UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

LABO Programmeren in C en C++

Oefeningenbundel

PARTIM C

OPLOSSINGEN

Leen Brouns

Helga Naessens

Wim Van den Breen

Opleiding Industrieel Ingenieur Informatica / Elektronica / Automatisering

september 2017

REEKS 1

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

Oefening 1

```
#include <stdio.h>
#define AANTAL 15    /* te kiezen */

int main(){
    int i;
    printf("Hello world!\n");
    for(i=AANTAL; i>0; i--){
        printf("%d ",i);
    }
    printf("\nSTART");
    return 0;
}

/*  MERK OP
 *
 *  DECLARATIE VAN i
 *  C eist dat de teller buiten de lus gedeclareerd wordt. Te betreuren,
 *  maar in latere versies van C ook rechtgezet. Van zodra we in C++
 *  werken, wordt dit als NOT DONE beschouwd (en gequoteerd).
 *  Ook de plaats van declaratie is op de C-manier: vooraan in het
 *  programma. Eveneens NOT DONE beschouwd zodra we met C++ bezig zijn.
 */
```

Oefening 2

```
#include <stdio.h>

int main(){
    int i;
    for(i=0; i<=64; i++){
        printf("%4o %4d %4x\n",i,i,i);
    }
    return 0;
}
```

Oefening 3

```
/*
Testen op de teller binnen een for-lus is NOT DONE,
tenzij in het onwaarschijnlijke geval dat de test een ingewikkelde
berekening op die teller vraagt.
Maar testen of je aan het begin dan wel aan het einde van de lus zit,
DOE JE NIET.
Ga maar eens na wat deze overbodige testen voor de efficientie van je
programma betekenen, als we niet van 10 tot 1 maar van
10000000000000000000000000000 tot 1 laten aftellen!
*/
```

Oefening 4

```
/*  int macht = 1;
    while(macht < 10000){
        macht *= 2;
        printf("%d ",macht);  // TE LAAT
    }

De volgorde van derde en vierde regel moet omgewisseld worden.
Dit is heel eenvoudig te vermijden, door volgende tip toe te passen:


De VIER delen van de while-lus zijn:

DEEL 1       klaarzetten van het 'item' waarop getest wordt
DEEL 2       while (voorwaarde waar item aan moet voldoen om
                    door te gaan in de lus){
   DEEL 3           // hier komen alle zaken die wel herhaald moeten
                    // worden, maar niet tot DEEL 4 behoren
   DEEL 4           // opnieuw klaarzetten van het 'item' waarop getest
                    // wordt (dit kan eventueel meerdere regels beslaan)
             }

De JUISTE VOLGORDE om deze vier delen te schrijven zijn als volgt:

Eerst schrijf je DEEL 2.
Uit de opgave blijkt namelijk heel makkelijk hoelang je mag doorgaan met
herhalen, dus volgt er ook snel het 'item' waarop getest wordt.

Dan schrijf je DEEL 1 EN DEEL 4.
Deel 1 zet het item klaar (deftige initialisatie);
deel 4 is daar (dikwijls) quasi een kopie van.
Je zet deel 4 ook meteen ONDERAAN de lus.

Dan schrijf je DEEL 3.
Alles wat moet herhaald worden, maar niet tot deel 4 behoort, komt VOOR
deel 4, tussen de accolades.
*/
```

Oefening 5

```
/*
Als je een for-lus gebruikt, geef je de lezer het sein dat je op voorhand
weet hoe dikwijls je de code gaat uitvoeren.
Als je dan toch al van plan was om tussendoor van gedachten te veranderen
(zie de 'break' in de 'if'), had je dat beter op voorhand duidelijk
gemaakt door een (correcte) while-lus te gebruiken.

Dat is ook de bestaansreden van BEIDE lussen:
de for-lus gebruik je als het aantal herhalingen op voorhand vastligt,
de while-lus gebruik je in de andere gevallen.

Bovendien: de bovengrens 20 in de for-lus kan ondergedimensioneerd zijn
als je de grens 10000 aanpast.
Dat duidt er nogmaals op dat deze oplossing niet foolproof is.
(Ja, ze zou correcter met constanten kunnen werken,
maar dan nog blijft de opmerking over de 'break' die niet netjes is!)
*/
```

Oefening 6

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

/* Als x in RADIALEN staat, is sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
 *
 * LET OP! om zo min mogelijk zaken te herberekenen, bewaar je telkens
 *         de laatst gebruikte term (bvb -x^3/3!) en leid je daaruit de
 *         volgende term af.
 *
 *         Anders zou je veel berekeningen opnieuw doen.
 *         Vergelijk:
 *             om x^5/5! rechtstreeks uit te rekenen, heb je volgende
 *             bewerkingen nodig: x*x*x*x*x / (1*2*3*4*5)
 *
 *             om x^5/5! af te leiden uit -x^3/3! (het 'vorige
 *                                                 resultaat'),
 *             heb je volgende bewerkingen nodig:
 *             ('vorige resultaat') * (-x*x)/(4*5)
 *
 *         Het verschil tussen beide wordt alleen maar groter, hoe verder
 *         je in de reeksontwikkeling bent.
 */

/* ////////////////////////////////////////////////////////////////////// */
/* versie 1 */
int main1(){

    double x = 0.23;

    double teller = x;
    double noemer = 1;
    double resultaat = teller/noemer;
    int i=2;

    while(fabs(teller/noemer) > 0.00001){
        teller = teller * x * x * (-1);
        noemer = noemer * i * (i+1);
        resultaat += teller/noemer;
        i += 2;
    }

    printf("zelf berekende sin is %f\n",resultaat);
    printf("sin uit bib        is %f\n",sin(x));

    return 0;
}

/* ////////////////////////////////////////////////////////////////////// */
/* versie 2 */
int main2(){

    srand(time(NULL)); /* Laat je dit weg, dan genereert het programma bij elke run
        telkens hetzelfde getal.
                           Probeer uit!  */

    double x = rand()%6400/1000.0-3.2;  /* let op! vraag je 4 cijfers na de komma,
                                           dan krijg je geen gelijkmatige randomverdeling,
                                           omdat de bovengrens van rand() kleiner is dan
                                               64000 !!*/
```

```
    /* het vervolg is identiek - enkel de output werd wat beter verzorgd */
    double teller = x;
    double noemer = 1;
    double resultaat = teller/noemer;
    int i=2;

    while(fabs(teller/noemer) > 0.00001){
        teller = teller * x * x * (-1);
        noemer = noemer * i * (i+1);
        resultaat += teller/noemer;
        i += 2;
    }

    printf("zelf berekende sin van %.3f is %f\n",x,resultaat);
    printf("sin uit bib                  is %f\n",sin(x));

    return 0;
}

/* De rand-functie van C is niet geschikt voor doorgedreven statistische toepassingen!!
   Het bereik is te klein, en er valt wel wat aan te merken op de kwaliteit.
   In C++ heb je nieuwe bibliotheken die meer garanties bieden.
                                */

/* Bijlage bij versie 2:
   Hieronder vind je een mogelijke manier om zelf te controleren of het gebruik van rand()
   het gewenste effect heeft. Zo kan je bvb. zelf ontdekken dat de code
       double x = rand() % 64000 / 10000.0 - 3.2;
   niet het gewenste effect heef (nl. 3.1999 wordt nooit gegenereerd).
                            */

int main2bijlage(){
    int i;
    for(i=0; i<50; i++){
        double x = rand()%6400/1000.0-3.2;
        printf(" %.3f",x);             /* geeft al een eerste visuele indicatie*/
    }

    while(0==0){
        double x = (rand()%6400/1000.0)-3.2;
        if(x == -3.2) printf("*");       /* gaat na of de ondergrens bereikt wordt    */
        else if (x == 3.199) printf("+");/* gaat na of de bovengrens bereikt wordt    */
        else if (x == 0) printf (".");   /* beetje overbodige test; enkel voor de fun */
        else if (x<-3.2 || x >= 3.2){    /* gaat na of je niet buiten de grenzen valt */
            printf("  --> %f",x);
            break;                  /* FOEI! gebruik GEEN "break;" in je code
                                       (dit wijst op een quick-and-dirty-mentaliteit) */
        }                           /* Extra oefening: herschrijf zonder break.       */
    }
    return 0;
}

/* ////////////////////////////////////////////////////////////////////// */
/* versie 3 */

int main3(){

    double x;
    printf("Geef een reeel getal in; optimaal tussen -3.2 en 3.2: ");
    scanf("%lf",&x);
    printf("\nIk las in: %.3f\n",x);
```

```
    double teller = x;
    double noemer = 1;
    double resultaat = teller/noemer;
    int i=2;

    while(fabs(teller/noemer) > 0.00001){
        teller = teller * x * x * (-1);
        noemer = noemer * i * (i+1);
        resultaat += teller/noemer;
        i += 2;
    }

    printf("zelf berekende sin is %f\n",resultaat);
    printf("sin uit bib        is %f\n",sin(x));

    return 0;
}


/* ////////////////////////////////////////////////////////////////////// */
/* versie 4 */

double mijn_sin(double x){

    double teller = x;
    double noemer = 1;
    double resultaat = teller/noemer;
    int i=2;

    while(fabs(teller/noemer) > 0.00001){
        teller = teller * x * x * (-1);
        noemer = noemer * i * (i+1);
        resultaat += teller/noemer;
        i += 2;
    }

    return resultaat;
}

/* hoofdprogramma werd hier wat gewijzigd;
   werd niet expliciet gevraagd in opgave */
int main(){

    int i;
    for(i = -320; i< 320; i++){
        double x = i / 100.0;
        double mijn_sinus = mijn_sin(x); /* let op: geef variabelen nooit
                                            dezelfde naam als reeds bestaande functies*/
        double echte_sinus = sin(x);
        double verschil = mijn_sinus - echte_sinus;
        printf("\nzelf berekende sin van %.2f is %f, sin uit bib is %f; verschil is %f",
                                    x,mijn_sinus,echte_sinus,verschil);
    }

    return 0;
}
```

Oefening 7

```
#include <stdio.h>

int main(){
    const int AANTAL = 3;  /* voor testdoeleinden zet je dit klein genoeg! */

    int getal;
    int som = 0;
    int aantal = 1;
    printf("Geef een positief geheel getal: ");
    scanf("%d",&getal);
    while(aantal < AANTAL && getal >= 0){
        som += getal;
        aantal++;
        printf("Geef een positief geheel getal: ");
        scanf("%d",&getal);
    }
    if(getal < 0){
        printf("U gaf helaas geen %d positieve gehele getallen op.\n",AANTAL);
    }
    else{
        som += getal; /* verwerken van het laatste getal, dat ook positief was */
        printf("Dank, u gaf %d positieve gehele getallen op.\n",AANTAL);
    }
    printf("De som van de positieve getallen die u opgaf is %d.",som);

    return 0;
}
```

Oefening 8

```
#include <stdio.h>

#define INPUT 12345678


int cijfersom(int x){
    int som = 0;
    while(x>0){
        som += x%10;
        x /= 10;
    }
    return som;
}

int cijfersom_herhaald(int x){
    while(x > 9){
        x = cijfersom(x);
    }
    return x;
}

/* 'rec' staat voor 'recursief' */
int cijfersom_rec(int x){
    if(x<10) return x;
    return cijfersom_rec(x/10+x%10);
}
```

```
int main(){


    int i;
    for(i=0; i<3; i++){
        int getal=0;
        printf("\nGeef een strikt positief geheel getal op: ");
        while(scanf("%d",&getal)==0 || getal <= 0){
            while(getchar()!='\n'){
                printf(".");
            };
            printf("\nGeef een strikt positief geheel getal op: ");
        }
        int cijfersom_1 = cijfersom_herhaald(getal);
        int cijfersom_2 = cijfersom_rec(getal);

        printf("De cijfersom van %d is %d      ;",getal,cijfersom_1);
        if(cijfersom_1==cijfersom_2){
            printf(" wat gelijk is aan %d.",cijfersom_2);
        }
        else{
            printf(" wat HELAAS NIET gelijk is aan %d.",cijfersom_2);
        }
    }
    return 0;
}
```

Oefening 9

```
#include <stdio.h>

/* Recursieve versie

int faculteit(int x){
    if(x<2) return 1;
    return x*faculteit(x-1);
}
*/

int faculteit (int x){
    int i;
    int fac = 1;
    for(i = 2; i<= x; i++){
          fac *= i;
    }
    return fac;
}

void schrijf_faculteit(int x){
     printf("%d",faculteit(x));
}

int main(){
    int i;
    printf("5! =");
    schrijf_faculteit(5);

    printf("\n0! tot en met 40! geeft\n");
    for(i=0; i<=40; i++){
       schrijf_faculteit(i);
```

```
        printf("\n");
    }

    return 0;
}

/* MERK OP:
 *
 * Er verschijnen negatieve getallen in de lijst, te wijten aan overflow.
 *
 * Voor wie zich de vraag stelt: de snelheid van de rechtstreekse
 * berekening en de recursieve zijn niet merkbaar verschillend.
 * Enkel als de stack (= geheugenruimte die bijhoudt waar het
 * programma gebleven was met de recursie) vol begint te raken, zou je
 * vertraging kunnen zien. Maar tegen dan zijn we al lang de limiet voor
 * de overflow gepasseerd...
 *
 * Wil je tijdsverschillen echt afmeten tegen elkaar, dan zou je een
 * grote herhaling moeten inbouwen, en timen. Dit is hier echter niet
 * gevraagd - eerst zorgen dat alle oefeningen van reeks 1 af zijn.
 */
```

Oefening 10

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

int ggd(int a, int b){
    if(a<0 || b<0){
        return ggd(abs(a),abs(b));
    }
    if(a==0 || b==0){
        return a+b;    /* een van beide is nul; de ggd is de andere */
    }
    return ggd(b,a%b); /* indien a<b, zal dit gelijk zijn aan ggd(b,a)
                          en kan de recursie vanaf die stap de getallen
                          echt verkleinen */
}

int main(){
    if(ggd(-6,-8)==2)
        printf("ggd(-6,-8) ok\n");
    else
        printf("ggd(-6,-8) niet ok\n");
    if(ggd(24,18)==6)
        printf("ggd(24,18) ok\n");
    else
        printf("ggd(24,18) niet ok\n");
    if(ggd( 0,-5)==5)
        printf("ggd( 0,-5) ok\n");
    else
        printf("ggd( 0,-5) niet ok\n");
    if(ggd(6,-35)==1)
        printf("ggd(6,-35) ok\n");
    else
        printf("ggd(6,-35) niet ok\n");
    return 0;
}
```

```
/* Merk op:
 * de waarden voor het uittesten (zowel a,b als uitkomst) zouden we beter
 * in een array bewaren, zodat het uitprinten in een lus geschreven kan
 * worden. Dat kan je doen na volgende theorieles.
 * Het is wel belangrijk dat je bij het uittesten duidelijk laat merken
 * aan de gebruiker of de test geslaagd is (zodat hij zelf niet moet
 * zitten narekenen of de gegeven uitkomst juist is).
 * Meer over 'testing' komt in SO I aan bod - voor de studenten die deze
 * cursus op hun curriculum hebben staan.
 *
 * Nog een belangrijke opmerking: hier werden de accolades weggelaten
 * om plaats te sparen (omdat we geen lus gebruikten, hebben we teveel
 * duplicated code en dus heel veel plaats nodig).
 * Het weglaten van accolades is echter GEEN SLIMME ZET, het kan je
 * heel wat zoekwerk besparen als je ze WEL altijd plaatst (en/of als je
 * de automatische editeerfunctie van je ontwikkelomgeving gebruikt, die
 * de kantlijnen aanpast aan de code die je schreef).
 */
```

Oefening 11

```
/*
Wat hier op te merken viel:

blijkbaar KOPIEERT C de waarde van de meegegeven parameters naar de
variabelen die klaarstaan in de parameterlijst van de functie/procedure;
in het hoofdprogramma blijven de oorspronkelijke waarden dus ongewijzigd.
*/
```

REEKS 2

Arrays

Oefening 12

```
#include <stdio.h>

/* DEEL 2 */
/* Merk op:
 *    sizeof(tabel)/sizeof(int) geeft hier 2 als uitkomst!
 *    De verklaring vind je in de theorieles.
 */
void schrijf(const char tabel[], int aantal){  /* of (const char * tabel,...) */
    printf("IN PROCEDURE is sizeof(tabel)/sizeof(int) = %i\n",sizeof(tabel)/sizeof(int));
    int i;
    for(i=0; i<aantal; i+=2){
        printf("%c",tabel[i]);
    }
}


int main(){
    /* DEEL 1 */
    char letters [] =
       {'p','o','r','e','o','i','f','o','i','e','c','i','i',':','a','-','t','('};
    /* char * letters = {...} is NIET ok! */

    int lengte = sizeof(letters)/sizeof(char);
    printf("lengte van array met letters is %i\n",lengte);

    int i;
    for(i=0; i<lengte; i+=2){
        printf("%c",letters[i]);
    }

    /* DEEL 2 */
    printf("\nLetters uitschrijven via procedure\n");
    schrijf(letters,lengte);

    return 0;

}
```

Oefening 13

```
#include <stdio.h>

const int EPSILON = 0.00023; // een klein getal; om doubles te kunnen vergelijken
                             // (doubles vergelijk je nooit exact ! )

int index_van(const double array[], int aantal, double gezocht);

int main(){
    double rij []= {8.8,4.4,2.2,6.6,0.0,10.0};
    double x;
    printf("Geef een reeel getal op  ");
    scanf("%lf",&x);
```

```
    printf("\nJe gaf het getal %.1f op.",x);

    int lengte;
    lengte = sizeof(rij)/sizeof(double);

    int index = index_van(rij,lengte,x);
    if(index==-1){
        printf("\nHet getal %.1f werd niet gevonden in de niet-geordende array.",x);
    }
    else{
        printf("\nIn de niet-geordende array staat %.1f op plaats %d.\n",x,index);
    }

    return 0;
}

/*    Merk op: Je MOET stoppen zodra je vond wat gezocht werd.
               Je MAG NIET de hele array doorlopen
               als je al vond wat je zocht!! */
int index_van(const double array[], int aantal, double gezocht){
    int i = 0;
    while(i<aantal && fabs(gezocht-array[i]) > EPSILON){
        i++;
    }
    return (i<aantal ? i : -1);
}


/*  Wat er verandert als de array met zekerheid geordend is:
 *    Hier stop je eventueel nog vroeger, nl. als je blijkbaar al
 *    te ver in de array zit.
 */
```

Oefening 14

```
#include <stdio.h>

void schrijf(const char rij[],int aantal);
void schuif_links(char rij[], int lengte);

int main(){

    char rij[] = {'s','a','p','a','p','p','e','l'};
    int lengte = sizeof(rij)/sizeof(char);
    int i;

    schrijf(rij,lengte);

    for(i = 0; i < 3; i++){
        schuif_links(rij,lengte);
    }
    schrijf(rij,lengte);

    return 0;
}

void schrijf(const char rij[],int aantal){
     int i;
     for(i=0; i<aantal; i++){
         printf("%c",rij[i]);
```

```
     }
     printf("\n");
}

void schuif_links(char rij[], int lengte){
    char reserve = rij[0];
    int i;
    for(i=1; i<lengte; i++){
        rij[i-1] = rij[i];
    }
    rij[lengte-1] = reserve;
}
```

Oefening 15

```
#include <stdio.h>



int main(){

    srand(time(NULL)); /* anders krijg je altijd diezelfde getallen! */

    const int MIN = 100;
    const int MAX = 120;
    const int AANTAL = 20;

    int getal;
    int aanwezigheid[MAX-MIN+1];
    int i;
    for(i = 0; i < MAX-MIN+1; i++){ /* of in procedure */
        aanwezigheid[i] = 0;
    }

    /* let op! als je - ter controle - hier in de lus
       ook wil uitschrijven welk getal er effectief gegenereerd werd,
       zorg er dan voor dat je slechts EEN MAAL rand() oproept!!! */
    for(i = 0; i < AANTAL; i++){
        getal = MIN + rand()%(MAX-MIN+1);
        aanwezigheid[getal - MIN] = 1;
        printf("%d ",getal);
    }

    printf("\nKwamen niet voor:\n");
    for(i = 0; i < MAX-MIN+1; i++){
        if (aanwezigheid[i] == 0){
            printf("%d ",(i+MIN));
        }
    }

    return 0;
}
```

Oefening 16

```
#include <stdio.h>

void print_lijn(char c, int lengte);
void teken_horizontaal(const int* frequenties);
```

```
void teken_vertikaal(const int* frequenties);
int is_kleine_letter(char c);
int is_grote_letter(char c);
void lees_input(int* frequenties);

/*
Merk op: omdat het programma expliciet met het alfabet werkt dat 26
letters bevat, hebben we van het getal 26 geen constante gemaakt.
Anders zou een andere programmeur op het idee kunnen komen om die
constante een andere waarde te geven, waardoor het hele programma
in de soep draait.
*/

int main(){
    int frequenties[26] = {0};
    lees_input(frequenties);
    teken_horizontaal(frequenties);
    teken_vertikaal(frequenties);
    return 0;
}


void print_lijn(char c, int lengte){
    int i;
    for(i = 0; i < lengte; i++){
        printf("%c",c);
    }
}

void teken_horizontaal(const int* frequenties){
    int i;
    for(i = 0; i < 26; i++){
        printf("\n%c:   ",('a'+i));
        print_lijn('*',frequenties[i]);
    }
}

void teken_vertikaal(const int* frequenties){
    int grootste_frequentie = 0;
    int i;
    for(i = 0; i < 26; i++){
        if(grootste_frequentie < frequenties[i]){
            grootste_frequentie = frequenties[i];
        }
    }

    for(i = grootste_frequentie; i > 0; i--){
        printf("\n");
        int k;
        for(k = 0; k < 26; k++){
            if(frequenties[k] >= i){
                printf("%c",'a'+k);
            }
            else{
                printf(" ");
            }
        }
    }
}


int is_kleine_letter(char c){
```

```
    return ('a' <= c && c <= 'z');
}

int is_grote_letter(char c){
    return ('A' <= c && c <= 'Z');
}

void lees_input(int* frequenties){
    char letter;
    scanf("%c",&letter);
    while(letter != '$'){
        if(is_kleine_letter(letter)){
            frequenties[letter-'a']++;
        }
        else if(is_grote_letter(letter)){
            frequenties[letter-'A']++;
        }
        scanf("%c",&letter);
    }
}
```

Oefening 17

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

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

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

    return 0;
}
```

Oefening 18

```
#include <stdio.h>

void zoek_extremen(const int * rij, int lengte, int * min, int * max){
    int i;
    *min = rij[0];
    *max = rij[0];
    for(i=1; i<lengte; i++){
        if(*min > rij[i]){
            *min = rij[i];
        }
        else if(*max < rij[i]){
            *max = rij[i];
```

```
        }
    }
}

/* de printf-statements zijn NIET gevraagd, maar hebben wel een belangrijk
 * didactisch nut! */
void zoek_extremen_rec(const int * rij, int lengte, int * min, int * max){
    printf("IN recursie, met lengte=%i, min=%i, max=%i\n",lengte,*min,*max);
    if(lengte == 1){
        *min = rij[0];
        *max = rij[0];
    }
    else{
        zoek_extremen_rec(rij,lengte-1,min,max);
        if(rij[lengte-1] < *min){
            *min = rij[lengte-1];
        }
        if(rij[lengte-1] > *max){
            *max = rij[lengte-1];
        }
    }
    printf("UIT recursie, met lengte=%i, min=%i, max=%i\n",lengte,*min,*max);
}

main(){
    int rij [] = {5,7,9,6,4,2,3,8,5,-11,999,-11,5,4,2};
    /* geeft -11 voor min; 999 voor max */

    /* int rij [] = {-5,-2,-888,-1,-3,-9,-4,-8,-7};      */
    /* geeft -888 voor min; -1 voor max */

    int min,max;
    /* zoek_extremen(rij,sizeof(rij)/sizeof(int),&min,&max); */
    zoek_extremen_rec(rij,sizeof(rij)/sizeof(int),&min,&max);

    printf("min is %d,  max is %d",min,max);
}

/*
 *  Belangrijk: test ook met enkel negatieve getallen!
 *  Heb je opgemerkt dat de initialisatie van zowel min als max gebeurt
 *  met het EERSTE ELEMENT uit de array, in plaats van met de willekeurig
 *  gekozen waarde 0?
 *
 */
```

REEKS 3

pointers, functiepointers en C-strings

Oefening 19

```
/*
schrappen in rood: dan staat er 'compileert niet'
schrappen in blauw: dan staat er 'gevaarlijk' of 'zeer slordig'
*/

int main(){
            int i=7, j;
            double d;
            int *ip, *jp, *tp;
            double *dp;
            const int * p1;

            int t[25];
            int k;
            for(k=0; k<25; k++){
                t[k] = k*10;
            }

    /* 1*/  i = j;   /* gevaarlijk: i had een degelijk waarde; j bevat 'rommel'.
                        na deze actie weet je niet meer wat in i zit. Voor het
                        vlot vervolg van de oefening negeren we deze regel!       */
                   /* TIP: lees  '='  als 'krijgt de waarde van'                  */
    /* 2*/ jp = &i; /* ok: jp wijst naar geheugenplaats met naam i;
                           types kloppen                                          */
    /* 3*/ j = *jp; /* ok: j krijgt nu de waarde 7                                */
    /* 4*/ *ip = i; /* FOUT: ip is nog zwevende pointer; *ip is geen
                        gereserveerde geheugenplaats                              */

    /* 5*/ ip = jp; /* ok: ip wijst nu ook naar geheugenplaats met naam i         */
    /* 6*/ &i = ip; /* compileert niet: een adres kan je niet wijzigen            */
    /* 7*/ (*ip)++; /* ok: 7 wordt nu 8                                           */
    /* 8*/ *ip *=i; /* ok: 8 wordt nu 64                                          */
    /* 9*/ ip++;    /* gevaarlijk: ip wijst niet in een tabel;
                        de buur van ip is dus onbekend                            */

    /*10*/ tp = t+2;          /* ok: tp wijst nu naar derde element in de array t */
    /*11*/ j = &t[5] - tp;    /* ok: j wordt nu 5-2=3                             */
    /*12*/ t++;               /* compileert niet: array kan je niet opschuiven    */
    /*13*/ (*t)++;            /* ok: eerste element van de tabel verhoogt met 1   */
    /*14*/ *tp--;             /* zeer slordig; not done:
                                 eerst vraag je "*tp" op (onzin als het alleen
                                 op de regel staat),
                                 en dan schuif je pointer 1 naar voor
                                 (dat op zich kan wel)                            */

    /*15*/ j = (*tp)++;  /* ok: aangenomen dat tp in vorige regel een plaats naar
                              links opschoof in de array, krijgt j nu de waarde
                              van het tweede element van de array - waarna dat
                              tweede element met 1 verhoogt  (concreet: j is
                              nu 10, terwijl de array 0 11 20 30 ... bevat)    */
    /*16*/ i = *tp++;    /* ok: i krijgt de waarde van het tweede element in de
                             tabel  (concreet: 11),
                             tp schuift een plaats naar rechts in de tabel
                             (concreet: tp wijst naar 20)                     */
```

```
    /*17*/ p1 = ip;       /* ok: p1 is pointer naar const en belooft dus meer
                           dan ip deed                                        */
    /*18*/ jp = p1;       /* compileert niet of geeft warning en is ZEKER NOT
                           DONE: p1 is pointer naar const, terwijl
                           jp niet belooft om de geheugenplaats waarnaar-ie wijst
                           ongewijzigd te laten.                              */
                          /* warning: "assignment discards 'const' qualifier from
                           pointer target type [enabled by default]"          */
    /*19*/ (*p1)--;       /* compileert niet: p1 is een pointer naar const, dus
                           mag je via p1 geen wijzigingen aanbrengen aan *p1  */

    /*20*/ dp = &i;       /* compileert niet of geeft warning: pointer naar double
                           kan niet geinitialiseerd worden met adres van int  */
    /*21*/ dp = ip;       /* compileert niet of geeft warning: idem als
                           hierboven                                          */
    /*22*/ jp = p2;       /* ok: jp wijst nu naar element waar p2 naar wijst (p2
                               is een constante pointer dus verhuist zelf nooit,
                               maar heeft niet beloofd om de geheugenplaats
                               waar-ie naar wijst ongewijzigd te laten, dus
                               moet jp dat ook niet doen)                     */
    /*23*/ p2 = p1;       /* compileer niet of geeft warning: p2 is een constante
                           pointer, dus kan geen nieuwe waarde aannemen       */
    /*24*/ *p2 += i;      /* in orde; gezien p2 bij initialisatie naar i verwees
                           (regel 23 negeren we want is fout),
                           wordt de inhoud van i bij deze verdubbeld          */
    return 0;
}
```

Oefening 20

```
/*
Oplossing:
op het scherm verschijnt er

0 11 20 62 40 50
11 62 62


Oplossing van de bijvraag:

In de eerste en laatste for-lus kan je de bovengrens 3 vervangen door
    sizeof(pt)/sizeof(int*).
In de tweede for-lus kan je de bovengrens 6 vervangen door sizeof(t)/sizeof(int).
*/

#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)++;             /* merk op: *ppt is een pointer;
                             die wordt een plaats naar rechts opgeschoven */
    **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

```
#include <stdio.h>

/*    Merk op: hier werd 'const' bij de eerste parameter weggehaald.
                De reden: als we een niet-const pointer willen teruggeven
                naar een array, dan moeten we toegang hebben tot deze
                array aan de hand van een niet-const pointer!
                Dus omdat het returntype van de functie int*
                is in plaats van const int*, moet ook de eerste parameter
                type int* hebben.
                (Het returntype mag niet const int* zijn, omdat we
                de returnwaarde in het hoofdprogramma willen gebruiken
                om het gevonden getal te bewerken!)                    */
int* plaats_van(int * array, int aantal, int gezocht){
    int i = 0;
    while(i<aantal && gezocht != *array){
        array++;
        i++;
    }
    return (i<aantal ? array : NULL);
}

void plaats_ptr_op_getal(int ** dptr, int aantal, int gezocht){
    int i = 0;
    while(i<aantal && gezocht != **dptr){
        (*dptr)++;
        i++;
    }
    if (i==aantal) *dptr = NULL;
}

int* plaats_in_geordende_array_van(int * array, int aantal, int gezocht){
    int i = 0;
    while(i<aantal && gezocht > *array){
        array++;
        i++;
    }
    if(i == aantal) return NULL;
    if(gezocht == *array) return array;
    return NULL;
}
```

```
/* Merk op: om snel te controleren zonder inmenging van de gebruiker,
            zou je ook de waarde van x kunnen 'hardcoderen'. Dat kan tijd sparen bij het
                testen,
            maar hangt van persoonlijke voorkeur af.
                                                                    */
int main(){
    int rij []= {8,4,2,0,6,10};
    int x;
    printf("Geef een geheel getal op  ");
    scanf("%d",&x);
    printf("\nJe gaf het getal %d op.",x);

    int lengte;
    lengte = sizeof(rij)/sizeof(int);

    int * plaats = plaats_van(rij,lengte,x);
    if(plaats == NULL){
        printf("\nHet getal %d werd niet gevonden in de niet-geordende array.",x);
    }
    else{
        printf("\nIk vond het getal en vervang het door zijn tienvoud.\n");
        *plaats *= 10;
        int i;
        for(i=0; i<lengte; i++){
            printf("%d ",rij[i]);
        }
    }

    printf("\n\nVIERDE DEEL VAN DE OEFENING\n\nGeef een geheel getal op  ");
    scanf("%d",&x);
    printf("\nJe gaf het getal %d op.",x);

    int * ptr = rij; // zet de pointer klaar vooraan in de array
    plaats_ptr_op_getal(&ptr,lengte,x);
    if(ptr == NULL){
        printf("\nHet getal %d werd niet gevonden in de niet-geordende array.",x);
    }
    else{
        printf("\nIk vond het getal en vervang het door zijn tweevoud.\n");
        *ptr *= 2;
        int i;
        for(i=0; i<lengte; i++){
            printf("%d ",rij[i]);
        }
    }


    int rij_op_orde []= {2,4,6,8,10};
    printf("\n\nGeef een geheel getal op  ");
    scanf("%d",&x);
    printf("\nJe gaf het getal %d op.",x);

    int lengte_op_orde = sizeof(rij_op_orde)/sizeof(int);

    plaats = plaats_in_geordende_array_van(rij_op_orde,lengte_op_orde,x);
    if(plaats == NULL){
        printf("\nHet getal %d werd niet gevonden in de geordende array.",x);
    }
    else{
        printf("\nIk vond het getal en vervang het door zijn honderdvoud.\n");
        *plaats *= 100;
```

```
        int i;
        for(i=0; i<lengte_op_orde; i++){
            printf("%d ",rij_op_orde[i]);
        }
    }

    return 0;
}
```

Oefening 22

```
// MERK OP: in het hoofdprogramma (dat gegeven was) hebben we hier
//          een ampersand (&) toegevoegd bij het oproepen van de functies
    som/product/verschil.
//          Dat spoort beter met de signatuur van vul_tabel, waar je som/product/verschil
//          met een functiePOINTER opvangt.
//          Maar blijkbaar kan de compiler beide versies aan.

#include <stdio.h>
#define AANTAL 5

void vul_tabel(const int *, const int *, int *, int, int (*)(int,int) );

int som(int a, int b);
int product(int a, int b);
int verschil(int a, int b);

void schrijf(const int * t, int aantal);

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;
}


void vul_tabel(const int * t_1, const int * t_2, int * resultaat, int lengte, int
    (*bewerking)(int,int) ){
    int i;
    for(i=0; i<lengte; i++){
        resultaat[i] = bewerking(t_1[i],t_2[i]);
    }
}

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");
}
```

Oefening 23

```
#include <stdio.h>

void my_toupper(char * s){
    // hier zou je nog eerst if(s!=0) kunnen testen!
    // of in de precondities vermelden dat s noch zwevend,
    // noch nullpointer mag zijn
    if(*s >= 'a' && *s <= 'z'){
        *s = *s - 'a' + 'A';      /* Hier willen we GEEN mysterieuze */
    }                             /* getallen zoals 32 zien staan!   */
    s++;                          /* dat is onleesbaar!              */
    while(*s != 0){
        if(*s >= 'A' && *s <= 'Z'){
            *s = *s - 'A' + 'a';
        }
        s++;
    }
}

/* Merk op:
 * bovenstaande code is een procedure; dus apart op de regel oproepen.
 * Je zou dit kunnen omvormen naar een functie; in latere oefeningen
 * komt dit (nl. returntype char*) aan bod.
 */

int main(){
    /* char * woord = "snEEuwwITJE<3!!";              DIT MAG NIET */
    char woord[50] = "snEEuwwITJE<3!!";     /*         DIT MAG WEL */

    /* Laatste vraag: inlezen van toetsenbord: */
    /* char woord[50];
    printf("Geef een woord op: ");
    scanf("%49s",woord);
    */

    my_toupper(woord);
    printf("%s",woord);
    return 0;
}
```

Oefening 24

```
#include <stdio.h>


void verzetNaarEersteHoofdletter(const char ** l){
    while(**l!=0 && (**l<'A' || **l>'Z')){
        (*l)++;
    }
}

const char* pointerNaarEersteKleineLetter(const char* l){
    while(*l!='\0' && (*l<'a' || *l>'z')){
        l++;
    }
    return l;
}

void schrijf(const char* begin, const char* eind){
    while(begin!=eind){
        printf("%c",*begin);
        begin++;
    }
}

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);
    printf("\n");

    begin = zus2;
    verzetNaarEersteHoofdletter(&begin);
    eind = pointerNaarEersteKleineLetter(begin);
    schrijf(begin,eind);

    return 0;
}
```

Oefening 25

```
#include <stdio.h>

void wis(char * s);

int main(){

    /* mag je hier niet doen:
       char * woord = "...........";
       ZAL CRASHEN; GEEN COMPILEERFOUT    */
    char woord [] = "8d'a7!<t-)>+. -)4h&!e9)b*( )j'(e)!4\n8g|'92o!43e5d/.' 2
        3g*(e('d22a'(a25n'(";
    wis(woord);
    printf("%s",woord);
```

```
    char woord2[81];
    printf("\n\n\nGeef nu zelf een tekst in (met spaties);\n");
    printf("ik haal er alle stukken tussenuit die geen kleine letter of blanco zijn.\n");
    fgets(woord2,81,stdin);
    wis(woord2);
    printf("%s",woord2);

    return 0;
}


void wis(char * s){
    char * loper = s;
    while(*s!='\0'){
        if(islower(*s) || isspace(*s)){
            *loper = *s;
            loper++;
        }
        s++;
    }
    *loper = 0;
}
```

Oefening 26

```
#include <stdio.h>
void pivoteer(const char * begin, const char * na_einde, char * pivot){
    int i;
    int links = pivot - begin;
    int rechts = na_einde - pivot - 1;
    int min = links;
    if (min > rechts) min = rechts;
    for(i=1; i<=min; i++){
        char hulp = pivot[i];
        pivot[i] = pivot[-i];
        pivot[-i] = hulp;
    }
}

void schrijf(const char * begin, const char * na_einde){
    while(begin != na_einde){
        printf("%c",*begin++);
    }
}

int main(){
    return 0;
}
```

REEKS 4

Structs, malloc

Oefening 27

```
/* te bewaren onder de naam begroet.c
/* Op de command line komt er dan bvb
/*      begroet oriana thomas han
/* en als antwoord komt er dan
/*      Dag Oriana!
        Dag Thomas!
        Dag Han!                        */

/* Je kan ook via Dev-Cpp instellen dat de argumenten
/*      oriana thomas han
/* moeten doorgegeven worden, maar dat vraagt wat meer zoek- en klikwerk   */

#include <stdio.h>

void my_to_upper(char * s){
    if(*s >= 'a' && *s <= 'z'){
        *s = *s - 'a' + 'A';      /* Hier willen we GEEN mysterieuze */
    }                             /* getallen zoals 32 zien staan!   */
    s++;                          /* dat is onleesbaar!              */
    while(*s != 0){
        if(*s >= 'A' && *s <= 'Z'){
            *s = *s - 'A' + 'a';
        }
        s++;
    }
}

int main(int argc,char**argv){
    int i;

    for(i=1; i<argc; i++){
        my_to_upper(argv[i]);
    }

    if(argc > 1) {
        for(i=1; i<argc; i++){
            printf("Dag %s!\n",argv[i]);
        }

    }
    else printf("Dag allemaal!");
}
```

Oefening 28

```
#include <stdio.h>

void my_to_upper(char * s){
    if(*s >= 'a' && *s <= 'z'){
        *s = *s - 'a' + 'A';      /* Hier willen we GEEN mysterieuze */
    }                             /* getallen zoals 32 zien staan!   */
    s++;                          /* dat is onleesbaar!              */
```

```
    while(*s != 0){
        if(*s >= 'A' && *s <= 'Z'){
            *s = *s - 'A' + 'a';
        }
        s++;
    }
}

char* alfab_kleinste(char**woorden,int aantal){
    char* kl = woorden[0];
    int i;
    for(i=1; i<aantal; i++){
        if(strcmp(kl,woorden[i]) > 0){
            kl = woorden[i];
        }
    }
    return kl;
}

int main(int argc,char**argv){
    int i;
    char * kleinste;

    for(i=1; i<argc; i++){
        my_to_upper(argv[i]);
    }

    if(argc > 1) {
        kleinste = alfab_kleinste(argv+1,argc-1);
                        /* Merk op: sla het eerste element van argv over!
                           En doe dat HIER, het is niet de taak van de functie
                           alfab_kleinste om het eerste element over te slaan. */
        printf("Dag %s!",kleinste);
    }
    else printf("Dag allemaal!");
}
```

Oefening 29

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

#define AANTAL 6
#define LENGTE 81 /* wel degelijk 80+1: het nulkarakter verdient ook een plaats! */
                  /* test uit wat er gebeurt als je dit door 8 vervangt:
                     "Jennifer" telt 8 letters                                    */
typedef struct{
    char naam[LENGTE];
    int leeftijd;
}persoon;



int main(){

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

    persoon leerlingen[AANTAL];
```

```
    for(i=0; i<AANTAL; i++){
        strcpy(leerlingen[i].naam,namen[i]);
        leerlingen[i].leeftijd = leeftijden[i];
    }

    for(i=0; i<AANTAL; i++){
        printf("%s is %d jaar oud\n",leerlingen[i].naam,leerlingen[i].leeftijd);
    }

    return 0;
}
```

Oefening 30

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

#define AANTAL 6

typedef struct{
    char naam[81];
    int leeftijd;
}persoon;

void schrijf_persoon(const persoon * p){
    printf("%s (%d)",(*p).naam,(*p).leeftijd);
}

void schrijf_int(const int * a){
    printf("%d",*a);
}

void schrijf_cstring(const char *const * a){  /* !! DUBBEL STER !! */
    printf("%s",*a);
}


/* Meest voor de hand liggende oplossing is */
void schrijf_array___(void * tabel, int aantal, int grootte, char tussenteken, void
    (*schrijf)(const void*)){
    int i;
    schrijf(tabel);
    for(i=1; i < aantal; i++){
        printf("%c",tussenteken);
        schrijf(tabel + i*grootte);
    }
    printf("\n\n");
}

/* De meeste compilers geven hier echter - terecht - warnings op:
      'pointer of type 'void*' used in arithmetic'
   Andere compilers weigeren het helemaal.
   De reden: het is niet standaard gedefinieerd wat "+i" betekent
   bij een void-pointer.

   De oplossing: de void-pointer wordt gecast naar een char-pointer, die wel
   weet hoe er gerekend moet worden. Bovendien beslaat een 'char' de kleinste eenheid
   van geheugenplaats, en zijn alle andere hier veelvoud van.
   Dus kunnen we hetzelfde principe toepassen: de stapgrootte nog aanpassen
   aan de grootte van het type waarnaar verwezen wordt.
```

```
   Met deze ingreep zijn de warnings weg, en zijn we uit de illegaliteit...
   Zie ook
   https://stackoverflow.com/questions/33154318/how-to-traverse-an-array-parameter-as-void-pointer.
*/

void schrijf_array(void * tabel, int aantal, int grootte, char tussenteken, void
    (*schrijf)(const void*)){
    char* hulpptr = (char*)tabel;
    int i;
    schrijf(hulpptr);
    for(i=1; i < aantal; i++){
        printf("%c",tussenteken);
        schrijf(hulpptr + i*grootte);
    }
    printf("\n\n");
}

int main(){

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

    persoon leerlingen[AANTAL];
    for(i=0; i<AANTAL; i++){
        strcpy(leerlingen[i].naam,namen[i]);
        leerlingen[i].leeftijd = leeftijden[i];
    }

    schrijf_array(leeftijden,AANTAL,sizeof(int),    '+', (void (*)(const
        void*))schrijf_int);
    schrijf_array(namen,     AANTAL,sizeof(char*),  ';', (void (*)(const
        void*))schrijf_cstring);
    schrijf_array(leerlingen,AANTAL,sizeof(persoon),'\n',(void (*)(const
        void*))schrijf_persoon);

    /* merk op:
       (1) voor schrijf_int, schrijf_cstring en schrijf_persoon in bovenstaande 3 regels
           mag ook een & staan, maar het moet niet (compiler kan beide interpreteren)
       (2) voor leeftijden, namen, leerlingen in bovenstaande 3 regels
           mag nog (void*) staan (casten naar void-pointer), maar het moet niet
           (compiler kan hiermee weg).
               */

    return 0;
}
```

Oefening 31

```
#include <stdio.h>
#define MAX 4  /* MERK OP: werken met constanten brengt hier niet echt soelaas,
                           als je met scanf zou werken. Dus werk met fgets!    */

char* lees(){
    char tekst[MAX+1];          /* +1 !! */
    printf("Geef een tekst: ");
    fgets(tekst,MAX+1,stdin);
    int lengte = strlen(tekst);
```

```
    /* indien 4+1 karaktervelden voorzien, en 'aap\n' ingelezen */
    if(tekst[lengte-1] == '\n'){
        tekst[lengte-1] = 0;    /* of '\0' */
        lengte--;
    }
    /* indien 4+1 karaktervelden voorzien, en 'noot' ingelezen:
       alles ok; niets doen of lege lus                          */
    /* indien 4+1 karaktervelden voorzien, en 'appelmoes' ingelezen:
       'appe\0' bewaard; 'lmoes' moet nog weg                     */
    else{
        while (getchar() != '\n');
    }
    char * nieuw = malloc((lengte+1)*sizeof(char));
    strcpy(nieuw,tekst); /* of letter per letter... */
    return nieuw;
}

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

Oefening 32

```
#include <stdio.h>
#define MAXLENGTE 4  /* Werk je met scanf? Dan kan je niet met constanten
                        werken; dan moet het hardgecodeerd.    */
#define MAXAANTAL 3  /* nadien nog een nullpointer opslaan */


char* lees(){
    char tekst[MAXLENGTE+1];          /* +1 !! */

    printf("Geef een tekst: ");
    fgets(tekst,MAXLENGTE+1,stdin);
    int lengte = strlen(tekst);

    if(tekst[lengte-1] == '\n'){
        tekst[lengte-1] = 0;    /* of '\0' */
        lengte--;
    }
    else{
        int ch;
        while ((ch = fgetc(stdin)) != EOF && ch != '\n') {
            /* null body */;
        }
    }
    char * nieuw = malloc((lengte+1)*sizeof(char));
    strcpy(nieuw,tekst);
    return nieuw;
}

char** lees_meerdere(){
    int aantal=0;
    char* teksten[MAXAANTAL]; /* nieuwe_teksten is misschien 1 langer
```

```
                                /* (voor nullpointer)                    */
    char* tekst = lees();
    while(strcmp(tekst,"STOP")!=0 && aantal < MAXAANTAL-1){
        teksten[aantal] = tekst;
        aantal++;
        tekst = lees();
    }
    if(aantal == MAXAANTAL-1 && strcmp(tekst,"STOP")!=0){
        teksten[aantal] = tekst;
        aantal++;
    }

    char** nieuwe_teksten = malloc((aantal+1)*sizeof(char*));
    int i=0;
    for(i=0; i<aantal; i++){
        nieuwe_teksten[i] = teksten[i];
    }
    nieuwe_teksten[aantal] = NULL;

    return nieuwe_teksten;
}

void geef_vrij(char** teksten){
    while(*teksten != NULL){
        printf("x");
        free(*teksten);
        teksten++;
    }
}

int main(){

    char ** teksten = lees_meerdere();
    printf("Ik las deze teksten in: "); /* overloop met hulpptr;
                                        /* anders kan free niet meer! */
    char ** w = teksten;
    while(*w != NULL){
        printf(".%s.\n",*w);
        w++;
    }

    geef_vrij(teksten);
    free(teksten);

    return 0;
}
```

Oefening 33

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

#define AANTAL_WOORDEN             10
#define GEMIDDELDE_LENGTE_WOORDEN   7
#define LENGTE_ARRAY_T     AANTAL_WOORDEN * (1+GEMIDDELDE_LENGTE_WOORDEN)


/* versie met indexering van pt; niet gevraagd */
/*
void lees(char ** pt){
```

```
    int i;
    for(i=0; i<AANTAL_WOORDEN; i++){
        scanf("%s",pt[i]);
        pt[i+1] = pt[i]+strlen(pt[i])+1;
    }
    pt[AANTAL_WOORDEN]=0;
}*/

/* versie met schuivende pt; zoals gevraagd */
void lees(char ** pt){
    int i;
    for(i=0; i<AANTAL_WOORDEN; i++){
        scanf("%s",*pt);
        *(pt+1) = *pt+strlen(*pt)+1;
        pt++;
    }
    *pt=0;
}

void schrijf(const char * const * pt){
    while(*pt!=0){
        printf("\n%s",*pt);
        pt++;
    }
}

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);

    schrijf(pt);

    return 0;
}

/*************/
/*   EXTRA   */
/*************/

/*  Merk op: als je met scanf wil aangeven hoeveel letters er maximaal
 *           ingelezen mogen/kunnen/zullen worden, dan moet dat aantal
 *           een hardgecodeerd getal zijn. (Zelfs het gebruik van een
 *           constante of define is niet mogelijk.)
 *           Bij het gebruik van fgets(..,aantal,..) hoeft dat niet;
 *           hier is 'aantal' een variabele. Daarom komt scanf niet
 *           in aanmerking voor gebruik in onderstaande oefening.
 */

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

#define LENGTE_ARRAY_T     10


/* versie met indexering van pt */
```

```
/* Opgelet! Bij het gebruik van fgets zal de 'new line' (als gebruiker entertoets induwt)
   ook in de c-string opgeslagen worden; dat verwijderen we eruit in regel (***) */

void lees(char ** pt, int aantal_plaatsen){
    int i = 0;
    while(aantal_plaatsen>1){
        printf("Geef een woord in van maximaal %d karakter(s):  ",(aantal_plaatsen-1));
        fgets(pt[i],aantal_plaatsen,stdin);
        if(pt[i][strlen(pt[i])-1] == '\n') pt[i][strlen(pt[i])-1] = 0; /* (***) */
        int lengte_woord = strlen(pt[i]);
        pt[i+1] = pt[i] + lengte_woord+1;
        aantal_plaatsen -= lengte_woord+1;
        i++;
    }
    pt[i]=0;
}

void schrijf(char const * const * pt){
    while(*pt!=0){
        printf("\n%s",*pt);
        pt++;
    }
}

int main(){
    char* pt[LENGTE_ARRAY_T/2+1];  /* zodat je ook nog een nullpointer
                                      kan wegsteken op het einde van de
                                      pointertabel */
    char t[LENGTE_ARRAY_T];

    pt[0] = t;
    int lengte = LENGTE_ARRAY_T;

    lees(pt,lengte);

    schrijf(pt);

    return 0;
}
```

Oefening 34

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

/* dan moet je niet overal 'struct' voor het type 'Deeltal' schrijven */
typedef struct{
    int waarde;
    int aantal_delers;
    int* delers;
}Deeltal;

void schrijf_ints(const int * x, int aantal){
    int i;
    for(i=0; i<aantal-1; i++){
        printf("%i-",x[i]);
    }
    printf("%i",x[aantal-1]);
}
```

```
void schrijf_deeltal(const Deeltal * x){
    printf("%d ",x->waarde);
    schrijf_ints(x->delers,x->aantal_delers);
    printf("\n");
}


int main(){
    Deeltal g;
    g.waarde = 6;
    g.aantal_delers=3;
    g.delers = (int*) malloc(g.aantal_delers*sizeof(int));
    g.delers[0] = 1;
    g.delers[1] = 2;
    g.delers[2] = 3;

    schrijf_deeltal(&g);

    free(g.delers);

    return 0;
}
```

Oefening 35

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

typedef struct{
    int waarde;
    int aantal_delers;
    int * delers;
}Deeltal;

int aantal_delers_van(int x){
    int aantal = 1;
    int d;
    for(d = 2; d <= x/2; d++){
        if(x%d == 0){
            aantal++;
        }
    }
    return aantal;
}

int * delers_van(int x, int aantal_delers){
    int * delers = (int*) malloc(aantal_delers*sizeof(int));
    int index = 0;
    int d;
    for(d = 1; d <= x/2; d++){
        if(x%d == 0){
            delers[index++] = d;
        }
    }
    return delers;
}

/* g mag geen zwevende pointer zijn! */
void vul_aan(Deeltal * g){
```

```
    g->aantal_delers = aantal_delers_van(g->waarde);
    g->delers = delers_van(g->waarde,g->aantal_delers);
}

/* g mag geen zwevende pointer zijn! */
void lees_deeltal(Deeltal * g){
    scanf("%i",&(g->waarde));
    if(g->waarde < 0){
        g->waarde *= -1;
    }
    vul_aan(g);
}

void lees_deeltallen(Deeltal* t, int aantal){
    int i;
    for(i=0; i<aantal; i++){
        lees_deeltal(&t[i]);
    }
}

void schrijf_ints(const int * x, int aantal){
    int i;
    for(i=0; i<aantal-1; i++){
        printf("%i-",x[i]);
    }
    printf("%i",x[aantal-1]);
}
void schrijf_deeltal(const Deeltal * x){
    printf("%d   ",x->waarde);
    schrijf_ints(x->delers,x->aantal_delers);
    printf("\n");
}
void schrijf_deeltallen(const Deeltal * ptr, int aantal){
    int i;
    for(i=0; i<aantal; i++){
        schrijf_deeltal(&ptr[i]);
    }
}

const Deeltal * zoek(int waarde, const Deeltal * ptr, int aantal){
    int i=0;
    while(i<aantal && ptr[i].waarde != waarde){
        i++;
    }
    if(i==aantal)
        return 0;
    else
        return &ptr[i];
}

void free_delers(Deeltal * g){
    /* belangrijk om te controleren of je goed bezig bent: */
    printf("\nik geef (delers van) dit deeltal vrij: %d",g->waarde);
    free(g->delers);
}
void free_deeltallen(Deeltal * g, int aantal){
    int i;
    for(i=0; i<aantal; i++){
        free_delers(&g[i]);
    }
}
void free_deeltallen_volledig(Deeltal ** g, int aantal){
```

```
    int i;
    for(i=0; i<aantal; i++){
        free_delers(&(*g)[i]);
    }
    free(*g);
}

main(){

    printf("Geef een eerste deeltal in.\n");
    Deeltal x;
    lees_deeltal(&x);
    schrijf_deeltal(&x);

    int aantal;
    printf("Nu komt het vervolg; hoeveel deeltallen wil je nog ingeven?");
    scanf("%d",&aantal);
    printf("Geef nu die deeltallen in.\n");

    Deeltal * deeltallen = (Deeltal*) malloc(aantal * sizeof(Deeltal));
    const Deeltal * gezocht;

    lees_deeltallen(deeltallen,aantal);
    schrijf_deeltallen(deeltallen,aantal);

    printf("\nIk zoek 20: ");
    gezocht = zoek(20,deeltallen,aantal);
    if(gezocht!=0) schrijf_deeltal(gezocht);
    else printf("\n20 niet gevonden... ");


    free_delers(&x);

    free_deeltallen(deeltallen,aantal);
    free(deeltallen);    /* moet je nog apart doen! Tenzij je voor de alternatieve
                            procedure free_deeltallen_volledig(...) gaat   */

    /*
    free_deeltallen_volledig(&deeltallen,aantal);
    */

    /* merk op: 'gezocht' geef je uiteraard niet vrij,
                want dit wijst naar geheugen dat reeds vrijgegeven werd.
                (en free op een nullpointer oproepen crasht) */

    return 0;
}
```

REEKS 5

Gelinkte lijsten en bit fiddling

Oefening 36

```
/*
 De pointer l is uiteraard niet gewijzigd; enkel k.
 De pointer l is dus nog steeds de nullpointer.
*/
```

Oefening 37

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

typedef struct knoop knoop;

struct knoop{
    int d;
    knoop * opv;
};

/* deel 1 */
void voeg_vooraan_toe(int x,knoop ** m){
    knoop * even_bijhouden = *m;
    *m = (knoop*) malloc(sizeof(knoop));
    (*m)->d = x;
    (*m)->opv = even_bijhouden;
}


/* deel 2 - niet-recursieve schrijfprocedure */
void print_lijst(const knoop * l){
    while( l != 0){
        printf("%d ",l->d);
        l = l->opv;
    }
    printf("X\n");
}

/* deel 3  - recursieve schrijfprocedure */
void print_lijst_recursief(const knoop * l){
    if( l!=0 ){
        printf("%d ",l->d);
        print_lijst_recursief(l->opv);
    }
    else{
        printf("X\n");  /* 'X' om einde van de lijst aan te geven,
                           zodat ook een lege lijst zichtbaar is */
    }
}

/* geeft geheugen weer vrij */
void vernietig_lijst(knoop **l) {
   knoop *h;
   while (*l != 0) {
       h = *l;
```

```
      *l = h->opv;
      printf("ik geef knoop %d vrij\n",h->d);  /* interessant voor educatieve
          doeleinden!!! */
      free(h);
   }
}


int main(){
    knoop * l = 0;

    voeg_vooraan_toe(7,&l);  /* geef &l mee; aan een kopie van de pointer l
                              * wijzigingen aanbrengen helpt immers niet
                              * (zie vorige oefening!!)                    */
    voeg_vooraan_toe(3,&l);
    print_lijst(l);          /* geef l mee; er moet toch niets veranderen
                              * aan dit adres                              */
    print_lijst_recursief(l);
    vernietig_lijst(&l);
    return 0;
}
```

Oefening 38

```
/* werk voort op vorige oefening */
#include <stdlib.h>
#include <time.h>

typedef struct knoop knoop;

struct knoop{
    int d;
    knoop * opv;
};

void voeg_vooraan_toe(int x,knoop ** m){
    knoop * even_bijhouden = *m;
    *m = (knoop*) malloc(sizeof(knoop));
    (*m)->d = x;
    (*m)->opv = even_bijhouden;
}

void print_lijst(const knoop * l){
    while( l != 0){
        printf("%d ",l->d);
        l = l->opv;
    }
    printf("X\n");
}

/* geeft geheugen weer vrij */
void vernietig_lijst(knoop **l) {
   knoop *h;
   while (*l != 0) {
       h = *l;
      *l = h->opv;
      printf("ik geef knoop %d vrij\n",h->d);  /* interessant voor educatieve
          doeleinden!!! */
      free(h);
   }
```

```
}

/* lijst met elementen in stijgende volgorde, dubbels mogelijk */
knoop* maak_gesorteerde_lijst_automatisch(int aantal, int bovengrens){
    knoop * l = 0;
    int getal = bovengrens;
    int i;
    for(i=0; i<aantal; i++){
        getal -= rand()%3;
        voeg_vooraan_toe(getal,&l);
    }
    return l;
}

/* Omdat je de eerste knoop nooit zal verwijderen,
   hoef je niet per se via knoop** door te geven      */
void verwijder_dubbels(knoop * l){
    knoop * h = l;
    while(h != 0){
        while (h->opv != 0 && h->opv->d == h->d){
            knoop * m = h->opv;
            h->opv = m->opv;
            free(m);
        }
        h = h->opv;
    }
}

/* HIERONDER EEN ALTERNATIEF, zie je wat er hier gebeurt?     */
void verwijder_dubbels_alternatieve_versie(knoop * l){
    /* h staat aan het begin van de gelijke waarden */
    /* m staat aan het einde van de gelijke waarden */
    knoop * h = l;
    knoop * m = l;
    while(m != 0){
        while(m != 0 && h -> d == m -> d){
            m = m -> opv;
        }
        if(h -> opv != m){
            knoop * magweg = h -> opv;
            knoop * einde_magweg = magweg;
            while(einde_magweg != 0 && einde_magweg->opv != m){
                einde_magweg = einde_magweg -> opv;
            }
            einde_magweg -> opv = 0;
            vernietig_lijst(&magweg);
        }
        h -> opv = m;
        h = m;
    }
}


int main(){
    srand(time(NULL));
    knoop * l = maak_gesorteerde_lijst_automatisch(10,100);
    print_lijst(l);

    printf("\nnu worden dubbels verwijderd: \n");
    verwijder_dubbels(l);
    printf("\nna verwijderen van dubbels: \n\n");
    print_lijst(l);
```

```
    /*
    printf("\nnu worden dubbels verwijderd: \n");
    verwijder_dubbels_alternatieve_versie(l);
    printf("\nna verwijderen van dubbels: \n\n");
    print_lijst(l);
    */
    vernietig_lijst(&l);

    return 0;
}
```

Oefening 39

```
/* werk voort op voorgaande oefening */

void keerom(knoop ** l){
    if( *l != 0 && (*l)->opv != 0){
        knoop * a;
        knoop * b;
        knoop * c;
        a = *l;
        b = a->opv;
        c = b->opv;
        while(c!=0){
            b -> opv = a;
            a = b;
            b = c;
            c = c->opv;
        }
        b -> opv = a;
        (*l) -> opv = 0;
        *l = b;
    }
}

int main(){
    srand(time(NULL));
    knoop * l = maak_gesorteerde_lijst_automatisch(10,100);
    print_lijst(l);

    printf("\nnu wordt lijst omgekeerd: \n");
    keerom(&l);
    printf("\nna omkeren: \n\n");
    print_lijst(l);

    vernietig_lijst(&l);

    return 0;
}
```

Oefening 40

```
/* werk voort op voorgaande oefening */
/* bouw nog veiligheid in: geen twee dezelfde lijsten mergen! */


/* deze versie vermijdt dubbele pointers in de implementatie,
   maar is veel langer dan de versie die daarna komt.
*/
```

```
knoop * merge(knoop ** a, knoop ** b){
    knoop * l;
    knoop * i = *a;
    knoop * j = *b;
    knoop * k;
    if(*a == 0){
        l = *b;
        *b = 0;
        return l;
    }
    else if(*b == 0){
        l = *a;
        *a = 0;
        return l;
    }

    if((*a)->d < (*b)->d){
        l = *a;
        i = (*a)->opv;
        k = l;
    }
    else{
        l = *b;
        j = (*b)->opv;
        k = l;
    }
    while(i != 0 && j != 0){
        if(i->d < j->d){
            k -> opv = i;
            i = i -> opv;
        }
        else{
            k -> opv = j;
            j = j -> opv;
        }
        k = k -> opv;
    }
    if(i != 0){
        k -> opv = i;
    }
    if(j != 0){
        k -> opv = j;
    }

    *a = 0;
    *b = 0;
    return l;

}

/* Deze versie is korter dankzij knoop**k = &l */
knoop * merge_bis(knoop ** a, knoop ** b){
    knoop * l;
    knoop ** k = &l;
    knoop * i = *a;
    knoop * j = *b;

    while(i != 0 && j != 0){
        if(i->d < j->d){
            *k = i;
            i = i->opv;
        }
```

```
        else{
            *k = j;
            j = j->opv;
        }
        k = &((*k)->opv);
    }
    if(i != 0){
        *k = i;
    }
    if(j != 0){
        *k = j;
    }
    *a = 0;
    *b = 0;

    return l;
}

int main(){
    srand(time(NULL));
    knoop * m = maak_gesorteerde_lijst_autom(3,1000);
    knoop * n = maak_gesorteerde_lijst_autom(3,1000);
    printf("\nhier zijn twee lijstjes: \nLIJST m:\n");
    print_lijst(m);
    printf("\nLIJST n:\n");
    print_lijst(n);
    printf("\nDeze worden gemerged. \n\n");
    /* knoop * mn = merge(&m,&n);   */
    knoop * mn = merge_bis(&m,&n);

    printf("\nNa het mergen: \nLIJST m:\n");
    print_lijst(m);
    printf("\nLIJST n:\n");
    print_lijst(n);
    printf("\nRESULTAAT: \n");
    print_lijst(mn);
    printf("\n\n\n");

    vernietig_lijst(&mn);

    return 0;
}
```

Oefening 41

```
#include <stdio.h>
typedef struct knoop knoop;

struct knoop{
    int d;
    knoop * opv;
};
/* voorlopig nog nodig om 'zoek' te kunnen testen; nadien weghalen zodat je
   niet per ongeluk met ongesorteerde lijst verder doet. */
void voeg_vooraan_toe(int x,knoop ** m){
    knoop * even_bijhouden = *m;
    *m = (knoop*) malloc(sizeof(knoop));
    (*m)->d = x;
    (*m)->opv = even_bijhouden;
}
```

```
void print_lijst(const knoop * l){
    if( l!=0 ){
        printf("%d ",l->d);
        print_lijst(l->opv);
    }
    printf("\n");
}
knoop ** zoek(int x, knoop ** kn){
    while( *kn != 0 && (*kn)->d < x){   /* hier zoeken in GEORDENDE lijst!! */
        kn = &((*kn)->opv);
    }
    return kn;
}

int main(){
    knoop * l = 0;

    voeg_vooraan_toe(7,&l);
    voeg_vooraan_toe(3,&l);
    print_lijst(l);

    knoop ** hier = zoek(7,&l);
    if(*hier!=0){
        ((*hier)->d)++;      /* we verhogen 7 met een eenheid (testcase)         */
    }
    print_lijst(l);
    hier = zoek(10,&l);
    if(*hier!=0){
        ((*hier)->d)++;   /* 10 werd niet gevonden, dus er zal niets gebeuren */
    }                     /* merk op: als je '->' schrijft, moet je er zeker  */
    print_lijst(l);           /* van zijn dat wat vOOr '->' staat, geen null-     */
                          /* pointer is.                                      */

    /* iet vergeten ?!!! */
    return 0;
}
```

Oefening 42

```
/* vanaf nu gebruiken we voeg_vooraan_toe niet meer,
 * omdat dit de orde van de lijst om zeep zou kunnen helpen
 */
#include <stdio.h>
typedef struct knoop knoop;

struct knoop{
    int d;
    knoop * opv;
};
void print_lijst(const knoop * l){
    if( l!=0 ){
        printf("%d ",l->d);
        print_lijst(l->opv);
    }
    printf("\n");
}
knoop ** zoek(int x, knoop ** kn){
    while( *kn != 0 && (*kn)->d < x){
        kn = &((*kn)->opv);
    }
```

```
    return kn;
}
void voeg_toe(int x, knoop ** m){
    knoop ** hier = zoek(x,m);
    knoop * naar_achter_te_schuiven;
    naar_achter_te_schuiven = *hier;
    *hier = (knoop*) malloc(sizeof(knoop));
    (*hier)->d = x;
    (*hier)->opv = naar_achter_te_schuiven;
}


int main(){
    int tab[10] = {5,4,6,9,2,8,3,1,0,7};
    knoop * l = 0;

    int i;
    for(i=0; i<10; i++){
        voeg_toe(tab[i],&l);
    }
    print_lijst(l);

    /* niets vergeten?? */
    return 0;
}
```

Oefening 43

```
void free_lijst(knoop ** m){
    if(*m != 0){
        free_lijst(&((*m)->opv));
        printf("ik geef knoop met inhoud %d vrij\n",(*m)->d);
        free(*m);
        *m = 0;
    }
}
int main(){
     // ...
     free_lijst(&l);
     return 0;
}
```

Oefening 44

```
// gaat voort op vorige oefeningen

void verwijder(int x, knoop ** m){
    knoop ** hier = zoek(x,m);
    if(*hier!=0 && (*hier)->d == x){
        knoop * te_verwijderen = *hier;
        *hier = (*hier)->opv;
        free(te_verwijderen);
    }
    else{
        printf("\n%i niet in lijst ",x); /* ENKEL IN TESTFASE LATEN STAAN
                                          * - of exceptie werpen
                                          * - of returntype van maken */
    }
}
```

```
int main(){
     int tab[10] = {5,4,6,9,2,8,3,1,0,7};
     struct Knoop * l = 0;

     int i;
     for(i=0; i<10; i++){
         voeg_toe(tab[i],&l);
     }
     print_lijst(l);

     /* verwijder 0 2 4 6 8 10 12 14 16 18 !! (in gewijzigde volgorde weliswaar) */
     for(i=0; i<10; i++){
         verwijder(2*tab[i],&l);
     }

     print_lijst(l);

     free_lijst(&l);
     return 0;
}
```

Oefening 45

```
/*
Oplossing wordt niet gepubliceerd.
*/
```

Oefening 46

```
#include <stdio.h>

typedef unsigned int uint;

const int LENGTE = sizeof(uint)*8;

int bit_i(uint x, int i){
    /* verschillende mogelijkheden:             */
    /* return ((x<<(LENGTE-1-i))>>(LENGTE-1)); */
    /* return (x & (1<<i)) >> i;                */
    return (x >> i) & 1;
}

void schrijf(uint x, int lengte){
    int i;
    int viervoud = lengte/4*4;
    for(i=lengte-1; i>=viervoud; i--){
        printf("%u",bit_i(x,i));
    }
    for(i=viervoud-1; i>=0; i-=4){
        printf(" ");
        int k;
        for(k=0; k<4; k++){
            printf("%u",bit_i(x,i-k));
        }
    }
    printf("    %u\n",x);
}
// OPMERKING
```

```
// Misschien was jouw code een stuk korter, en hieraan gelijk:
//
//        int i;
//        for(i=lengte-1; i>=0; i--){
//            printf("%u",bit_i(x,i));
//            if(i%4==0) printf(" ");   //(**)
//        }
//        printf("\n");
//
//     Hierboven wordt er binnen de lus een test uitgevoerd, om te besluiten of er een
    spatie moet geprint worden.
//     DAT IS ECHTER GEEN ZUINIGE PRAKTIJK!! Zo'n supersimpele test, waarvan je toch
    praktisch op voorhand
//     de uitkomst kent, nl.
    ...vals-vals-vals-WAAR-vals-vals-vals-WAAR-vals-vals-vals-WAAR... in een terugkerend
//     patroon, remt de uitvoering van je programma.
//     Dat is in deze cursus nog niet van cruciaal belang, maar gelieve geen vuile
    manieren te kweken die er dan in
//     de cursus Algoritmen weer uit moeten...
//
// Dus moet je nu op zijn minst al beseffen dat de gepresenteerde code wel meer werkt
    vraagt om te coderen
// (en te testen voor verschillende lengtes!), maar dat ze wel efficienter werkt.
// (En dat er op voorhand een korte lus staat om de 'leidende bits' af te printen, heeft
    ermee te maken
//  dat de laatste vier bits samen uitgeschreven moeten worden.)

uint eenbit(int i){
    return 1<<i;
}

int aantal_eenbits(uint getal){
    int teller=0;
    while (getal!=0) {
        teller+=(getal&1);
        getal>>=1;
    }
    return teller;
}

uint bit_i_aangezet(uint x, int i){
    return x | eenbit(i);
}

// niet expliciet gevraagd; wel nuttig
uint alle_bits_omgedraaid(uint x){
    return ~x;
}

uint bit_i_uitgezet(uint x, int i){
    return x & ~(eenbit(i));
}

uint bit_i_gewisseld(uint x, int i){
    return x ^ eenbit(i);
}

int zijn_gelijk(uint a, uint b){
    return a^b;
}

int is_even(uint g){
```

```
        return 1 & ~(g&1);
}

// De code van de procedure wissel_even_met_oneven_bits(uint x)
// wordt aan eigen inventiviteit overgelaten.
// Veel puzzelplezier!
// (... en uittesten)

/*
Antwoord op de voorlaatste vraag:

n & (n-1) == 0 betekent dat n en n-1 nergens beide een bit gelijk hebben aan 1. (**)

Voorbeeld: stel
n   = abcde1000
n-1 = abcde0111

de bits die meer naar links staan (abcde) kunnen niet 1 zijn, want dat zou in
tegenspraak zijn met de eerste vaststelling (**).

Dus moeten ze allemaal 0 zijn, zodat n van de vorm 000001000 is.
Dus n heeft slechts 1 bit die aanstaat. Met andere woorden:
de test n & (n-1) == 0 gaat na of n een macht is van 2.
*/


int product(int a,int b){
        int res=0;
        while (a>0){
                if (a&0x1==1)
                        res+=b;
                b<<=1;
                a>>=1;
        }
        return res;
}

main(){
    printf("lengte van uint is %u\n",LENGTE);

    // ! we doorlopen de for-lus nu wel met een unsigned int !! (... en het heeft een
        goede reden)
    uint i;
    for(i=0; i<40; i++){
        schrijf(i,10);
    }
    printf("\n");
    for(i=0; i<40; i++){
        schrijf(eenbit(i),LENGTE);
    }
    printf("\n");

    uint getal = 951;
    for(i=0; i<10; i++){
        printf("getal:                  %u    ",getal); schrijf(getal,LENGTE);
        printf("getal met bit %u aan:   %u    ",i,bit_i_aangezet(getal,i));
            schrijf(bit_i_aangezet(getal,i),LENGTE);
        printf("\n");
    }

    printf("\n");
    for(i=0; i<10; i++){
```

```
        printf("bits omgedraaid van %u: ",i); schrijf(alle_bits_omgedraaid(i),10);
    }
    printf("\n");

    //uint getal = 951;
    for(i=0; i<10; i++){
        printf("getal:                  %u    ",getal); schrijf(getal,10);
        printf("getal met bit %u uit:   %u    ",i,bit_i_uitgezet(getal,i));
            schrijf(bit_i_uitgezet(getal,i),10);
        printf("\n");
    }

    for(i=0; i<10; i++){
        printf("getal:                          %u    ",getal); schrijf(getal,10);
        printf("getal met bit %u omgedraaid:   %u    ",i,bit_i_gewisseld(getal,i));
            schrijf(bit_i_gewisseld(getal,i),10);
        printf("\n");
    }


    for(i=0; i<10; i++){
        if(is_even(i)==0) printf("\n%u is oneven ",i);
        else printf("\n%u is   even",i);
    }

    // test van zijn_gelijk nog niet toegevoegd; vul dat zelf aan

    printf("\nGeef twee getallen:\n");
    int a,b;
    scanf("%i %i",&a,&b);
    if(product(a,b) == a*b) printf("\nproduct is juist berekend");
    else printf("\product is niet juist berekend");

    return 0;

}
```