktertekens toe, dat kan je niet controleren. Stel die constante in op 4. Denk eraan om bij het uitschrijven elke tekst te laten volgen door een speciaal karakter (bijvoorbeeld `'!'`), zodat je kan nagaan of er niet teveel witruimte achteraan je tekst plakt.

Oefening 33

In deze oefening lezen we een aantal (nl. 10) woorden in, en slaan deze op in één lange array van karaktertekens. We kunnen toch aan alle woorden apart, als we (zie schets hieronder):

1. elk woord netjes afsluiten met een nullkarakter (dat zal automatisch gebeuren als we `scanf(...)` juist gebruiken), **EN**
2. de start van elk woord onthouden door er een pointer naar te laten wijzen, **EN**
3. toegang hebben tot de onderste tabel uit de schets (toegang tot de bovenste tabel is dan niet meer nodig).



Nog één opmerking: de onderste tabel heeft als laatste element een nullpointer. Dat zal ervoor zorgen dat de schrijfprocedure niet per se moet weten hoe lang de uit te schrijven (pointer)array is. Vul de gegeven code aan waar gevraagd. Ga voorlopig nog niet na of de gebruiker geen woorden opgeeft die te lang zijn.

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

```
#define AANTAL_WOORDEN            10
#define GEMIDDELDE_LENGTE_WOORDEN  7
#define LENGTE_ARRAY_T           ...
         (gebruik AANTAL_WOORDEN en GEMIDDELDE_LENGTE_WOORDEN om dit te berekenen)
         (voorbeeld: AANTAL_WOORDEN + GEMIDDELDE_LENGTE_WOORDEN)

void lees(...){
    ...         /* gebruik schuivende pointers, geen indexering */
}

void schrijf(...){
    ...
}

int main(){
    char* pt[AANTAL_WOORDEN+1];  /* zodat je ook nog een nullpointer
                                    kan wegsteken op het einde van de
                                    pointertabel */
    char t[LENGTE_ARRAY_T];

    pt[0] = t;

    printf("Geef %d woorden in:\n",AANTAL_WOORDEN);
    lees(pt);     /* leest alle woorden in                */
    schrijf(pt);  /* schrijft alle woorden onder elkaar uit*/

    return 0;
}
```

Extra
Er werd nog niet expliciet nagegaan of de gebruiker geen woorden opgeeft die te lang zijn. Dit kan je op twee manieren inbouwen.

1. Leg bij elk woord dat je inleest dezelfde beperking op; lees bijvoorbeeld niet meer dan `GEMIDDELDE_LENGTE_WOORDEN` in.
2. Leg bij elk woord dat je inleest een variabele beperking op; namelijk het aantal elementen dat nog vrij is in de array.

De eerste optie heeft het nadeel dat je allicht veel 'overschot' (niet-gebruikte elementen) hebt in de array t. De tweede optie heeft het nadeel dat de gebruiker als eerste woord een heel lang woord kan ingeven - waarna er geen plaats meer is voor de volgende. Pas de oefening daarom als volgt aan, gebruik makend van de tweede manier om in te lezen:

Schrijf een programma dat woorden inleest, tot de gegeven array `t` van lengte `LENGTE_ARRAY_T` vol is. Laat de gebruiker telkens weten hoe lang het op te geven woord maximaal mag zijn. Geeft de gebruiker een woord op waardoor de tabel overvol raakt, dan zorg je ervoor dat enkel het eerste deel van dat woord bewaard wordt.

Oefening 34

Loop niet te snel in deze oefeningen, volg de opdracht nauwgezet. (Dus geen functies/procedures waar het niet staat.)

1. Definieer een struct `Deeltal` met drie velden: `waarde`, `aantal_delers` en `delers`. De `waarde` zal een geheel getal bevatten; `aantal_delers` geeft aan hoeveel delers (tussen 1 en `waarde`-1) dit getal heeft; `delers` is een pointer naar `int`(s), hier zullen de delers te vinden zijn.
2. Werk dit deel uit in het hoofdprogramma.
   Maak lokaal een variabele van het type `Deeltal` aan. Vul op de volgende regels de drie velden van deze variabele in: `waarde` wordt 6, `aantal_delers` wordt 3 en de drie delers zijn 1, 2 en 3. Gebruik hiervoor geen lus.
3. Schrijf een procedure `schrijf_ints(...)` die een array van `int`s meekrijgt, en deze gehele getallen naast elkaar uitschrijft met een liggend streepje tussen.
4. Gebruik de bovenstaande procedure in de procedure `schrijf_deeltal(d)` die een deeltal (=getal en zijn delers) uitschrijft. Voor het getal 6 zou er komen:

   ```
   6  1-2-3
   ```

   Merk op: C laat geen hergebruik van functienamen toe (C++ wel)! Heb je in de schrijf-procedure(s) een kopie gemaakt, of ben je zuiniger geweest?

Oefening 35

1. Schrijf de functie `aantal_delers_van(x)` die telt hoeveel delers het geheel getal `x` heeft. (Overloop hiervoor alle getallen tussen `x` en `x/2`.)
2. Schrijf een functie `delers_van(x,aantal)` die een pointer teruggeeft naar een array die alle delers bevat van het geheel getal `x`. Test uit door in je hoofdprogramma de delers van 6 niet handmatig in te geven.
3. Schrijf een procedure `vul_aan(g)` die een gegeven `Deeltal` juist initialiseert. Het eerste veld van `g` (nl. `waarde`) is goed ingevuld bij aanroep van de procedure; de andere velden van `g` worden in de procedure aangevuld.
4. Schrijf een procedure `lees_deeltal(g)` die de velden van een deeltal invult: het veld `waarde` wordt hierbij ingelezen vanop het toetsenbord; de andere velden worden automatisch berekend.
5. Schrijf een procedure `lees_deeltallen(t,aantal)` die een `aantal` deeltallen inleest. Gebruik bovenstaande procedure (uiteraard).
6. Schrijf een procedure `schrijf_deeltallen(t,aantal)` die een array `t` van deeltallen uitschrijft. Overloop de array met indexering, en gebruikt een hulpprocedure om elk deeltal apart uit te schrijven.
7. Test voorgaande procedures uit in een hoofdprogramma: lees één deeltal in en schrijf het uit. Vraag de gebruiker nadien om een aantal op te geven, waarna je hem/haar exact zoveel deeltallen vraagt. Deze schrijf je ook uit.
8. Schrijf een functie `zoek(int waarde, Deeltal const * ptr, int aantal)` die een pointer teruggeeft naar het deeltal met waarde `waarde` in de array aangegeven door `ptr`. (Wat geef je terug indien het gezochte niet aanwezig is?) Respecteer de types in de parameterlijst.
9. Schrijf een of meerdere procedures die memory-leaks voorkomen. Aantal en aard van de parameters bepaal je zelf. Tip: voor je de geheugenplaats van een deeltal weer vrijgeeft aan het geheugen, schrijf je de waarde van dat deeltal uit. Zo kan je zelf nakijken of alle deeltallen zijn vrijgegeven.

REEKS 5

Gelinkte lijsten en bit fiddling

Oefening 36

```
typedef struct Knoop Knoop;
struct Knoop{
    int d;
    Knoop * opv;
};

int main(){
    Knoop * l = 0;
    Knoop * k = l;
    k = (Knoop*) malloc(sizeof(Knoop));
    return 0;
}
```

Teken de twee pointers `l` en `k`, en wat er precies gebeurt in bovenstaande code. Is `l` gewijzigd?

Oefening 37

```
typedef struct Knoop Knoop;
struct Knoop{
    int d;
    Knoop * opv;
};
int main(){
    Knoop * l = 0;
    voeg_vooraan_toe(7, ... );
    voeg_vooraan_toe(3, ... );
    print_lijst( ... );
    return 0;
}
```

Schrijf de nodige procedures, opdat bovenstaand hoofdprogramma zou werken. Op de puntjes geef je de lijst `l` mee. Denk na over de manier waarop je dat doet.

1. De procedure `voeg_vooraan_toe` breidt de gegeven lijst (tweede parameter) uit met één knoop: die komt vooraan, en bevat het gegeven getal (eerste parameter).
2. De procedure `print_lijst` implementeer je eerst niet-recursief.
3. Implementeer de procedure `print_lijst` daarna recursief.
4. Er ontbreekt een procedure aan het hoofdprogramma. Schrijf hoofding, implementatie en aanroep van deze procedure.

Oefening 38

```
int main(){
    srand(time(NULL));
    knoop * l = maak_gesorteerde_lijst_automatisch(10,100);
    print_lijst(l);
    printf("\nnu worden dubbels verwijderd: \n");
    verwijder_dubbels(...); /* aan te vullen */
```

```
    printf("\nna verwijderen van dubbels: \n\n");
    print_lijst(l);
    ... /* aan te vullen */
    return 0;
}
```

In bovenstaand hoofdprogramma wordt er automatisch een stijgend gesorteerde lijst aangemaakt, waar dubbele elementen in kunnen zitten. Implementeer de functie `maak_gesorteerde_lijst_automatisch(aantal,bovengrens)`. De eerste parameter geeft aan hoeveel elementen de lijst moet bevatten. De tweede parameter geeft aan wat het grootste getal zal zijn. Een tip: bouw de lijst op door vooraan telkens een iets kleiner getal dan het vorige toe te voegen. Daarvoor genereer je een getal uit de verzameling $\{0, 1, 2\}$ met de functie `rand()` uit `stdlib.h`.

Schrijf de procedure `verwijder_dubbels(...)` die alle dubbels uit de gelinkte lijst verwijdert. Als enige parameter geef je de gelinkte lijst mee. Vul het hoofdprogramma verder aan zoals het hoort.

Oefening 39

Werk voort op voorgaande oefening. Schrijf een procedure `keerom(lijst)` die een gegeven lijst volledig omkeert: de eerste knoop zit nu vooraan, de laatste achteraan. Test uit.

Oefening 40

Werk voort op voorgaande oefening. Het hoofprogramma wordt nu:

```
int main(){
    srand(time(NULL));
    knoop * m = maak_gesorteerde_lijst_autom(10,1000);
    knoop * n = maak_gesorteerde_lijst_autom(5,1000);
    printf("\nLIJST m:\n");    print_lijst(m);
    printf("\nLIJST n:\n");    print_lijst(n);
    printf("\nDeze worden gemerged. \n\n");

    knoop * mn = merge(...,...);

    printf("\nLIJST m:   \n"); print_lijst(m);
    printf("\nLIJST n:   \n"); print_lijst(n);
    printf("\nRESULTAAT: \n"); print_lijst(mn);
    ....   /* aan te vullen */
    return 0;
}
```

Schrijf de functie `merge` die twee gerangschikte lijsten als parameter neemt. Beide lijsten zullen op het einde leeg zijn; hun knopen zitten allemaal in de nieuwe (eveneens gerangschikte) lijst die teruggegeven wordt. Omdat de lijsten die je meegeeft gerangschikt moeten zijn (precondities van de functie), kan je de lijsten efficiënt samenvoegen. Neem hiervoor onderstaande code als voorbeeld.

```
int * merge(const int * a, const int * b, int size_a, int size_b){
    int i = 0; /* indexeert a */
    int j = 0; /* indexeert b */
    int k = 0; /* indexeert c */
    int size_c = size_a + size_b;
    int * c = (int*)malloc(size_c * sizeof(int));
```

```
    while(i < size_a && j < size_b){
        if( a[i] < b[j] ){
            c[k++] = a[i++];
       }
       else{
            c[k++] = b[j++];
       }
    }
    while(i < size_a){
        c[k++] = a[i++];
    }
    while(j < size_b){
       c[k++] = b[j++];
    }
    return c;
}
```

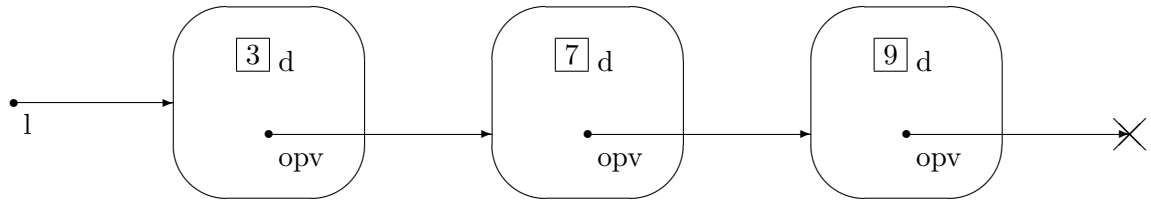
Oefening 41

Hieronder de procedure `voeg_getal_toe` die een knoop toevoegt aan een lijst die stijgend geordend is en moet blijven.

```
void voeg_getal_toe(int g, knoop **l) {
   knoop *h, *m;
   if (*l == 0 || g <= (*l)->getal) {
      h = (knoop*) malloc(sizeof(knoop));
      h->getal = g; h->next = *l; *l = h;
   }
   else {
      h = *l;
      while (h->next != 0 && h->next->getal < g)
         h = h->next;
      m = h->next;
      h->next = (knoop*) malloc(sizeof(knoop));
      h = h->next; h->getal = g; h->next = m;
   }
}
```

Je ziet dat er een opsplitsing nodig is in deelgevallen: een stuk code dat van toepassing is indien het toevoegen vooraan in de lijst gebeurt, en een stuk code dat alle andere gevallen behandelt. Dit dubbele werk willen we vermijden. Het is immers werk gespaard als je code kan schrijven die in elke situatie bruikbaar is.

De hulppointer `h` werd gedeclareerd als `knoop *` en duidt de knoop aan waar de nieuwe knoop aangehangen zal worden. Op tekening:



Als we het getal 4 willen toevoegen, zal `h` naar de eerste knoop wijzen. Als we het getal 10 toevoegen, zal `h` naar de laatste knoop wijzen. Als we het getal 2 willen toevoegen, zal `h` geen knoop hebben om naar te wijzen. (Vandaar de opsplitsing in gevallen in de code.)

Het valt echter op te merken dat je enkel `h->opv` nodig hebt in de code. De inhoud van de knoop waar `h` naar wijst (`h->d`) heb je niet nodig. Waarom dan naar de hele knoop (zowel `h->d` als `h->opv`) wijzen? Is toegang tot `h->opv` niet voldoende? Dat is de oplossing voor ons probleem: we laten een hulppointer `k` wijzen naar de horizontale pijlen (de links tussen de knopen). Willen we 4 toevoegen, dan wijst `k` naar de tweede horizontale pijl. Willen we 10 toevoegen, dan wijst `k` naar de laatste horizontale pijl. Willen we 2 toevoegen, dan wijst `k` naar de eerste horizontale pijl. En het behandelen van een speciaal geval is niet meer nodig!

In oefening 41 herschrijven we de procedure `voeg_getal_toe` die hierboven gegeven werd, in oefening 44 schrijven we de procedure `verwijder` die een knoop met bepaalde inhoud verwijdert. Omdat we in beide procedures eerst de plaats moeten bepalen waar de knoop bij of weg moet, **implementeer je nu de functie** `zoek(getal,lijst)` die een pointer teruggeeft waarmee je toegang hebt tot de plaats waar de knoop met inhoud `getal` staat (of zou moeten staan). Welk returntype gebruik je?

Oefening 42

Herschrijf de procedure `voeg_getal_toe(getal,lijst)` die een knoop met inhoud `getal` toevoegt aan een lijst die ondersteld wordt stijgend geordend te zijn (en te blijven). Dubbels zijn toegelaten. Maak hierbij gebruik van de functie `zoek(getal,lijst)` die je in oefening 41 schreef.

Oefening 43

Werk je sinds oefening 41 in een nieuw bestand? Heb je dan gedacht aan opkuisen van de lijst? Schrijf nu, als je dat nog niet deed, een *recursieve* versie van de procedure die een lijst volledig vrijgeeft en test uit. (Schrijf net voor het vrijgeven van een knoop, diens inhoud uit.)

Oefening 44

Schrijf een procedure `verwijder(x,lijst)` die — indien aanwezig — de knoop met inhoud `x` uit de stijgend gerangschikte lijst `lijst` verwijdert. Maak hierbij gebruik van de functie `zoek(getal,lijst)`.

Oefening 45

Om een idee te hebben van wat er zoal gevraagd kan worden op de test van C, volgt hieronder een ongecensureerde testvraag. Oplossingen worden niet gepubliceerd. Uiteraard heeft het geen zin aan deze oefening te beginnen als je alle vorige niet onder de knie hebt. Veel puzzelplezier!

Gegeven een tekst zoals

```
De Dit Er zon is was schijnt een eens door goede een het zin. prinses. raam. STOP
```

Als je deze zin woord voor woord inleest en elk woord beurtelings in één van drie gelinkte lijsten stopt, krijg je volgende drie gelinkte lijsten:

```
De zon schijnt door het raam.
Dit is een goede zin.
Er was eens een prinses.
```

Schrijf een functie `geef_array_van_lijsten(int aantal)` die een opsomming van woorden (afgesloten met STOP) inleest en beurtelings in één van '`aantal`' gelinkte lijsten stopt. De array (met lengte '`aantal`') die deze lijsten bevat, wordt teruggegeven. De woorden worden ingelezen vanop het scherm en zullen niet meer dan 80 letters bevatten, maar neem niet meer geheugenplaats in dan nodig.

Let op: om netjes bij te houden waar je in de gelinkte lijsten gebleven bent, houd je een array bij van precies '`aantal`' hulppointers, die ervoor zorgen dat je elke gelinkte lijst achteraan kan uitbreiden met een nieuwe knoop, zonder telkens de reeds opgebouwde lijsten van voor af aan te moeten doorlopen.

Oefening 46

Schrijf onderstaande functies en procedures, waarbij je enkel gebruik maakt van bitoperatoren. Gebruik een `typedef` zodat je `unsigned int` korter kan noteren als `uint`, en een constante `const int LENGTE = ...` die het aantal bits in een unsigned int bewaart.

1. De functie `bit_i(uint x, int i)` geeft het gehele getal 0 of 1 weer (type `int`), naargelang de inhoud van bit `i` van het getal `x`. Merk op: we zullen de laatste, minst significante bit, met het volgnummer 0 aanduiden. Test uit met enkele getallen.
2. Schrijf een procedure `schrijf(uint x, int lengte)` die gebruik maakt van bitoperatoren om het getal `x` uit te schrijven. Bij dit uitschrijven zullen er slechts `lengte` bits uitgeschreven worden (de minst significante). Test uit.
3. Schrijf de functie `eenbit(int i)` die de unsigned int teruggeeft die slechts één bit heeft aanstaan, nl. die met volgnummer `i`.
4. Schrijf de functie `aantal_eenbits(uint x)` die het aantal bits teruggeeft dat aanstaat in de binaire voorstelling van `x`. Werk niet via de functie `bit_i(x,i)`, dat is inefficiënt.
5. Schrijf de functie `bit_i_aangezet(uint x, int i)` die de unsigned int teruggeeft die gelijk is aan het gegeven getal x op hoogstens één bit na: de bit met volgnummer i staat aan.
6. Schrijf de functie `bit_i_uitgezet(uint x, int i)` die analoog werkt aan bovenstaande functie. Nu staat de i-de bit van het resultaat zeker uit.
7. Schrijf de functie `bit_i_gewisseld(uint x, int i)` die analoog werkt aan bovenstaande functie. Nu wordt de i-de bit veranderd van aan naar uit (of omgekeerd).
8. Schrijf de logische functie `zijn_gelijk(uint x, uint y)` die nul teruggeeft indien de parameters gelijk zijn. De `==`-operator mag niet gebruikt worden.
9. Schrijf de logische functie `is_even(uint x)` die 1 teruggeeft als het getal even is, anders 0. Gebruik enkel bitoperatoren, zelfs geen minteken.
10. Schrijf de procedure `wissel_even_met_oneven_bits(uint x)` die van een gegeven getal de bits op even posities wisselt met de bits op oneven posities. Dus `1011 0010` wordt `0111 0001`. Tip: maskeer bits op (on)even posities, schuif op, stel weer samen.
11. Welke test wordt er uitgevoerd door de code `n&(n-1)==0` ?
12. Schrijf een functie `product(int a, int b)` die het product van `a` en `b` bepaalt zonder gebruik te maken van de vermenigvuldigingsoperator `*`. Tip: denk aan cijferrekenen (techniek uit lagere school).