Dit document is grotendeels afleid van "Coronado Enterprises C TUTOR (ver 2.00) Nov 21, 1987"

Een Inleiding tot C.

Wagemakers Staf

De programmeertaal C is oorspronkelijk ontwikkeld door Dennis Ritchie van Bell Laboratries, en was ontworpen om te werken onder UNIX op een PDP-11. Alhoewel C ontwikkeld was om onder UNIX te werken wordt C tegenwoordig als de standaard programmeertaal gezien op verschillende platformen. C is geen goede programmeertaal voor beginners, hiervoor zijn andere programmeertalen geschreven zoals bijvoorbeeld BASIC (**B**eginners **A**llpurpose **S**ymbolic **I**nstruction **C**ode). Omdat er op verschillende komputersystemen een C compiler voorhanden is, zijn toepassingen die in deze programmeertaal geschreven zijn eenvoudiger over te zetten naar een ander platform.

Contents

Een inleiding tot C.

1 "Je eerste C programma."

Een nieuwe programmeertaal leren is niet zo eenvoudig, men moet nieuwe instrukties en begrippen gebruiken, oude gewoontes afleren, ... De beste programmeertaal is immers deze welke men kent en gewoon is om te gebruiken!

1.1 Alle begin is moeilijk!

Bij C moet men met de volgende zaken rekening houden.

1. C maakt een onderscheid tussen grootte en kleine letters, de variabele "INDEX" is dus niet hetzelfde als "InDeX" of "index".

2. C houdt enkel rekening met de eerste 32 karakters.

De manier om een programmeertaal te leren is te kijken naar een simpel programma dat in deze taal geschreven is. Onderstaand voorbeeld is zo'n eenvoudig programma (het doet namelijk niets).

```
main()
{
}
```

Het woord "main" is erg belangrijk in een C programma, het moet namelijk eenmaal voorkomen of beter het mag maar eenmaal voorkomen in een C programma. De ronde haakjes geven aan dat het om een procedures gaat, tussen deze haakjes kunnen we eventueel ook variabelen plaatsen (hierover later meer). Met het woordje "main" geven we aan waar een programma begint. De {} geven het begin en het einde van de procedure aan.

1.2 Een programma dat iets doet!

Onderstaand programma is een meer intersant voorbeeld. Dit programmaatje zet "Dit is een test!" op het scherm. Met de instruktie printf kunnen we dus data op het scherm zetten. De data die aan een instruktie of een procedure doorgegeven wordt staat in C altijd tussen ronde haakjes, met de aanhalingstekens geven we aan dat het om tekst gaat welke rechtstreeks naar het scherm gestuurd moet worden. Met ";" geven we het einde van een instruktie of funktie aan, dit moet in C altijd gebeuren!

```
main()
{
 printf ("Dit is een test!");
}
```

Het volgende programma zet wat meer tekst op het scherm, "\n" komt overeen met een return of m.a.w. ga naar de volgende lijn.

```
main()
{
  printf("Dit is de eerste lijn.\n");
```

```
  printf("En dit is de");
  printf(" volgende lijn.\n\n");
  printf("Dit is de laatste lijn.\n");
}
```

1.3 En cijfers?

Het volgende voorbeeld zet de waarde van een variabele op het scherm.

```
main()
{
int getal;
    getal = 13;printf(" De waarde van getal is %d\n",getal);
    getal = 27;printf(" De waarde van getal is %d\n",getal);
    getal = 10;printf(" De waarde van getal is %d\n",getal);
}
```

Met "int getal" definiren we een integer met de naam "getal". In C moeten alle variabelen eerste gedefinieerd worden. Met "%d" bepalen hoe de variabele "getal" op het scherm gezet wordt, hier decimaal.

1.4 Hoe plaatsen we kommentaar in een C programma?

Kommentaar wordt vaak gebruikt in een programma het dient om de werking van een programma te verduidelijken, voor jezelf n iemand anders. Met "/*" geven we het begin van de kommentaar aan, met "*/" het einde.

```
/* Met deze tekst houdt de C compiler gn rekening */
main() /* Hiermee ook niet!  */
{
  printf("Wat is kommentaar?!"); /* Kommentaar mag doorlopen
                                    op de volgende regel(s) */
}
```

1.5 Programmeer opdrachten!

1. Schrijf een programma dat je naam, adres en telefoonnummer op het scherm zet. Voor de postkode en nummer gebruiken we variabelen.

2 Lussen en sprongen.

In dit hoofdstuk bekijken we een belangrijk onderdeel namelijk "programma kontrole". Hiermee kunnen we bepaalde zaken een aantal keer uitvoeren, de werking van ons programma benvloeden enz.

2.1 De "while" lus

Onderstaand programma demonstreert de werking van de while instruktie. Het programma begint met het definiren van een integer met de naam "x", deze variabele wordt eerste op 0 gezet. Integenstelling tot andere programmeertalen (zoals bv. BASIC) heeft een nieuwe variabele in C een willekeurige waarde. De "while" lus wordt uitgevoerd totdat er niet meer aan de voorwaarde wordt voldaan, de instrukties die tot de "while" lus behoren worden gegroepeerd door de akkolades.

```
main()
{
int x;
    x = 0;
    while(x<6) {printf("x = %d\n",x);x=x+1;}
}
```

2.2 De "do ... while" lus

Een variant van de "while" lus is de "do ... while" lus, het volgende voorbeeld illustreert deze. Dit programma is bijna identiek aan het vorige, alleen wordt de lus gestart met de "do" instruktie, de voorwaarde staat hier achteraan de lus namelijk de "while" instruktie. Dit heeft als gevolg dat de "do ... while" lus altijd nmaal doorlopen wordt! Indien we dus x=0 vervangen door x=7 zal het programma de waarde van x eenmaal afprinten, ook al is er niet aan de voorwaarde voldaan!

```
main()
{
int x;
    x=0;
    do  {printf("De waarde van x is %d\n",x); x=x+1;} while (x<6);
}
```

2.3 De "for" lus

De "for" lus is eigelijk niets anders dan een verkorte notatie van de "while" lus, volgend voorbeeld illustreert de "for" lus. Net zoals bij de "while" lus geven we een variabele een bepaalde waarde, stellen we een voorwaarde en verhogen (of verlagen) deze variabele, alleen is de notatie vl korter.

```
main()
{
int x;
    for (x=0;x<6;x=x+1) printf ("x = %d\n",x);
}
```

2.4 De "if" en "else" instrukties

Met de "if" instruktie kunnen we een bepaalde voorwaarde stellen, hierdoor is een mogelijk de werking van een programma te benvloeden door bv. een variabele. Met de "else" instruktie kunnen we bepaalde delen van het programma laten uitvoeren die niet aan de "if" voorwaarde voldeden.

```
main()
{
int x;
    for (x=0;x<10;x=x+1) {
        if (x==5) printf("x is nu 5\n");
        if (x<5)  printf("x = %d, dit is kleiner dan 5\n",x);
        else
            printf("x = %d, dit is groter dan 5\n",x);
    }
}
```

Let op de dubbele gelijk - aan tekens bij de "if" instruktie, C maakt een onderscheidt tussen een vergelijking (aangeduid met "=="), en een overdracht (aangeduid met "="). Achter de "if" of "else" verwacht de C - kompiler slecht n instruktie, voor meerdere kommando's moeten we gebruik maken van de akkolades ({ <instr. 1> ; <instr. 2> ; ... ;}).

2.5 break en continue

Met break en continue kunnen we een procedure of een instruktie onderbreken en voorzetten.

```
main()
{
int x;
    for(x=5;x<15;x=x+1) {
        if(x==8) break;
        printf("De break - lus, x = %d\n",x);
        }
    for(x=5;x<15;x=x+1) {
        if(x==8) continue;
        printf("De continue - lus, x = %d\n",x);
        }
}
```

In de eerste "for" lus, de "break" lus dus, zal de lus door if(x==8) break; onderbroken worden indien x gelijk is aan 8. Tijdens de tweede "for" lus, de "continue" lus dus, zal indien x=8, het programma verder gaan aan het einde "for" - lus. Of m.a.w. de waarde 8 zal niet afgeprint worden, maar de "for" lus loopt normaal door!

2.6 De "switch" instruktie.

Met de "switch" instruktie kunnen net zoals bij de "if - else" instrukties de werking van ons programma benvloedden door een variabele. Dankzij de "switch" instruktie kunnen we echter lange en onoverzichtelijke "if - else" opeenvolgingen vermijden!

```
main()
{
int x;
    for (x=3;x<13;x=x+1) {
        switch (x) {
                case 3   :  printf("x is nu 3\n");break;
                case 4   :  printf("x is nu 4\n");break;
                case 5   :
                case 6   :
                case 7   :
                case 8   :  printf("x is nu tussen 5 & 8\n");break;
                case 11  :  printf("x is nu 11\n");break;
            default  :  printf("x voldoet niet aan voorwaarde\n");break;
            }    /* einde switch */
     }           /* einde for    */
}
```

2.7 De "goto" instruktie.

De "goto" instruktie is een omstreden instruktie, er zijn namelijk mensen die er boeken over geschreven hebben. Hierin (probeerde) ze duidelijk te maken dat de "goto" instruktie niet thuis hoorde in een "gestruktureerde" programmeertaal. De "goto" instruktie is echter een volwaardige instruktie in C en vele andere programmeertalen, bovendien zijn er situaties te bedenken waarin de "goto" instruktie de eenvoudigste (en dus de beste) oplossing is. Het zou dus stom zijn om dan gn "goto" te gebruiken!

```
main()
{
int hond, kip, kat;
    goto echte_start;
    ergens:  printf("Dit is nog een lijn met tekst\n");
    goto gedaan;
/* In dit gedeelte staat de enige zinvolle "goto" */
/*-----------------------------------------------*/
    echte_start:
    for(hond=1;hond<6;hond=hond+1)   {
       for(kat=1;kat<6;kat=kat+1)    {
          for(kip=1;kip<4;kip=kip+1) {
             printf("Hond = %d  Kat = %d  Kip = %d\n",hond,kat,kip);
             if ((hond + kat + kip) > 8) goto voldoende;
            };
         };
      };
     voldoende:  printf("Dit zijn voldoende dieren ( te veel eigenlijk ).\n");
/* -----------------------------------------------*/
   printf("Dit is een eerste lijn\n"); goto lab1;
   lab2:
   printf("De derde ...  \n"); goto ergens;
   lab1:
   printf("En dit ...  de tweede\n");goto lab2;
   gedaan:
   printf("dit is echt de laatste!\n");
}
```

2.8 Programmeer opdrachten!

1. Schrijf een programma dat je naam 10 keer op het scherm zet. Maak dit programma 3 maal, elke keer met een andere lus metode.

2. Schrijf een programma dat van 1 tot 10 telt, print elk getal op het scherm. Bij 3 n 7 toon je een ekstra berichtje op het scherm.

3 Datatypes, bewerkingen & vergelijkingen

I n de twee vorige hoofdstukken hebben al een datatype leren kennen nl. de integer, in dit hoofdstuk gaan we meer datatypes bekijken. Ook gaan we dieper in op wiskundige bewerkingen en vergelijkingen, C heeft hiervoor ook verkorte notaties dit maak de bronkode vaak moeilijk leesbaar voor mensen die C niet zo goed kennen.

3.1 Eenvoudige bewerkingen

In het onderstaand voorbeeld zie we enkele eenvoudige bewerkingen nl.: +(optelling), -(aftrekking), *(vermenigvuldiging), /(deling), %(rest).

```
main()
{
int a,b,c;
    a=14;
    b=3;  printf("a=13;b=3");
    c=a+b;printf("c=a+b=%d\n",c); /* optelling         */
    c=a-b;printf("c=a-b=%d\n",c); /* aftrekking       */
    c=a*b;printf("c=a*b=%d\n",c); /* vermenigvuldiging */
    c=a/b;printf("c=a/b=%d\n",c); /* deling           */
    c=a%b;printf("c=a%b=%d\n",c); /* rest berekening  */
    c=12*a+b/2-a*b*2/(a*c+b*2);printf("c=12*a+b/2-a*b*2/(a*c+b*2) =%d\n",c);
    a=a+1;printf("a=a+1=%d\n",a);
    b=b*5;printf("b=b*5=%d\n",b);
    a=b=c=20;printf("a=b=c=20");
    a=b=c=12*13/4;printf("a=b=c=%d\n");
}
```

3.2 Andere datatypes

```
main()
{
int   a,b,c;          /* -32768  ...  32767   zonder decimaal punt  */
char  x,y,z;          /* -128    ...  127     zonder decimaal punt  */
float d,e,f;          /* 3.4E-38 ...  3.4E+38 met decimaal punt     */
      a=b=c=27;
                x=y=z='A';
                d=e=f=3.6792;
                a=y;                    /* a is nu 65 (karakter 'A') */
                x=b;                    /* x is nu -27               */
                d=b;                    /* d is nu -27.00            */
                a=e;                    /* a is nu 3                 */
}
```

In het vorige voorbeeld leren we een aantal nieuwe datatypes kennen (int kenden we al).

int	:	Integer, wordt door de komputer voorgesteld als 2 bytes en kan bijgevolg een getalwaarde aannemen van -32768 tot +32767.
char	:	Dit type is eigenlijk bedoeld voor een karakter voor te stellen (meestal in ASCII) en wordt in het geheugen van de komputer bewaard als een byte. Hierdoor kan een "char" een getalwaarde aannemen van -127 tot +128.
floar	:	Is bedoeld voor het weergeven van getallen na de komma, bij de meeste C kompilers tot 7 cijfers na de komma.

Laten toch nog even dieper ingaan op de werking van het vorige programma. Zoals hierboven besproken is een "char" in wezen een "int" die door 1 byte voorgesteld wordt, we kunnen dus zonder problemen een "char" omzetten in een "int". In de andere richting (van "int" naar char" dus) is er geen standaard, indien de integer buiten het bereik van de "char" valt zal deze een waarde krijgen tussen de -127 en de +128.

Zetten we een integer om in een "float" zal deze laatste de getalwaarde van de integer aannemen. In andere richting (van "float" naar "int"), zal de C - kompiler de cijfers na de komma laten vallen.

3.3 Uitgebreide datatypes

C heeft nog variaties op degene die we al besproken hebben. Zo heeft een "int" de volgende uitbreidingen:

long int	:	lange integer, heeft bij de meeste C kompilers een bereik van -2147483648 tot 2147483647.
short int	:	korte integer, is bij de meeste C kompiler hetzelfde als een "int" en heeft dus een bereik van -3
unsigned int	:	integer zonder teken, van 0 tot 65535.

Enkel de "long", "short" en "unsigned" moet men eigelijk in een programma schrijven, de "int" zal door ervaren programmeurs weggelaten worden.

Het "float" datatype heeft ook een uitbreiding nl. de "double" en heeft een bereik van 1.7E-308 tot 1.7E+308, het aantal cijfers na de komma verschilt soms van C kompiler tot C kompiler. Er zijn nog andere samenstellingen mogelijk zoals de "long unsigned int", "unsigned char", enz.

3.4 Konversie karakters

We hebben al een konversie karakter van "printf" bekeken nl. "%d", met "printf" kunnen echter nog andere datatypes weergeven.

```
main()
{
int a;                          /* normale integer      */
long b;                         /* lange integer        */
short c;                        /* korte integer        */
unsigned d;                     /* integer zonder teken */
char e;                         /* karakter             */
float f;                        /* float                */
double g;                       /* dubbele precisie float*/
   a=1023;b=2222;c=123;d=1234;e='X';f=3.14159;g=3.1415926535898;
   printf("a=%d\n",a);          /* decimale   uitvoer   */
   printf("a=%o\n",a);          /* oktale     uitvoer   */
   printf("a=%x\n",a);          /* hex.        uitvoer  */
   printf("b=%ld\n",b);         /* lange dec.  uitvoer  */
   printf("c=%d\n",c);          /* korte decimale       */
   printf("d=%u\n",d);          /* zonder teken         */
   printf("e=%c\n",e);          /* karakter             */
   printf("f=%f\n",f);          /* float                */
   printf("g=%f\n\n",f);        /* double float         */
   printf("a=%d\n",a);          /* normale int.  uitvoer */
   printf("a=%7d\n",a);         /* rechts uitgelijnd    */
   printf("a=%-7d\n",a);        /* links uitgelijnd     */
   c=5;d=8;
   printf("a=%*d\n",c,a);
```

```
   printf("a=%*d\n\n",d,a);
   printf("f=%f\n",f);          /* normale float uitvoer       */
   printf("f=%12f\n",f);
   printf("f=%12.3f\n",f);      /* drie cijfers na de komma    */
   printf("f=%12.5f\n",f);      /* vijf cijfers na de komma    */
}
```

Andere konversie karakters zijn:

"%d"	:	decimale uitvoer, deze kenden we al.
"%o"	:	oktale uitvoer.
"%x"	:	hexadecimaal.
"%ld"	:	lange decimale uitvoer.
"%u"	:	decimaal zonder teken.
"%c"	:	karakter.
"%f"	:	"float", wordt ook gebruikt voor een "double".

Ook is het mogelijk om met "printf" de uitvoer op een bepaalde manier te schikken, om bv. waarde in kolommen op het scherm te krijgen. In het voorgaande programma zien we ook hier een voorbeeld van. Met "%7d" bv. wordt alles rechts uitgelijnd met een veldbreedte van zeven, met "%-7d" links uitgelijnd met een veldbreedte van zeven.

Algemeen:

-	:	links uitlijnen.
(n)	:	veldbreedte van de cijfers voor de komma, hiervoor kunnen we ook een variabele gebruiken door gebruik te maken van "*".
.	:	scheidt (n) van (m)
(m)	:	aantal cijfers na de komma, ook hier kunnen we gebruik maken van een variabele.

3.5 Eenvoudige vergelijkingen

Het volgende programma illustreert enkele eenvoudige vergelijkingen, in de kommentaar wordt het resultaat vermeld. Misschien nieuw is de logische "not" en inverteert het resultaat van een vergelijking of een andere bewerking, zo is != niet gelijk aan.

```
main()
{
int x=11,y=11,z=11;
float r=12.987,s=12.987;
   if (x==y) z=-13;     /* Dit maakt z gelijk aan z-13  */
   if (x>z) z=-10;      /* Dit maakt z gelijk aan z-10  */
   if (!(x>z)) z=500;   /* Dit verandert niets          */
   if (b<=c) z=0;       /* Dit maakt z gelijk aan 0     */
   if (r!=s) z=20;      /* Dit maakt z gelijk aan 20    */
}
```

Groter dan of gelijk, kleiner dan, ... vergelijkingen bestaan natuurlijk ook in C, maar worden in dit voorbeeld niet aangehaald.

3.6 waar of nietwaar?????

Het is misschien niet slecht om eens te bekijken wat waar ("true") en nietwaar ("false") voor C is. C bekijkt nietwaar als gelijk aan nul, en waar als groter dan nul of m.a.w. positief, de meeste C kompilers nemen voor waar "1" aan maar het is niet aan te raden deze "1" te gebruiken in een wiskundige bewerking.

```
main()
{
int x=11,y=11,z=11;
  if (x=(r!=s)) z=111;      /* Dit maakt x positief, en z = 111 */
  if (x=y) z=222;           /* Dit maakt z = 222                */
  y=0; if (x=y) z=333;      /* Dit verandert niets              */
  y=3;x=1; if (x=y) z=444;  /* Dit maakt z = 333                */
  if (x) z=555;             /* Dit maakt z = 555                */
  x=0; if (x) z=666;        /* Dit verandert niets              */
}
```

In de eerste vergelijking is het resultaat waar, x wordt dus positief en z 111.

De tweede vergelijking is geen vergelijking, de getalwaarde van "y" wordt overgedragen naar "x" vermits deze positief is wordt dit door de C kompiler als waar bekeken. In de derde vergelijking wordt dit verduidelijkt vermits "x" nul wordt, wordt deze als nietwaar bekeken. De vierde vergelijking is eigelijk analoog de getalwaarde van "y" (3 dus) wordt overgedragen op "x" het resultaat is dus waar ongeacht de vorige waarde van "x".

De laatste vergelijk is vrij duidelijk vermits "x" gelijk is aan nul wordt deze als nietwaar bekeken.

3.7 "and" en "or" vergelijkingen.

Een "and" - vergelijking wordt in C voorgesteld als "&&", en een "or" - vergelijking als "||", het volgende voorbeeld illustreert "and" -en "or" vergelijkingen.

```
main()
{
int x,y,z;x=y=z=77;
   if ((x==y) && (x==77)) z=33;         /* Dit maakt z =33                */
   if ((x>y) || (z>12)) z=12;           /* Dit maakt z = 12               */
   if (x && y && z) z=11;               /* Dit maakt z = 11               */
   if ((x=1) && (y=2) && (z=3)) z=22;   /* Dit maakt x=1, y=2, z=3, z=22.0 */
   if ((x==2) && (y=3) && (z=4)) z=14;  /* Dit verandert niets            */
}
```

De eerste is vrij duidelijk de twee vergelijkingen zijn waar, het resultaat is ook waar. De tweede (een "or" vergelijking) is ook waar vermits z>12.Nummer vier is een goed voorbeeld van een "and" vergelijking vermits zowel "x", "y" en "z" groter zijn dan nul (waar dus) is de vergelijking ook waar. De vijfde is analoog, vermits "x" niet gelijk is aan 2, is deze nietwaar.

3.8 Pas op!!!

De volgende vergelijkingen zijn veel gemaakte fouten.

```
if (x > x); z=27;      /* z verandert altijd ongeacht de vgl.  */
if (x!=x) z=27.345;    /* z zal nooit veranderen               */
if (x=0)  z=27.345;    /* x wordt 0, z verandert nooit!        */
if (x==1) & (y==4);    /* Dit is geen and vergelijking, maar bewerking */
if (x==1) | (y==4);    /* Dit is geen or vergelijking, maar bewerking */
```

3.9 Verkorte notaties.

Verkorte notaties zijn zeer populair in C, toch zijn veel mensen er geen voorstanders en gebruiken deze dan ook niet. Het programma wordt op deze manier hl wat minder duidelijk voor andere (zeker voor beginnende programmeurs), toch is het belangrijk deze te begrijpen om de bronkode van iemand anders te kunnen lezen. Bovendien bespaard het veel wat schrijfwerk.

3.9.1 Verkorte bewerkingen.

```
main()
{
int x=0,y=2,z;
    printf("x=%d, y=%d,z=%d\n",x,y,z);  /* Dit verhoogt x met 1 */
    x=x+1;  printf("x=x+1=%d\n",x);     /* Dit ook!  */
    x++;    printf("x++, x=%d\n",x);    /* Dit ook!  */
    z=y++;  printf("z=y++, z=%d, y=%d\n",z,y); /* z=2, y=3 */
    z=++y;  printf("z=++y, z=%d, y=%d\n",z,y); /* z=4, y=4 */
    y=y-1;  printf("y=y-1=%d\n",y);     /* Dit vermindert y met 1 */
    y--;    printf("y--, y=%d\n",y);    /* Dit ook!  */
    --y;    printf("--y, y=%d\n",y);    /* Dit ook!  */
    z=y--;  printf("z=y--, z=%d, y=%d\n",z,y); /* z=1, y=0  */
    z=--y;  printf("z=--y, z=%d, y=%d\n",z,y); /* z=-1 y=-1 */
}
```

Vorig programma demonstreert enkele verkorte notaties van bewerkingen, met x++ en ++x doe we hetzelfde als x=x=+1. Toch bestaat er een verschil tussen deze twee, dit wordt in de twee volgende bewerkingen duidelijk. Met z=y++, wordt eerst de getal waarde van "y" overgedragen aan "z" daarna wordt "y" met 1 vermeerderd. Met z=++y, wordt eerst "y" met 1 vermeerderd deze nieuwe getalwaarde van "y" wordt dan overgedragen aan "z". De volgende voorbeelden zijn analoog aan de vorige, met –y en y– wordt "y" met 1 verminderd.

```
main()
{
float a=0.0,b=3.14159;
  printf("a=%.5f, b=%.5f\n",a,b);
  a=a+12; printf("a=a+12=%.5f\n",a);    /* Dit telt 12 bij a */
  a+=12;  printf("a+=12=%.5f\n",a);     /* Dit ook!  */
  a*=4.3; printf("a*=4.3=%.5f\n",a);    /* Dit vermenigvuldigt a met 4.3 */
  a-=b;   printf("a-=b=%.5f\n",a);      /* Dit vermindert a met b */
  a/=10.0;printf("a/=10.0=%.5f\n",a);          /* Dit deelt a door 10 */
}
```

In het bovenstaand programma hebben nog enkele voorbeelden van verkorte bewerkingen, hierbij wordt een variabele met zichzelf bewerkt door een vast getal of een andere variabele.

3.9.2 Verkorte vergelijkingen.

Ook voor vergelijkingen bestaan er verkorte notaties het volgende programma is hier een voorbeeld van.

```
main()
{
int a=5,b=4,c=10;printf("a=%d, b=%d, c=%d\n",a,b,c);
   if (b >= 3) a = 2;
       else a=10;
   printf("a=%d\n",a);
   a=5;b=4;c=10;              /* Dit doet hetzelfde als */
   a = ((b >= 3) ?  2 :  10);   /* de vorige vgl.          */
   printf("a=%d\n",a);
   c = (a>b?a:b); printf("c=b=%d\n",c);
   c = (a>b?b:a); printf("c=a=%d\n",c);
}
```

De eerste vergelijking is de "normale" uitdrukking, de tweede is de verkorte schrijfwijze van de eerste. Indien de voorwaarde (b>=3) waar is wordt a 2, indien de voorwaarde nietwaar is wordt a gelijk 10. De twee volgende vergelijkingen zijn analoog aan de vorige.

3.10 Samenvatting

3.10.1 Datatypes

type	bit breedte	getalwaarde
char	8	-128 ... +127
unsigned char	8	0 ... 256
signed char (=char)	8	-128 ... +127
int	16	-32768 ... +32767
short int of short (=int)	16	-32768 ... +32767
unsigned int of unsigned	16	0 ... 65536
float	32	-3.4E+38 ... +3.4E+38
double	64	-1.7E+308 ... +1.7E+308

3.10.2 Vergelijkingen

<	kleiner dan	!	not
>	groter dan	!=	niet gelijk aan
<=	kleiner dan of gelijk aan	&&	and
>=	groter dan of gelijk aan	||	or
==	gelijk aan		

3.10.3 Bitmanipulaties

In dit hoofdstuk hebben geen bit manipulaties besproken in de onderstaande tabel staan de meeste gebruikte bewerkingen op bit niveau vermeld.

&	and
|	or
i << n	SHL, i = i * 2^n
i >> n	SHR, i = i / 2^n
~	inv

3.11 Programmeer opdrachten!

1.Schrijf een programma dat van 1 tot 12 telt en dat telkens ook het kwadraat uitrekent.

```
1       1
2       4
3       9       enz.
```

2.Schrijf een programma dat van 1 tot 12 telt en dat telkens ook 1/x uitrekent tot 5 cijfers na de komma.

```
1       1.00000
2       0.50000
3       0.33333
4       0.25000 enz.
```

3. Schrijf een programma dat van 1 tot 100 telt, maar enkel de waardes tussen 32 en 39 op het scherm print.

4 funkties

Tot nu hebben we enkel funkties gebruikt welke al in C zelf voorhanden waren zoals printf(), het is ook mogelijk om zelf funkties aan te maken. Deze maken om veel gebruikte handelingen te groeperen, bovendien wordt het programma er beter leesbaar door.

4.1 Lokale & globale variabelen

Het volgende programma voorbeeld van programma met funkties.

```
int som;                /* Dit is een globale variabele */
main()
{
int index;
    hoofding();         /* Dit roept de funktie hoofding aan!  */
    for (index=1;index<=7;index++)
         kwadraat(index);  /* Dit roept de funktie kwadraat aan!  */
    einde();            /* Dit roept de funktie einde aan!     */
    }
hoofding()              /* Dit is de funktie hoofding */
{
    som=0;
    printf("Dit is het begin van het prg.\n\n");
}
kwadraat(getal)         /* Dit is de funktie kwadraat */
int getal;
{
int a;
    a=getal*getal;
    som+=a;
    printf("Het kwadraat van %d is %d\n",getal,a);
}
einde()         /* Dit is de funktie einde */
{
    printf("De som van de kwadraten is %d\n",som);
}
```

Het programma begin met het definiren van een integer met de naam som, vermits deze niet in een funktie gedefinieerd is spreken we van een globale variabele. Een globale variabele kan in alle funkties gebruikt worden en is voor al deze funkties ook hetzelfde. Variabelen welke in een funktie gedefinieerd zijn lokale variabelen, en kunnen enkel in de funktie gebruikt worden waarin ze gedefinieerd zijn. De funktie "kwadraat" is een voorbeeld hoe we een integer aan een funkties kunnen doorgeven, door kwadraat(index)wordt de getalwaarde van "index" overdragen aan "getal". Vermits "getal" in "kwadraat" gedefinieerd is kunnen we deze getalwaarde in de funktie kwadraat gebruiken. De werking van dit programma is vrij eenvoudig, in de "for" lus doorloopt index de getalwaarden van 1 tot 7, deze worden overdragen aan de funktie kwadraat welke (hoe kan het anders) het kwadraat uitrekent en afprint. In kwadraat wordt ook de som van al de kwadraten berekend, vermits som een globale variabele is, kan deze in heel het programma gebruikt worden. Met de funktie einde() printen wij de som van al de kwadraten af.

4.2 Waarden doorgeven zonder globale variabelen te gebruiken.

In het voorgaande voorbeeld maakten we gebruik van de globale variabele "som" om een getalwaarde aan het hoofdprogramma terug door te geven. Dit kan ook anders het volgende programma is hier een voorbeeld van.

```
main()
{
int x,y;

    for (x=0;x<=7;++x) {
      y = kwadraat(x);
      printf("Het kwadraat van %d is %d\n",x,y);
      }
    printf("\n");
    for (x=0;x<=7;++x) {printf("Het kwaadraat van %d is %d\n",x,kwadraat(x));}
}
kwadraat(a)
int a;
{
    a*=a;
    return(a);
}
```

Door de in de funktie "kwadraat", de instruktie return(a) toe te voegen kan de waarde van "a" in het hoofdprogramma gebruikt worden. In de eerste "for" lus wordt deze waarde overgedragen aan "y", in de tweede wordt ze direkt gebruikt bij het afprinten.

Nota: *In tegenstelling tot andere programmeertalen zoals bv. Pascal maakt C geen onderscheidt tussen "funkties" en "procedures". In Pascal is een funktie een stukje programma wat een getalwaarde uitrekent. Voor C zijn funkties en procedures hetzelfde, beide termen worden dan ook door elkaar gebruikt!*

4.3 funkties, welke geen integer teruggeven.

C gaat er altijd vanuit dat een funktie een "int" terug geeft, indien we andere datatypes gebruiken dienen we dit aan de C - kompiler "te vertellen".

```
float z;       /* Dit is een globale variabele */
main()
{
int index;
float x,y,sqr(),glsqr();
    for (index=0;index<=7;index++) {
       x = index;y = sqr(x);
       printf("Het kwadraat van %d is %10.4f\n",index,y);
    }
    for (index=0;index<=7;index++) {
       z=index; y = glsqr();
       printf("Het kwadraat van %d is %10.4f\n",index,y);
    }
}
float sqr(inval)
```

```
float inval;
{
float square;
    square=inval*inval;
    return(square);
}
float glsqr()
{
    return(z*z);
}
```

Het vorige programma is een vb. van funktie welke een "float" terug geeft. Het begint met het definiren van een globale variabele "z", welke we later zullen gebruiken. Daarna wordt in het hoofdprogramma "main" een integer gedefinieerd, gevolgd door 2 "floats". Hierna komen 2 vreemde "floats", "sqr()" en "glsqr()" lijken op funkties, wat ze ook zijn. Dit is de manier waarop we C duidelijk maken dat in het de procedure "main" een funktie een variabele terug geeft welke geen integer is, in dit voorbeeld een float.

In het midden van de listing staat de funktie "sqr(inval)", welke voorafgegaan wordt door het woordje "float", hiermee maken we duidelijk dat deze een variabele teruggeeft welke geen integer is. Dit lijkt misschien een beetje vreemd. Waarom moeten twee keer aangeven dat de funktie een "float" gaat teruggeven???? In C kunnen we echter gebruik maken van externe funkties, welke niet in n keer gekompileerd worden, in zo'n geval kan C niet weten welke datatype een externe funktie aan het hoofdprogramma gaat weergeven!

4.4 Variabelen en funkties

```
#include "stdio.h"      /* Prototypes voor invoer / uitvoer */
void sub1(void);        /* Prototype  voor sub1             */
void sub2(void);        /* Prototype  voor sub2             */
void sub3(void);        /* Prototype  voor sub3             */

int teller;             /* Dit is globale var., in heel het prg.  bruikbaar!  */

main()
{
register int index;     /* Enkel bruikbaar in "main" */
   teller=37;
   sub1();
   sub2();
   sub3();
   for (index=8;index>0;index--) {
       int a;          /* Enkel bruikbaar in de "for"-lus */
       for (a=0;a<=6;a++) printf("%d ",a);
       printf(" index is nu %d\n",index);
   }
}
int teller2;            /* Deze is vanaf nu bruikbaar */
void sub1(void)
{
int index;              /* Enkel beschikbaar in sub1 */
    index=23;
    printf("De sub1 waarde is %d\n",index);
    printf("teller = %d\n",teller);
}
void sub2(void)
```

```
{
int teller;             /* Deze var.  is enkel bruikbaar in sub2
                           en overschrijft de globale teller     */
    teller=53;
    printf("De sub2 waarde is %d\n",teller);
    teller2=77;
}
void sub3(void)
{
    printf("De sub3 waarde is %d\n",teller2);
}
```

Het vorige programma is een voorbeeld van variabele gebruik in funkties, tot nu hebben variabelen altijd in het begin van het programma of een funkties gedefinieerd. Dit is echter geen verplichting!

Als we het voorbeeld programma bekijken zien eerst 4 "rare" regels, negeer deze voorlopig hierop zullen we later terug komen.

Hierna beginnen we met het definiren van een globale variabele "teller", vermits deze buiten een procedure definieert is zal deze in al de volgende procedures bruikbaar zijn. Dit is ook zo voor "teller2" welke in het midden van het programma gedefinieerd is. De variabele "teller2" is echter niet bruikbaar in ons hoofdprogramma "main", omdat ze na deze procedure is aangemaakt.

4.5 "automatische" variabelen

Kijken we terug naar het hoofdprogramma dan zien we dat er een integer wordt gedefinieerd "index", voor "int index" staat ook nog het woordje "register" hierop zullen later terug komen. Deze variabele ("index") is enkel in het hoofdprogramma "main" bruikbaar vermits deze in de funktie gedefinieerd is. Dit noemt men ook soms een "automatische" (automatic voor de engelstaligen) variabelen, dit betekend dat de variabele enkel bruikbaar is in de funktie waarin ze is aangemaakt. En ander voorbeeld van een automatische variabele in de integer "a" in de "for" lus, vermits ze in de akkolades is aangemaakt is ze enkel bruikbaar in deze lus. Dit verklaart ook de term automatische variabelen, ze worden automatisch aangemaakt en weer vrijgegeven.

4.6 "void" funkties

Kijken we de subprocedures "sub1", "sub2" en "sub3" dan zien we dat deze worden voorafgegaan door het woordje "void", dit zelfde woordje staat ook tussen de ronde haakjes. Door "void" tussen de ronde haakjes te plaatsen maken we C duidelijk dat we aan deze funktie geen variabele willen doorgeven. Met het woordje "void" voor de funktie maken we C duidelijk dat deze geen waarde kan teruggeven.

4.7 Dezelfde variabele teruggebruiken

In de funktie "sub1" wordt dezelfde variabele naam "index" gebruikt als in het hoofdprogramma, vermits in beide funkties "index" een lokale variabele is er probleem. Beide variabele hebben wel dezelfde naam maar hebben een aparte plaatst in het geheugen van de komputer.

Hetzelfde gebeurt in de funktie "sub2" hier wordt dezelfde variabele naam gebruikt als de globale variabele "teller". Door deze naam opnieuw te gebruiken als een lokale in de funktie "sub2" is globale variabele "teller" niet beschikbaar in "sub2".

4.8 Wat is een "register" variabele?

Zoals de meeste onder jullie weten bevat een microprocessor registers, welke veel sneller zijn het geheugen. Door het woord "register" voor de variabele definitie te plaatsen geven we C aan dat we een register van de P willen gebruiken i.p.v. het geheugen. Het aantal registers is wel beperkt.

4.9 Wat is een "prototype"?

In de regels 2 tot 4 maken we een prototypes aan voor de procedures "sub1", "sub2" en "sub3". Hiermee geven we aan hoe deze funkties er gaan uitzien, tijdens het kompileren zal C kijken of we tegen deze voorwaarden geen fouten maken indien dit wel het geval is zal C dit melden met een foutmelding. Of m.a.w. hiermee beveiligen we onszelf tegen programmeer fouten!

4.10 "static" variabelen

Automatische variabele zijn enkel gekend in de funktie waarin ze aangemaakt zijn, ze worden aangemaakt in het geheugen bij het de deklaratie en weer vrijgegeven bij het beindigen van de funktie. Hierdoor gaat echter ook de waarde verloren!! Bij "static" variabelen, wordt een vaste geheugenplaats gereserveerd, de variabele in enkel bruikbaar binnen de funktie maar wordt niet uit het geheugen gewist, of m.a.w. de waarde blijft behouden. Het volgende voorbeeld zal dit ver duidelijken.

```
main()
{
static int i;
    for (i=1;i<=5;i++) printf("%3d %3d\n",i,f(i));
}
int f(i) int i;
{
int s=100; return(s+=i);
}
```

Met static:	1	101	Zonder static:	1	101
	2	103		2	102
	3	106		3	103
	...	...		...	
	5	115		5	105

4.11 Wat is "rekursie"??

Een rekursieve funktie is een funktie dit zichzelf aanroept het volgende programma is hier een voorbeeld van.

```
#include <stdio.h>
main()
{
int index =8;
    tel_af(index);
```

```
}
tel_af(teller)
int teller;
{
    char c;
    teller--;
    printf("teller = %d\n",teller);
    if (teller > 0) tel_af(teller);
    printf("Teller is nu 0\n");
}
```

4.12 Programmeer opdrachten!

1. Schrijf een programma dat je naam tien keer op scherm print, het printen gebeurt in een funktie.

5 Defines & Macros

Defines en macros zijn eigenlijk een hulp voor de programmeur, door van deze technieken gebruik te maken worden programma's duidelijker en overzichtelijker.

5.1 Wat is een "define"?

Met een define kunnen bv. een waardegetal en naam geven, hierdoor kan een programma duidelijker worden. Bovendien hoeven we niet heel het programma te doorlopen indien er een bepaalde waarde verandert moet worden. Een eenvoudig voorbeeldje ...

```
#define START 0   /* Start van de for-lus */
#define EINDE 9   /* Einde van de for-lus */
#define MAX(A,B)  ((A)>(B)?(A):(B))   /* MAX macro */
#define MIN(A,B)  ((B)>(A)?(A):(B))   /* MIN macro */
main()
{
int index,min,max;
int teller = 5;
    for (index=START;index<=EINDE;index++) {
       max=MAX(index,teller);
       min=MIN(index,teller);
       printf("Max.  = %d  ; Min = %d\n",max,min);
    }
}
```

Door de instruktie #define START 0 komt het woordje "START" overeen met de getalwaarde 0, met #define EINDE 9 komt "EINDE" overeen met 9. Deze twee worden later gebruikt om het begin en het einde in de "for"-lus aan te geven.

5.2 Wat is een "macro"?

Een macro is analoog met een gewone "define", alleen bestaant een macro uit n of meedere instrukties. In het voorgaande voorbeeldje staan 2 macro's nl. "MAX" en "MIN", welke gebruikt worden om het respektievelijk het maksimum en het minimum te berekenen. Toch bestaat er een belangrijk verschil tussen een procedure (of een funktie) en een macro, bij een macro zal de C - kompiler telkens als hij dezelfde macro tegenkomt vervangen door de instrukties van deze macro. Bij een funktie wordt ernaar het begint van de funktie gesprongen, of m.a.w. een funktie staat slecht n keer in het geheugen, een macro het aantal keer dat ze wordt gebruikt! Het gebruik van lange veel gebruikte macro is dus zeker niet aan te raden (hierbij wordt er veel geheugen verspilt).

5.2.1 Kijk uit!

Alhoewel macro's het leven van de programmeur moet vereenvoudigen zijn ze ook soms de reden van programatiefouten, het volgende voorbeeld bevat enkele fouten.

```
#define WRONG(A) A*A*A                  /* foute macro */
#define CUBE(A) (A)*(A)*(A)             /* Goede macro */
#define SQUR(A) (A)*(A)
#define ADD_WRONG(A) (A)+(A)            /* foute optelling */
#define ADD_RIGHT(A) ((A)+(A))          /* goede optelling */
```

```
#define START 1
#define STOP  7
main()
{
int i,offset;
   offset = 5;
   for (i=START;i<=STOP;i++) {
      printf("Het kwadraat van %3d is %4d, de 3 macht is %6d\n",
              i+offset,SQUR(i+offset),CUBE(i+offset));
      printf("De foute 3 macht van %3d is %6d\n",i+offset,WRONG(i+offset));
   }
   printf("\n En nu ...  De optelling macro's\n");
   for (i=START;i<=STOP;i++) {
      printf("Foute optelling = %6d, de juiste = %6d\n",
              5*ADD_WRONG(i),5*ADD_RIGHT(i));
   }
}
```

De eerste macro ("WRONG") is een voorbeeld van een foute derde machtsverheffing, in sommige gevallen zal ze werken in andere niet. De tweede ("CUBE") toont hoe het wel moet. Vervangen we "A" door bv. de eerste bewerking welke de macro moet uitvoeren dan krijgen we (1+5)*(1+5)*(1+5) = 216 voor de juiste ("CUBE"), voor de foute macro ("WRONG") krijgen we 1+5*1+5*1+5 = 16 wat een foutief antwoord is. De volgende macro ("SQUR") berekend het kwadraat, en is analoog aan de vorige macro's.

De volgende macro ("ADD_WRONG") is een vb. van een foutieve optelling, dit kunnen we opnieuw het beste aantonen door getalwaarden in te vullen, 5 * (1) + (1) = 6, terwijl we eigelijk 10 als resultaat verwachten. De macro "ADD_RIGHT" zal wel de juiste uitkomst geven, 5*(1+1) = 10.

5.3 Wat doet "enum" instruktie?

```
main()
{
enum {nul,een,twee,drie,vier,vijf,zes} resultaat;
enum {zon,ma,di,wo,don,vrij,zat} dag;
   resultaat = nul;   printf("resultaat = %d\n",resultaat);
   resultaat = een;   printf("resultaat = %d\n",resultaat);
   resultaat = twee;  printf("resultaat = %d\n",resultaat);
   resultaat = drie;  printf("resultaat = %d\n",resultaat);
   resultaat = vier;  printf("resultaat = %d\n",resultaat);
   resultaat = vijf;  printf("resultaat = %d\n",resultaat);
   resultaat = zes;   printf("resultaat = %d\n",resultaat);
   for(dag=ma;dag < vrij;dag++) printf("Dit is %d dag\n");
}
```

Met de "enum" instruktie in derde regel van het vb. maken we een integer aan met de naam resultaat, de namen tussen de akkolades stellen voor deze integer een getalwaarde voor van 0 tot 6, hetzelfde gebeurt met de integer dag. De "enum" kan bv. handig zijn om te gebruiken in kombinatie met de "switch" instruktie om zo een programma overzichtelijker te maken.

5.4 Programmeeropdrachten!

1.Schrijf een programma dat van 7 tot -5 telt, gebruik de "#define" instruktie om het begin en het einde aan te geven.

6 Strings & Arrays

C maakt geen onderscheid tussen strings en arrays (in tegenstelling tot bv. BASIC), voor C is een string een verzameling van karakters. Wel zijn er bepaalde instrukties die het werken met strings eenvoudiger (zouden) maken.

6.1 Wat is array?

Een array kunnen we het best omschrijven als een verzameling van data, welke van hetzelfde type zijn bv. integers, "floats", ... Een array van het "char" type noemen we een string. Het werken met arrays kan soms vrij verwarrend zijn, vooral bij meergedimensioneerde.

6.2 Wat is een string?

Zoals eerder aangehaald is een string een verzameling van karakters, meestal gebruikt om tekst weer te geven bv. het beeldscherm. In C wordt (zoals in vele andere programmeertalen) een string afgelosten door de ASCII waarde nul, een voorbeeldje ...

```
main()
{
char naam[5];
   naam[0] = 'D';
   naam[1] = 'A';
   naam[2] = 'V';
   naam[3] = 'E';
   naam[4] = 0;  /* Nul kar.  = einde string !!!!  */
   printf("De naam is %s\n",naam);
   printf("Een letter van de naam is %c\n",naam[2]);
   printf("Een deel van de naam is %s\n",&naam[1]);
}
```

Met char naam[5]; maken we een array aan van karakters welke 5 groot is met de naam "naam". Of m.a.w. "naam" is een string welke 5 karakters kan bevatten. Met de volgende programmaregels kopiren we de woord "DAVE" in de string. Met het "printf" - kontrole karakter %s kunnen we een string afprinten, ook is het mogelijk om n karakter van de string af te printen zoals in het voorbeeld gellustreerd. De laatste "printf" toont een deel van de string, hiervoor wordt gebruikt gemaakt van het "&" teken, hiermee bedoelen we een adres. Dit is eigenlijk een inleiding tot het volgende hoofdstuk en zullen we ook dan behandelen, nog even geduld ...

6.3 Enkele "string" instrukties

In het vorig voorbeeld hebben we gezien hoe we data in een string kunnen plaatsen, dit was echter vrij omslachtig. C heeft echter instrukties die dit eenvoudiger maken, een voorbeeldje ...

```
main()
{
char naam1[12],naam2[12],alfa[25];
char titel[20];
     strcpy(naam1,"Rosalinda");
     strcpy(naam2,"Zeke");
     strcpy(titel,"Dit is de titel");
```

```
    printf("     %s\n\n",titel);
    printf("Naam 1 is %s\n",naam1);
    printf("Naam 2 is %s\n",naam2);
    if (strcmp(naam1,naam2)>0)    /* geeft 1 als naam1 > naam2 */
        strcpy(alfa,naam1);
    else
        strcpy(alfa,naam2);
    printf("alfa=%s\n",alfa);
    strcpy(alfa,naam1);
    strcat(alfa,"    ");
    strcat(alfa,naam2);
    printf("Beide zijn :  %s\n",alfa);
}
```

De volgende "string" instruktie worden in het vb. gebruikt :

strcpy	:	Hiermee kopiren we data in een string, inkl. de nul welke de string afsluit.
strcmp	:	Hiermee kunnen we "strings" met elkaar vergelijken, voor de vergelijking wordt een ASCII tabel gebruikt. Dit heeft als gevolg dat kleine - en hoofdletters invloed hebben op de vergelijking. C heeft funkties om een string naar hoofdletter of kleine letters te konverteren, hierover later meer ...
strcat	:	Deze instruktie kunnen we gebruiken om "strings" samen te voegen, de tweede "string" tussen de haakjes wordt bij de eerst geplakt.

6.4 Een array van integers

Het volgende programma is een voorbeeld van een array met integers, eerst wordt er een array aangemaakt met de naam "a" daarna een integer "x". In de eerste "for" lus wordt in de array data geladen, in de tweede wordt de array "a" achteruit apgeprint.

```
main()
{
int a[12];
int x;
   for(x=0;x<12;x++) a[x] = 2*(x+4);
   for(x=11;x>=0;x--) printf("a[%d]=%d\n",x,a[x]);
}
```

6.5 Een array van floats

Het volgende programma illustreert de array's met "floats".

```
char naam1[]="Eerste programma titel";
main()
{
int index;
int brol[12];
float waarde[12];
static char naam2[]="Tweede programma titel";
    for (index=0;index<12;index++) {
       brol[index]=index+10;
       waarde[index]=12.0*(index+7);
```

```
    }
    printf("%s\n",naam1);
    printf("%s\n",naam2);
    for (index=0;index<12;index++)
       printf("%5d %5d %10.3f\n",index,brol[index],waarde[index]);
}
```

De eerste regel van het programma demonstreert hoe op een eenvoudigere manier data in string kunnen plaatsen. Let op de legen haakjes bij de aanmaak van de string, de kompiler zal zelf voldoende plaats voorzien in het geheugen en de data in de string kopiren inklusief het afsluitkarakter nul. In het programma wordt nog een string op deze wijze aangemaakt, let op de het woordje "static", dit vermijdt dat ze een automatische zou zijn. Voor de rest is een eigelijk niets nieuws in het voorbeeld.

6.6 Data terugkrijgen uit een funktie

Array's gedragen zich in een funktie als de andere datatypes die we al kenden van de vorige hoofdstukken in een funktie. In het volgende voorbeeld wordt er een array van integers door gegeven aan de funktie "doe_maar". Zoals bij andere datatypes vertellen we de funktie door een variabele tussen de haakjes te plaatsen (hier "lijst"), hierna definiren we dit het datatype van deze variabele. Let hierbij op de lege haakjes bij de definitie van de array "lijst", opnieuw zal C zelf voldoende plaats voorzien voor de array.

```
main()
{
int index;
int matrix[20];
    for (index=0;index<20;index++) matrix[index]=index+1;
    for (index=0;index<5;index++)
       printf("Start     matrix[%d] = %d\n",index,matrix[index]);
    doe_maar(matrix);
    for (index=0;index<5;index++)
       printf("Einde     matrix[%d] = %d\n",index,matrix[index]);
}
doe_maar(lijst)
int lijst[];
{
int i;
   for (i=0;i<5;i++)  printf("Originele matrix[%d] = %d\n",i,lijst[i]);
   for (i=0;i<20;i++) lijst[i]+=10;
   for (i=0;i<5;i++)  printf("Bewerkte  matrix[%d] = %d\n",i,lijst[i]);
}
```

6.7 Multigedimensioneerde arrays

In het volgende voorbeeld worden meervoudige of multigedimensioneerde arrays gebruikt, dit wilt enkel zeggen dat de array uit meer dan n kolom bestaat. De arrays "m1" en "m2" zijn hier voorbeelden van, de rest van het programma is niet echt nieuw.

```
main()
{
int i,j;
int m1[8][8],m2[25][12];
```

```
    for (i=0;i<8;i++) for (j=0;j<8;j++) m1[i][j]=i*j;
    for (i=0;i<25;i++) for (j=0;j<12;j++) m2[i][j]=i+j;
    m1[2][6] = m2[24][10]*22;
    m1[2][2] = 5;
    m1[m1[2][2]][m1[2][2]]=177;   /* Dit is m1[5][5] = 177; */
    for (i=0;i<8;i++) {
       for (j=0;j<8;j++) printf("%5d ",m1[i][j]);
       printf("\n");
    }
}
```

6.8 Programmeer opdrachten!

1.Schrijf een programma met drie korte strings welke elk zes karakters groot zijn, kopieer hier met "strcpy" n, twee, drie in. Voeg de drie strings samen en print het resultaat tien keer op het scherm

2.Defineer 2 integer arrays, van 10 integers lang. Vul deze met willekeurige data met een for lus, tel beiden op en plaats het resultaat in een andere array. Print het resultaat van de drie arrays op scherm.

7 "pointers"

E en "pointer" is niets anders dan een adres, de meeste hogere programmeertalen schermen de programmeur bij het normale gebruik volledig af van deze adressen. C echter niet, dit komt in het begin misschien een beetje verwarrend over...

7.1 Een voorbeeldje ...

```
main()
{
int index,*pt1,*pt2;
   index = 39;
   pt1   = &index;
   pt2   = pt1;
   printf("De waarde is %d %d %d\n",index,*pt1,*pt2);
   *pt1  = 13;      /* Dit verandert index */
   printf("De waarde is %d %d %d\n",index,*pt1,*pt2);
}
```

In bovenstaande voorbeeld worden "pointers" gebruikt, voor het ogenblik negeren we de aanmaak van de 2 variabelen met een sterretje. In de volgende lijn gebeurt niets nieuws, de integer "index" wordt gelijkgesteld met 39. De volgende regel is echter iets raar, "pt1" wordt gelijkgesteld aan "index" voorafgegaan door "&", hiermee bedoelen de adreswaarde van "index". Of nog anders gezegd, "pt1" is een "pointer" naar de variabele "index". Hierna wordt "pt2" gelijkgesteld aan "pt1", hierdoor is "pt2" ook een pointer naar "index".

7.2 Twee belangrijke regels.

1.Een variabele voorafgegaan door een "&" is een adres naar deze variabele. Met pt1 = &index is de waarde van "pt1" gelijk aan het adres van "index".

2. Een "pointer" voorafgegaan door een sterretje komt overeen met de waarde van de variabele waarnaar de "pointer" wijst. Hierdoor zal het voorbeeld programma tijdens het afprinten 3 keer dezelfde waarde op het scherm zetten.

Geheugensteun :	1. Denk bij "&" aan een adres 2. Denk bij "*" aan een waarde

7.3 Er is maar n variabele!

Het is belangrijk in te zien dat er in het vorige voorbeeld maar n variabele is nl. "index", "pt1" en "pt2" zijn pointers naar deze variabele. Dit wordt aangetoond met de regel *pt1 = 13; hierdoor wordt de variabele "index" verandert.

7.4 Hoe maken we een pointer aan?

In de derde regel van het programma maken we een variabele "index" aan, gevolgd door 2 pointers "pt1" en "pt2" dit maken we C duidelijk door een sterretje. Een pointer wijst altijd naar een bepaald type van een variabele, hierdoor kunnen de pointers "pt1" en "pt2" niet naar een andere variabele wijzen dan een integer.

7.5 Een string is eigenlijk een pointer.

Aan de hand van het vorige voorbeeld hebben we heel wat teorie bekeken, belangrijke teorie, in C programma's worden pointers zeer vaak toegepast. Het is dus belangrijk om het werken met pointers goed te begrijpen. In het volgende voorbeeld bekijken hoe pointers toegepast kunnen worden op een string.

```
main()
{
char string[40],*daar,een,twee;
int *pt,lijst[100],index;
    strcpy(string,"Dit is een string ...");
    printf("De string is :%s\n",string);
    een=string[0];
    twee=*string;      /* een en twee zijn hetzelfde */
    printf("De eerste uitvoer is %c %c\n",een,twee);
    een=string[8];
    twee=*(string+8);  /* een en twee zijn hetzelfde */
    printf("De tweede uitvoer is %c %c\n",een,twee);
    daar = string+11;  /* string+10 is hetzelfde als string[10] */
    printf("De derde uitvoer is %c %c\n",string[11],*daar);
    for (index = 0;index < 100;index++) lijst[index]=index+100;
    pt = lijst+27;
    printf("De vierde uitvoer is %d %d\n",lijst[27],*pt);
}
```

In het vorige hoofdstuk hebben we gezien dat een string een array is van karakters, we kunnen een string (en dus ook een array) echter ook bekijken als een pointer naar het begin van een array. Dit kunnen we begrijpen aan de hand van het voorbeeld. Eerst maken we een string aan met de naam "string", gevolg door een pointer naar een karakter ("daar") en twee karakters ("een" en "twee").

Hierna plaatsen we met de instruktie "strcpy" data in de string, het karakter "een" wordt gelijkgesteld aan de eerste karakter van de string. Vermits een string per definitie ook een pointer is doen we hetzelfde met "twee", "twee" neemt ook de waarde aan van het eerste karakter in de string. Het resultaat is dat beide karakters "een" en "twee" gelijk zijn aan 'D'.

In het volgende deel van het programma gebeurt ongeveer hetzelfde, "een" wordt gelijkgesteld aan het negende karakter van de string (we beginnen vanaf 0 te tellen). Met "twee" gebeurt dit ook door gebruikt te maken van een string als een pointer. Het is fout om zaken te schrijven zoals "twee=*string[8];".

Vermits een "daar" een pointer is kunnen we "daar" gelijk stellen aan een adreswaarde van een element uit een string. Dit wordt gedemonstreerd met daar = string + 11. Laten we toch opmerken dat vermits "daar" aangemaakt is een pointer naar een karakter, het enkel naar een karakter kan wijzen. Daar een element uit een string een karakter is kan "daar" ook hiernaar wijzen.

7.6 Data aan een funktie geven als een pointer.

In het volgende voorbeeld geven data aan funktie door als een pointer.

```
main()
{
int pindas,appelen;
   pindas=100;
```

```
    appelen=101;
    printf("De startwaarden zijn %d %d\n",pindas,appelen);
    verander(pindas,&appelen);
    printf("De eindwaarde zijn %d %d\n",pindas,appelen);
}
verander(noten,fruit)
int noten,*fruit;
{
    printf("De waarde zijn %d %d\n",noten,*fruit);
    noten=135;
    *fruit=172;
    printf("De waarden zijn %d %d\n",noten,*fruit);
}
```

We beginnen met 2 variabelen te definiren "pindas" en "appelen", merk op dat beide normale variabelen zijn en geen pointers. We geven beiden een waarde en printen deze uit, roepen we de funktie "verander" aan. De variabele "pindas" wordt als een normale variabele aan de funktie door gegeven, de variabele "appelen" wordt als een adres doorgegeven. In de funktie "verander" geven dit aan door "noten" als een normale integer te definiren en "fruit" als een pointer naar een integer.

In de funktie "verander" printen we de twee waarden nogmaals uit, hierna veranderen we ze en printen de nieuwe waarden op het scherm. Dit zou vrij duidelijk moeten, vermits hier niets nieuws gebeurt.

Bij het verlaten van de funktie gebeurt er echter iets raars, bij het opnieuw afdrukken van de twee variabelen in het hoofdprogramma zien we dat "pindas" terug haar oude waarde heeft. Dit komt omdat C een kopie maakt van de waarde en deze aan de funktie doorgeeft, en dus de originele intakt laat. De variabele "appelen" is echter wel verandert door de funktie, C heeft hier een kopie gemaakt van de adres waarde en deze aan de funktie doorgegeven. Hierdoor hebben we in de funktie een pointer naar de integer "appelen" en wordt deze wel aangepast.

7.7 Programmeer opdrachten!

1. Maak een string aan kopieer hier data in met strcpy. Druk deze letter per letter af door gebruikt te maken van een "for" lus.

2. Pas nu het programma aan zodat de string achterstevoren op het scherm komt.

8 standaard invoer / uitvoer.

Dit komt misschien een beetje raar over, maar de C taal heeft geen invoer / uitvoer instrukties, deze moeten in principe door de gebruiker (de programmeur dus) zelf aangemaakt worden. Toch zijn in de loop der jaren enkele libraries ontwikkeld welk een "facto" standaard zijn geworden. Hierbij moeten we echter wel opletten bij het overzetten we van onze programma's naar een ander platform of C - kompiler vermits we hier niet met een officile te maken hebben ...

8.1 Include bestanden.

In n van de vorige hoofdstukken hebben we al gebruik gemaakt van een zgn. "header" bestand nl. "stdio.h", hierin zijn enkele standaard funkties in opgenomen om invoer/uitvoer van data in programma eenvoudiger te maken. Door gebruik te maken van deze "header" bestanden, worden deze in het programma opgenomen, of het programma wordt hierdoor groter. Dit is dan ook een reden waarom deze instrukties niet standaard bestaan in C. Er bestaan nog andere "header" bestanden raadpleeg de dokumentatie van uw C kompiler voor meer informatie. Ook is het mogelijk om zelf "header" bestanden aan te maken om zo funkties aan te maken, welke in meerdere verschillende programma's bruikbaar zijn.We kunnen op twee manieren "header" bestanden in ons programma opnemen :

"bestand.h"	:	hier gaat C het "header" bestand zoeken in de huidige direktorie, indien hij het bestand hierin niet vindt, zal hij verder zoeken in de "include" direktorie.
<bestand.h>	:	hier gaat C enkel zoeken in de include direktorie.

8.2 Een eerste voorbeeld.

```
#include <stdio.h>
main()
{
char c;
    printf("Type een kar., X = stop\n");
    do {
        c=getchar();putchar(c);
        } while (c != 'X');
}
```

In het bovenstaande voorbeeld zien we twee instrukties uit "stdio.h" nl. "getchar" en "putchar", zoals de namen dienen deze om een karakter te lezen ("getchar") en om het scherm te plaatsen ("putchar"). Indien we dit programma echter kompileren en uitvoeren zien echter iets raar. De funktie "getchar" wacht tot dat we de "return" toets indrukken het eerste getypte karakter wordt dan aan het programma doorgegeven. Via "putchar(c)" dit karakter dan nogmaals afgedrukt. Dit komt raar over omdat dit minder bruikbaar is in een programma. Gelukkig bestaan er andere instruktie om een karakter in te lezen.

8.3 De "scanf()" funktie.

Met de "scanf()" kunnen we allerei data van het toetsenbord lezen zoals integers, "floats", strings, ... Het volgende programma lees een integer in.

```
#include "stdio.h"
```

```
main()
{
int x;
    printf("Geef een getal van 0 tot 32767, type 100 om te stoppen.\n");
    do {
        scanf("%d",&x);
        printf("Het getal is %d\n",x);
    } while (x!=100);

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

Met scanf("%d",&x); lezen we een integer in, de data komt in x. Net zoals bij "printf" moeten we aangeven om welke type van data het gaat, we gebruiken hiervoor dezelfde konversie karakters. Let ook op het "&" voor x, hiermee geven we de adreswaarde door aan de funktie, "scanf()" verwacht altijd een pointer. Met de teorie uit het vorige hoofdstuk moet het duidelijk zijn waarom dit zo gebeurt. De rest van het programma zou vrij duidelijk moeten zijn, er telkens een integer ingelezen en afgedrukt totdat we het getal 100 ingeven.

8.4 Inlezen van een string.

Met "scanf()" kunnen we ook een string inlezen het konversie karakters is hetzelfde als bij "printf", het volgende programma is hier een voorbeeld van. Let ook op het ontbreken van "&" bij de "scanf" instruktie, vermits tekst een array is en per definitie een al pointer is (zie vorige hoofdstuk). Verder zou de werking van het programma vrij duidelijk moeten zijn. Het programma vraag telkens een string en druk deze daarna af, totdat van een string beginnen met 'X'.

Indien we dit programma kompileren en uitvoeren merken tot iets raars op, indien we in een string een spatie gebruiken wordt deze hierna weergegeven. Of beter gezegd, een spatie betekent voor "scanf()" einde string, ook dit komt vreemd over vermits dit minder bruikbaar is in programma. Er zijn echter ook libraries ontwikkeld welk invoer / uitvoer beter afhandelen dan "stdio.h" voorbeelden hiervan zijn "ncurses" uit de Unix wereld (ook overgebracht naar andere platformen), en "TurboVision" van Borland C++ (enkel dos).

```
#include "stdio.h"
main()
{
char tekst[25];
     printf("Geef een string in, tot 25 karakters.\n");
     printf("Een 'X' als eerste karakters zal het prg.  stoppen.\n");
     do {
         scanf("%s",tekst);
         printf("De string is -> %s\n",tekst);
     } while (tekst[0]!='X');
     printf("Einde ...\n");
}
```

Er bestaat echter nog een andere manier om een string in te lezen, via de "fgets()" funktie welke eigelijk bedoel is om een string uit een bestand te lezen, hierover meer in het volgende hoofdstuk. Het onderstaande programma is hier een voorbeeld van. Met fgets(string,10,stdin)laden we een string van maksimaal 10 karakters in "a", we kunnen wel meer karakters intypen maar enkel de eerste 10 zullen in "a" ingeladen worden. Met "stdin" geven we "fgets()" aan dat we de data inladen van het "standard input device", het toetsenbord, i.p.v. uit een bestand.

```
#include "stdio.h"
main()
{
char a[10];
fgets(a,10,stdin);
printf("U type :%s",string);
}
```

8.5 Geheugen invoer / uitvoer.

De volgende instrukties komen misschien raar over, en komen we ook niet in vele andere hoger programmeertalen tegen. In "stdio.h" bestaan er variaties van "printf()" en "scanf()" welke zaken in het geheugen kunnen schrijven of lezen, een voorbeeldje ...

```
main()
{
int nummers[5],resultaat[5],index;
char lijn[80];
    nummers[0]=74;
    nummers[1]=18;
    nummers[2]=33;
    nummers[3]=30;
    nummers[4]=97;
    sprintf(lijn,"%d %d      %d %d %d\n",nummers[0],nummers[1],
            nummers[2],nummers[3],nummers[4]);
    printf("%s",lijn);
    sscanf(lijn,"%d %d      %d %d %d\n",&resultaat[0],&resultaat[1],
            &resultaat[2],&resultaat[3],&resultaat[4]);
    for (index=0;index<5;index++)
         printf("Het eindresultaat is %d\n",resultaat[index]);
}
```

De eerste nieuwe instruktie die we tegenkomen is "sprintf()", deze is identiek aan "printf()" enkel wordt de data hier niet op het scherm gezet, maar in een string in het geheugen geprint. In het voorbeeld wordt er data in "lijn" gekopieerd, deze data o.a. bevat de getalwaarden van de array "nummers". Hierna de string "lijn" via een normale "printf()" op het scherm gezet.

De volgende nieuwe instruktie is "sscanf()" welke analoog is aan "scanf()" enkel werkt ook deze met string in het geheugen. In het voorbeeld wordt de data van array "nummers" op deze manier terug uit de string "lijn" gehaald en in de array "resultaat" geplaatst. Tenslotte drukken we via een "for" lus de array resultaat op het scherm.

Deze technieken kunnen handig zijn om bv. een string in te lezen en daarna in het geheugen de konversie te doen.

8.6 foutmeldingen...

Het kan in sommige toepassingen nodig zijn om zaken die op scherm komen naar een bestand te sturen. Dit kunnen we onder DOS of Unix eenvoudig doen door "prg(.exe) > test" aan de kommandolijn in te typen, hiermee sturen we data van "prg" naar het bestand test. Het kan echter ook nodig zijn dat de foutmeldingen op scherm blijven, en enkel de echte data naar het bestand worden gestuurd. Het volgende programma is hier een voorbeeld van. De instruktie "fprintf()" zal in het volgende hoofdstuk verder uitgelegd worden, zoals hier toegepast stuurt ze de data naar het "standard error device" of "stderr" het scherm dus. Runnen we het programma

dan al de data op het scherm zetten, voeren we het uit met "prg(.exe) > test" dan zal de data die via "printf()" verzonden wordt in het bestand "test" komen, de foutmeldingen komen nog steeds op het scherm.

```
#include "stdio.h"
main()
{
int index;
   for (index=0;index<5;index++) {
       printf("Deze lijn gaat naar het scherm.\n");
       fprintf(stderr,"Deze lijn gaat naar het stderr.\n");
    }
exit(4);
}
```

8.7 foutkodes ...

Foutkodes of "errorlevels" voor de engelstaligen, is een kode welke door een programma terug naar het besturingssysteem worden gezonden, deze zijn onder de meeste besturingssystemen bruikbaar. Dit gebeurt in vorige programma met exit(4), hiermee wordt het getal vier na het beindigen van het programma aan het besturingssysteem doorgegeven.

9 Lezen & schrijven van bestanden.

In "stdio.h" bestaan ook funkties om bestand aan te maken, te lezen en te schrijven. Eigenlijk is "stdio.h" hiervoor geschreven, dit verklaart ook waarom sommige funktie uit het vorige hoofdstuk niet echt deden werken zoals we zouden verwachten.

9.1 Openen van een bestand.

Het openen van een bestand kunnen we doen door gebruik te maken van de funktie "fopen()", welke er als volgt uit ziet:

fp = fopen("<Bestandsnaam>","operatie")

Met "fp" wordt een "filepointer" bedoeld, hiervoor is in "stdio.h" een definitie nl. "FILE", we kunnen een "filepointer" of bestandswijzer eenvoudig aanmaken door bv. "FILE *bestand;". Met <Bestandsnaam> bedoelen we de naam van het te openen bestand (dat had je wel door zeker ;-)), hiervoor kunnen we eventueel ook van een string gebruik maken. Met "operatie" geven we aan hoe het bestand gaan openen, de volgende mogelijkheden kunnen we hiervoor gebruiken:

"r"	:	read, lezen van een bestaand bestand.
"w"	:	write, aanmaken van een nieuw bestand bestaat de bestandsnaam al, dan wordt het oude bestand gewist.
"a"	:	append, bijvoegen van data aan een bestand.
"r+"	:	update, lezen en schrijven naar een bestand.
"w+"	:	schrijven + lezen.
"a+"	:	bijvoegen + lezen.

9.2 Sluiten van een bestand.

Elk bestand dat via de funktie "fopen()" geopend is, moet ook voor het programmaeinde gesloten. Enkel waarmee het bestand gesloten is zijn we zeker dat de veranderingen aan het bestand ook daadwerkelijk zijn aangebracht. De funktie van "stdio.h" hiervoor is fclose(fp), waar "fp" de bestandswijzer is naar het te sluiten bestand.

9.3 Schrijven naar een bestand.

Het volgende programma is een voorbeeld voor de aanmaak van een nieuw bestand. Met FILE *fp; maken we een "filepointer" of bestandswijzer aan met de naam "fp". Hierna maken een string "a" en een integer "x" aan. Met fp = fopen("10lijnen.txt","w"); maken we nieuw bestand aan met de naam "10lijnen.txt", bestond dit bestand al dan wordt het overschreven. Hier kopiren we data in de string "a". De tegenhanger van "printf()" om dat naar een bestand te schrijven is fprintf (fp,"<DATA>",<VAR1>,<VAR2>"). In de "for" lus wordt data in het bestand geplaatst via deze instruktie.

```
#include "stdio.h"
main()
{
FILE *fp;
char a[25];
int x;
```

```
    fp = fopen("10lijnen.txt","w"); /* een bestand open voor schrijven */
    strcpy(a,"Dit is een lijn.");

    for(x=1;x<=10;x++) fprintf(fp,"%s lijnnummer %d\n",a,x);
    fclose(fp); /* sluiten van het bestand */
}
```

9.4 Data bijvoegen aan een bestand.

Het volgende voorbeeld voegt data bij aan het einde van een bestand. Hiervoor gebruiken we "a" in de "fopen()" funktie, om data aan het bestand bij te voegen gebruiken we nu de funktie "putc()" uit "stdio.h". Deze is de tegenhanger van "putchar()", of beter het is eigenlijk dezelfde funktie, "putchar(<char>)" is hetzelfde als "putc(<char>,stdin)". Deze techniek kunnen we voor al de funkties voor bestanden gebruiken, willen lezen van het toetsenbord (de standaard invoer) gebruiken "stdin" als "filepointer". Willen naar het scherm schrijven gebruiken we "stdout" als "filepointer". Kijken we terug naar ons programma dan zien we dat er voor de rest niets bijzonder gebeurt, misschien er op wijzen hoe de string naar het bestand wordt geschreven. In de tweede "for" lus wordt a[x]; als voorwaarde gebruikt, een string wordt afgesloten door een "0" dus deze wordt nietwaar bij het einde van de string.

```
#include "stdio.h"
main()
{
FILE *fp;
char a[35];
int x,teller;

   strcpy(a,"Andere lijnen.");
   fp = fopen("10lijnen.txt","a"); /* open voor bijvoegen */

   for (teller=1;teller<=10;teller++) {
      for(x=0;a[x];x++) putc(a[x],fp);
      putc('\n',fp);
   }
   fclose(fp);
}
```

9.5 Lezen uit een bestand.

Het volgende programma lees uit een bestand hiervoor gebruiken "r" in de funktie "fopen()". In het programma wordt getest of te lezen bestand wel degelijk bestaat met if (fp ==NULL), "NULL" is voor ons aangemaakt in "stdio.h". Met de funktie "getc(fp)" halen we een karakter op uit het bestand. In de "do ... while" lus wordt er telkens een karakter opgehaald uit het bestand en op het scherm geplaatst totdat "c" gelijk is aan "EOF". Met "EOF" ("End Of File") wordt het einde van het bestand bedoelt. Misschien opmerken dat er in het programma n karakter te veel wordt afdrukt, het "EOF" karakter wordt immers mee afgedrukt.

```
#include "stdio.h"
main()
{
FILE *fp;
char c;
    fp=fopen("10lijnen.txt","r");
```

```
    if (fp==NULL) printf("Sorry, het bestand bestaat niet...\n");
    else {
        do {
            c=getc(fp);     /* haal een karakter uit het bestand */
            putchar(c);
            } while(c!=EOF);
    }
    fclose(fp);
}
```

9.6 Pas op!

Het gebruik van "EOF" geeft problemen met bv. "unsigned char" als karakter te gebruiken. Dit komt omdat "EOF" overeenkomt met "-1", en zal dus bij een "unsigned char" vertaalt worden als een 255. Of met andere worden er zal nooit een bestandseinde gevonden worden!

9.7 Lezen van een woord.

De tegenhanger van "scanf()" is "fscanf()", in het vorige hoofdstuk hebben we gezien dat "scanf()" stop met lezen bij een spatie. Dit is ook zo bij "fscanf()", dit kan zijn zin hebben bij het lezen uit een bestand waar de data gescheiden is door een spatie. Het volgende programma is bijna identiek aan het vorige enkel wordt de data nu per woord op gehaald en afgedrukt.

```
#include "stdio.h"
main()
{
FILE *fp;
char c;
char woord[100];
    fp=fopen("10lijnen.txt","r");
    if (fp==NULL) printf("Sorry, het bestand bestaat niet...\n");
    else {
        do {
            c=fscanf(fp,"%s",woord); /* haal een woord uit het bestand */
            printf("%s\n",woord);
            } while(c!=EOF);
    }
    fclose(fp);
}
```

Ook hier drukken we n woord te veel af vermits we eerst een woord ophalen, afdrukken en daarna pas testen op een "EOF". Dit kunnen eenvoudig oplossen door een "if" regel bij toevoegen, ook is het mogelijk door gebruikt te maken van een "while" lus. In het volgende programma is dit probleem opgelost met een "if".

```
#include "stdio.h"
main()
{
FILE *fp;
char c;
char woord[100];
    fp=fopen("10lijnen.txt","r");
    if (fp==NULL) printf("Sorry, het bestand bestaat niet...\n");
```

```
    else {
        do {
            c=fscanf(fp,"%s",woord); /* haal een woord uit het bestand */
            if (c!=EOF) printf("%s\n",woord);
            } while(c!=EOF);
    }
    fclose(fp);
}
```

9.8 De funktie "fgets(<string>,<aantal>,fp)"

De funktie "fgets()" hebben in het vorige hoofdstuk al gebruikt om een string inlezen van het toetsenbord. We kunnen ook gebruiken om dat uit een string te lezen, net zoals bij het lezen van het toetsen stop "fgets()" met lezen bij een "return". Indien deze funktie het einde van het bestand heeft bereikt geeft het de waarde "NULL" door. Voor de rest gebeurt er eigenlijk niets nieuws in het volgende programma.

```
#include "stdio.h"
main()
{
FILE *fp1;
char lijn[100];
char *c;
     fp1 = fopen("10lijnen.txt","r");
     do {
        c=fgets(lijn,100,fp1);
        if (c!=NULL) printf("%s",lijn);
     } while (c!=NULL);

     fclose(fp1);
}
```

9.9 Een string gebruiken als bestandsnaam.

Tot nu hebben in fopen altijd de bestandsnaam aan "fopen()" doorgeven tussen aanhalingstekens, we kunnen hiervoor ook een string gebruiken. Het volgende programma is identiek aan het vorige, enkel wordt er via "scanf()" een string lezen, welke we gebruiken als bestandsnaam in "fopen()".

```
#include "stdio.h"
main()
{
FILE *fp1;
char lijn[100],bestandsnaam[25];
char *c;
     printf("Geef een bestandsnaam ->");
     scanf("%s",bestandsnaam);
     fp1=fopen(bestandsnaam,"r");
     do {
        c = fgets(lijn,100,fp1);
        if (c!=NULL) printf("%s",lijn);
     } while (c!=NULL);
     fclose(fp1);
}
```

9.10 Aansturen van de printer.

Met de besproken funktie kunnen we ook de printer aansturen, vermits onder dos "PRN" als een bestand wordt bekeken (voor Unix kunnen we bv. "/dev/lp1" gebruiken). Het volgende programma is hier een voorbeeld van.

```
#include "stdio.h"
main()
{
FILE *fp1,*printer;
char c;
     fp1 = fopen("10lijnen.txt","r");
     printer=fopen("PRN","w");
     do {
        c=getc(fp1);
        if (c!=EOF) { putc(c,printer);putc(c,stdout); }
     } while (c!=EOF);
     fclose(fp1);
     fclose(printer);
}
```

9.11 Programmeer opdrachten!

1.Schrijf een programma dat twee bestandsnamen vraagt, n om een bestand te lezen n om te schrijven. Open beiden bestanden plus de printer, schrijf hierna een lus welke karakter per karakter het bestand lees. Stuur dit karakter naar de printer en naar het andere bestand.

2.Vraag een bestandsnaam. Lees het bestand lijnen per lijn, druk deze lijn samen met een lijnnummer op het scherm af.

10 "struct" & "union".

In dit hoofdstuk gaan we 2 nieuwe datatypes bekijken nl. "structure" en "union", alhoewel nieuw misschien foutief uitgedrukt is. Ze zijn bedoeld om data van verschillende types welke bij elkaar horen te groeperen.

10.1 De "struct" instruktie

```
main()
{
struct {
     char init;
     int leedtijd;
     int graad;
     } jongen,meisje;
     jongen.init='R';
     jongen.leedtijd=15;
     jongen.graad=75;
     meisje.leedtijd=jongen.leedtijd-1; /* zij is 1 j.  jonger */
     meisje.graad=82;
     meisje.init='H';
     printf("%c is %d jaar en heeft een graad van %d\n",
            meisje.init,meisje.leedtijd,meisje.graad);
     printf("%c is %d jaar en heeft een graad van %d\n",
            jongen.init,jongen.leedtijd,jongen.graad);
}
```

Het programma begint met een definiren van een "struktuur". De instruktie "struct" wordt gevolgd met enkele eenvoudige variabelen tussen de "{ }", welke onderdelen zijn van de struktuur. Na de "{ }" staan er 2 variabelen nl. "jongen" en "meisje", dit zijn de namen van de aangemaakte struktuur. Of m.a.w. "jongen" is een variabele met drie elementen "init", "leedtijd" en "graad" welke enkel data kan bevatten van het gedefinieerde type. Dit is ook zo voor "meisje" welk dus ook drie variabelen bevat, in het totaal hebben we dus 6 variabelen aangemaakt.

Verder zien we in het programma hoe we deze variabelen kunnen aanspreken, met "jongen.init" spreken we het variabele "init" van de struktuur "jongen" aan. Deze techniek maakt het programmeren dus heel wat eenvoudiger en het programma wordt er veel overzichtelijker door.

10.2 Een array van strukturen.

```
main()
{
struct {
   char init;
   int leedtijd;
   int graad;
   } kind[12];
int index;
    for (index=0;index<12;index++) {
       kind[index].init='A'+index;
       kind[index].leedtijd=16;
       kind[index].graad=84;
    }
    kind[3].leedtijd=kind[5].leedtijd=17;
    kind[2].graad=kind[6].graad=92;
```

```
    kind[4].graad=57;
    kind[10]=kind[4];
    for (index=0;index<12;index++)
       printf("%c is %d jaar en heeft een graad van %d\n",kind[index].init,
              kind[index].leedtijd,kind[index].graad);
}
```

In het vorige programma maken we gebruik van array waar de element van deze array bestaan uit een struktuur. We definiren zo'n array simpelweg door de "{ }" een array naam te plaatsen. Voor de rest gebeurt in het programma niet echt iets nieuws, misschien er toch even op wijzen dat het ook mogelijk is om twee strukturen aan elkaar gelijk te stellen. Dit wordt in het voorbeeld gellustreerd met de regel kind[10]=kind[4]; hiermee worden alle elementen van "kind[10]" gelijk aan de element van kind[4].

10.3 Het gebruik van "pointers" in strukturen.

Het volgende programma is bijna identiek aan het vorige, enkel worden er bij sommige operaties gebruikt gemaakt van pointers.

```
main()
{
struct {
     char init;
     int jaar;
     int graad;
     } kind[12],*point,extra;
int index;
   for (index=0;index<12;index++) {
       point=kind + index;
       point->init = 'A'+index;
       point->jaar = 16;
       point->graad= 84;
   }
   kind[3].jaar=kind[5].jaar=17;
   kind[2].graad=kind[6].graad=92;
   for (index=0;index<12;index++) {
      point=kind+index;
      printf("%c is %d jaar en heeft een graad van %d\n",(*point).init,
            kind[index].jaar,point->graad);
      }
      extra=kind[12];
      extra=*point;
}
```

Het eerste verschil gebeurt al bij de aanmaak van de struktuur, hier word er een "pointer" met de naam "point" aangemaakt, welke een "pointer" is naar de struktuur. Zoals in het hoofdstuk "pointers" kunnen "point" niet naar een ander datatype laten wijzen. In het hoofdstuk "strings & array's" hebben we ook gezien dat een array een verzameling is van pointers. We kunnen "point" dus zonder problemen gelijk stellen aan "kind" welke een "pointer" is naar het eerste variabele, hier een struktuur, uit de array "kind[..]". C weet hoeveel geheugen plaatsen het moet reserveren voor een struktuur. Tellen we bij "kind" bv. 1 bij dan zal C zich op de tweede struktuur zetten, in werkelijkheid wordt er bij "kind" dus de struktuurgrootte bijgeteld.

10.4 Strukturen zonder naam.

```
main()
{
struct persoon {
       char naam[25];
       int jaar;
       char status;        /*G = getrouwd   V = vrijgezel */
       };
struct data {
       int graad;
       struct persoon beschr;
       char eten[25];
       } student[53];
struct data leraar,sub;
leraar.graad=94;leraar.beschr.jaar=34;leraar.beschr.status='G';
strcpy(leraar.beschr.naam,"Eddy Smith");
strcpy(leraar.eten,"eet niet op school");
printf("%s is %d jaar  status:  %c  eten:  %s  graad:  %d\n",leraar.beschr.naam,
     leraar.beschr.jaar,leraar.beschr.status,leraar.eten,leraar.graad);
sub.beschr.jaar=87;sub.beschr.status='M';
strcpy(sub.beschr.naam,"Dikke oude Linda");
sub.graad=73;
strcpy(sub.eten,"Yogurt en brood");
printf("%s is %d jaar  status:  %c  eten:  %s  graad:  %d\n",sub.beschr.naam,
     sub.beschr.jaar,sub.beschr.status,sub.eten,sub.graad);
}
```

De eerste struktuur "persoon" in het bovenstaande voorbeeld wordt niet gevolgd door een variabele naam, we hebben dus enkel een struktuur aangemaakt. Het nut hiervan wordt duidelijk gemaakt in de volgende struktuur "data", hierin wordt de struktuur "persoon" gebruikt als variabele "beshr". Door dus een struktuur zonder een variabele naam aan te maken kunnen we deze gebruiken in heel het programma zoals een ander datatype (int, char, long ...).

10.5 Wat is een "union"??

Het volgende programma is een voorbeeld van een "union". In dit voorbeeld bestaat de "union" uit twee delen, het eerste deel is een integer "waarde" welke in 2 bytes in het geheugen van de komputer bewaard wordt. Het tweede deel bevat twee karakter (welke elk 1 byte groot zijn) nl. "een" en "twee". Deze twee karakters worden op dezelfde geheugenplaats bewaard als de integer "waarde", dit komt door de instruktie "union". Veranderen we de integer "waarde" dan "een" het eerste deel van "waarde" bevatten en "twee" het tweede. Dit wordt in het programma aangetoond door de "for" lus.

```
main()
{
union {
   unsigned waarde;
   struct {
      unsigned char een;
      unsigned char twee;
      } helft;
   } nummer;
long index;
   for (index=1;index<65535;index+=3000) {
```

```
        nummer.waarde=index;
        printf ("%8x %6x %6x\n",nummer.waarde,nummer.helft.twee,
                nummer.helft.een);
    }
}
```

10.6 Een ander voorbeeld ...

Het volgende programma is een meer praktisch voorbeeld, het houdt een database(je) bij van verschillende types van voertuigen. We zullen dit voorbeeld doorlopen van begin tot einde ...

```
#define AUTO  1
#define BOOT  2
#define LUCHT 3
#define SCHIP 4
main()
{
struct automobile {             /* struktuur voor automobile            */
   int banden;
   int schokdempers;
   int deuren;
};
typedef struct {                /* struktuur voor een boot of schip     */
   int plaats;
   int lengte;
} BOATDEF;
struct {
   char voertuig;               /* welke type voertuig                  */
   int  gewicht;                /* globaal gewicht v/h voertuig */
   union {
      struct automobile wagen;  /* deel 1 v/d union     */
      BOATDEF boot;                     /* deel 2 v/d union     */
      struct {
         char motors;
         int  breedte;
      } vliegtuig;              /* deel 3 v/d union     */
      BOATDEF ship;             /* deel 4 v/d union     */
   } voertuig_type;
   int  waarde;         /* waarde van het voertuig              */
   char eigenaar[32];   /* eigenaars naam                       */
} ford,schelde,f16;     /* de drie namen van de strukturen      */
    ford.voertuig = AUTO;
    ford.gewicht  = 2742;
    ford.voertuig_type.wagen.banden = 5;
    ford.voertuig_type.wagen.deuren = 2;
    schelde.waarde = 3742;
    schelde.voertuig_type.boot.lengte = 20;
    f16.voertuig = LUCHT;
    f16.voertuig_type.vliegtuig.breedte = 27;
    if (ford.voertuig == AUTO)  /* wat zo is!  */
       printf("De ford heeft %d banden.\n",ford.voertuig_type.wagen.banden);
    if (f16.voertuig == AUTO)   /* wat nietwaar is */
       printf("De f16 heeft %d banden.\n",f16.voertuig_type.wagen.banden);
}
```

We beginnen met het aanmaken van een paar konstanten met "#define". Hierna definiren we een struktuur "automobile" welke enkele data velden bevat, dit is vrij duidelijk vermits we hier niets nieuws doen, we definiren hier enkel de struktuur en maken nog gn variabelen aan.

10.7 Een nieuwe instruktie "typedef".

Hierna definiren we data met een nieuwe instruktie nl. "typedef". Hiermee definiren we een kompleet nieuw datatype welke gebruikt kan worden zoals bv. "int" of "char" gebruikt kan worden. Merk op dat de gedefinieerde struktuur geen naam heeft, op het einde waar normaal de variabele naam staat is nu ingenomen door "BOATDEF". We hebben nu een nieuw datatype "BOATDEF" welke gebruikt kan worden om een struktuur aan te maken waar we ook maar willen. Merk opnieuw op dat we hier gn variabelen aanmaken, maar enkel een nieuwe struktuur definiren.

Uiteindelijk definiren we een grote struktuur hierbij maken we gebruikt van de datatypes welke we hiervoor gedefinieerd hadden. De struktuur is opgebouwd uit 5 delen, twee eenvoudige variabelen "voertuig" en "gewicht", gevolgd door een "union" en als laatste we andere eenvoudige variabelen "waarde" en "eigenaar". Laten we even bekijken hoe de "union" is opgebouwd. Deze is gemaakt uit vier delen, het eerste deel is de variabele "wagen" welke aangemaakt wordt aan de hand van de struktuur "automobile" welke we eerder gedefinieerd hadden. Het tweede deel is de variabele "boot" welke een struktuur is zoals we deze eerder al gedefinieerd hadden in "BOATDEF". Het derde deel is de variabele "vliegtuig" welke aangemaakt wordt door een struktuur. Het vierde en het laatste deel is de variabele "ship" welke ook van het type "BOATDEF" is.

We hebben nu een struktuur welke gebruikt kan worden om elk van de vier verschillende types van data strukturen te bewaren. De grootte van welk record zal de grote zijn van de grootste "union". In dit geval is het eerste deel van de union het grootste vermits het bestaat uit 3 integers. Het eerste lid van union zal dus de werkelijke grootte bepalen. De resulteerde struktuur kan gebruikt worden om een van de vier datetypes te bewaren, maar het is aan de programmeur om bij te houden wat hij bewaard heeft. Hiervoor dient de variabele "voertuig" hiermee houden we bij welke type van voertuig (en dus ook het datatype) in de struktuur bewaard werd. De vier konstanten welke in het begin van het programma aangemaakt werden worden hiervoor gebruikt.

In de volgende regels wordt aangetoond hoe de aangemaakte struktuur gebruikt kan worden. Aan enkele wordt een waarde toegekend, enkele worden afgedrukt als illustratie.

Een "union" wordt niet al te vaak gebruikt, en bijna nooit door een beginnende programmeur. Je zult toch in sommige programma's tegen komen het is dus zeker de moeite om te weten wat een "union" is. Je moet zeker niet in het begin alle details te weten, besteed er dus niet t veel tijd aan. Het is beter deze zaken te nader te bekijken wanneer je ze nodig hebt.

10.8 Wat is een "bitveld"?

Het volgende programma is een voorbeeld van een "bitveld" (bitfield voor de engelstaligen). In dit programma maken we en "union" aan bestaat uit een integer "index" en bijgevolg 4 bytes groot is. De "union" bevat nog een struktuur welke 3 leden bevat, deze struktuur wordt op dezelfde geheugenplaats bewaard als de integer "index". De variabele "x" is enkel 1 bit breed, "y" en "z" zijn er 2 breed. Vermits de struktuur op dezelfde geheugenplaats bewaard wordt is "x" de minst beduiden bit, "y" bevat devolgende 2 en "z" de laatste twee.

Indien we "x", "y" en "z" elk n bit groot maken tellen we binair.

```
main()
{
union {
```

```
    int index;
    struct {
       unsigned x :  1;
       unsigned y :  2;
       unsigned z :  2;
    } bits;
} nummer;
    for (nummer.index = 0;nummer.index < 20;nummer.index++)
       printf("index = %3d, bits = %3d%3d%3d\n",nummer.index,nummer.bits.z,
              nummer.bits.y,nummer.bits.x);
}
```

10.9 Programmeer Opdrachten.

1.Definieer een struktuur welke een veld bevat voor een string om naam in te bewaren, een integer voor voeten en een voor armen. Zet in deze struktuur zes keer data, en print deze hierna uit.

Een mens heeft 2 voeten en 2 armen

Een hond heeft 4 voeten en 2 armen

Een stoel heeft 4 voeten en 0 armen

Een tafel heeft 4 voeten en 0 armen

...

2. Herschrijf 1. maar maak nu gebruik van "pointers".

11 Dynamisch geheugen

Bij dynamisch geheugen gaan we in ons programma pas geheugen bezetten als we het nodig hebben. Tot nu hebben we altijd gebruikt gemaakt van statisch geheugen, we definieerde een variabele en deze nam geduurde heel het programma geheugen in beslag. Met dynamisch geheugen kunnen we enkel geheugen gebruiken indien we het nodig hebben, met deze techniek kunnen we dus geheugen besparen. Bovendien weten we niet altijd op voorhand hoeveel geheugen we gaan nodig hebben.

11.1 Een voorbeeld ...

Aan de hand van het volgende voorbeeld zullen we de belangrijkste instruktie en begrippen die met dynamisch geheugen te maken hebben bekijken.

```
main()
{
struct dier {
   char naam[25];
   char ras[25];
   int leeftijd;
} *pt1, *pt2, *pt3;
  pt1= (struct dier *) malloc(sizeof(struct dier));
  strcpy(pt1->naam,"Brom");
  strcpy(pt1->ras,"Beer");
  pt1->leeftijd=1;
  pt2=pt1;      /* pt2 wijst nu naar de bovenstaande struktuur */
  pt1=(struct dier *) malloc(sizeof(struct dier));
  strcpy(pt1->naam,"Frank");
  strcpy(pt1->ras,"Labrador");
  pt1->leeftijd=3;
  pt3=(struct dier *)malloc(sizeof(struct dier));
  strcpy(pt3->naam,"Kristien");
  strcpy(pt3->ras,"Poesje");
  pt3->leeftijd=4;
               /* Afdrukken van de data */
  printf("%-10s is een %-10s en is %d jaar oud\n",pt1->naam,pt1->ras,
          pt1->leeftijd);
  printf("%-10s is een %-10s en is %d jaar oud\n",pt2->naam,pt2->ras,
          pt2->leeftijd);
  printf("%-10s is een %-10s en is %d jaar oud\n",pt3->naam,pt3->ras,
          pt3->leeftijd);
   pt1=pt3; /* pt1 wijst nu dezelfde struktuur als pt3 */
   free (pt3);  /* Dit maakt de struktuur vrij waar pt3 naar wijst */
   free (pt2);  /* Dit maakt de struktuur vrij waar pt2 naar wijst */
/* free (pt1);  Dit kan niet !!!  */
}
```

We starten het programma met het definiren van een struktuur dier, welke drie velden bevat. Merk op dat we geen enkele variabele aanmaken (en dus ook gn geheugen bezetten!), we definiren enkel drie pointers. In het programma zullen alle variabelen die er gebruikt worden dynamisch worden aangemaakt.

11.2 Het dynamisch kreren van variabelen.

Na het definiren van de pointers wordt er een variabele dynamisch aangemaakt, "pt1" zal naar een struktuur wijzen welke 3 variabelen bevat welk eerder in het programma gedefinieerd werden. Het hart van de aanmaak is de "malloc()" funktie, hiermee kunnen we geheugen bezetten. Met "malloc(n)" bezetten we een geheugendeel wat n bytes groot is in de "heap".

11.3 Wat is een "heap"?

Elke kompiler heeft zijn beperkingen op hoe groot een uitvoerbaar bestand kan zijn, hoeveel variabelen er gebruikt kunnen worden, hoe groot de bronkode mag zijn, ... Een van deze beperkingen bij DOS C-kompilers is de limiet van 64Kb, bij C-kompilers van andere besturingssystemen zoals Unix, OS/2 of een kompiler onder DOS met een zgn. DOS Extender bestaat deze beperking niet.Een "heap" is gebied van geheugen buiten deze 64Kb limiet welke gebruikt kan worden in het programma om variabelen in te bewaren (meestal via pointers). Met de "malloc()" funktie bezetten we geheugen in de "heap". C houdt bij waar geheugen bezet werd, het is ook mogelijk om een geheugengebied terug vrij te geven, welke gaten in de "heap" achter laat. Bij het opnieuw bezetten van geheugen zullen deze gaten, indien mogelijk, terug opgevuld worden.

11.4 De "sizeof()" funktie.

Met "sizeof()" kunnen we de grootte in bytes berekenen van een variabele, struktuur, enz. In dit voorbeeld gebruiken we de funktie in kombinatie met "malloc", om geheugen te bezetten welke juist voldoende plaats inneemt om de struktuur in te bewaren.

11.5 Wat is een "cast"?

Voor de "malloc" funktie hebben we nog een vreemd uitziende konstruktie, dit noemt men een "cast". Standaard geeft "malloc()" een "pointer" terug welke naar een karakter wijst (omdat een "char" 1 byte groot is), maar vaak hebben we geen "pointer" naar een karakter nodig. Het is echter mogelijk om de "pointer" welke "malloc()" teruggeeft om te rekenen naar een "pointer" welke wijst naar het datatype dat we in ons programma nodig hebben. Dit is precies wat deze "rare" konstruktie doet, het vertelt C dat we een "pointer" nodig naar een struktuur die er uitziet zoals "dier".

11.6 Het gebruiken van dynamisch aangemaakt geheugen.

Indien je de teorie over "pointers" en "strukturen" goed begrepen hebt moet de rest van het programma vrij duidelijk zijn. Via de "pointers" "pt1", "pt2" en "pt3" worden er dynamisch strukturen in het geheugen aangemaakt, via deze pointers wordt er ook data in deze strukturen geplaatst zoals we in het vorige hoofdstuk gezien hebben.

11.7 Het terug vrijmaken van geheugen met "free()".

Een andere nieuwe instruktie in het programma is "free(pt)", waar "pt" een "pointer" is naar het geheugenblok dat we terug wensen vrij te geven. Nadat in het programma de data afgeprint is stellen we "pt1" gelijk aan "pt3", hierdoor hebben we dus geen "pointer" meer naar het datagebied waar "pt1" naar wees. We kunnen dit datagebied dus ook niet meer vrijgeven! Dit voorbeeld is dit

echter geen probleem, bij het verlaten van het programma wordt al het eerder bezette geheugen terug vrijgegeven.

11.8 Een array van pointers.

Het volgende programma is een andere voorbeeld van dynamisch geheugen, hier maken we gebruik van array van pointers.

```
#include <malloc.h>
main()
{
struct dier {
   char naam[25];
   char ras[25];
   int  leeftijd;
} *pt[12],*point; /* Hiermee definiren we 13 pointers, gn variabelen */
int index;
     /* aanmaken van van dynamische strukturen met nonsens data */
   for (index=0;index<12;index++) {
      pt[index]=(struct dier *)malloc(sizeof(struct dier));
      strcpy(pt[index]->naam,"Kristien");
      strcpy(pt[index]->ras,"poesje");
      pt[index]->leeftijd=4;
   }
   pt[4]->leeftijd=12;
   pt[5]->leeftijd=15;
   pt[6]->leeftijd=10;
   /* het afdrukken van de data */
   for (index=0;index<12;index++) {
      point=pt[index];
      printf("%15s is een %10s, en is %d jaar oud.\n",point->naam,
             point->ras,point->leeftijd);
   }
   /* Het is een goede programmeer gewoonte om al het bezette geheugen
      terug vrij te geven */
   for (index=0;index<12;index++) free(pt[index]);
}
```

We beginnen het programma met het "malloc.h" aan ons programma toe te voegen, dit is echter niet bij alle C-kompilers nodig. We gebruiken dezelfde struktuur als in het voorgaande programma, maar deze keer maken we array van 12 "pointers" plus een ekstra "pointer" aan. Met } *pt[12],*point; maken deze aan. Zoals we als gezien hadden is een array al een "pointer", dus is bv. "pt[0]" een "pointer" naar een "pointer" welke het begin van de struktuur aanwijst. Dit komt in C vaker voor zo is bij "int ****pt", "pt" een "pointer" naar een "pointer" naar een "pointer" naar een "pointer" welke een integer aanwijst.

De rest van het programma komt vrij bekend over, nu we 12 pointers hebben gebruiken we een "for" lus om dynamisch struktuur variabelen aan te maken. Met de volgende "for" plaatsen we wat data in deze strukturen, daarna passen we via "pointers" enkele leeftijden aan. Met de voorlaatste "for" lus drukken we de velden van de strukturen af. De laatste "for" lus dient om het bezette geheugen terug vrij te geven, dit was in principe niet nodig vermits bij het verlaten van het programma dit geheugen toch terug vrijgegeven zou worden. Het echter een goede gewoonte om telkens indien we eerder bezet geheugen niet meer nodig hebben dit terug vrij te geven.

11.9 Een gelinkte lijst

Bij een gelinkte lijst of "linked list" in het engels wijst het eerste element (via een "pointer") de tweede aan, het tweede element wijst op zijn beurt naar het derde, de derde naar het vierde, ... enz ... Het volgende programma is hier een voorbeeld van.

```
#include <malloc.h>
#include <stdio.h>      /* dit is enkel nodig om NULL te definiren */
#define RECORDS 6
main()
{
struct dier {
   char naam[25];         /* De naam van het dier         */
   char ras[25];          /* Het type van dier            */
   int leeftijd;          /* De leeftijd van het dier     */
   struct dier *volgende; /* Een pointer naar een ander variabele van dit type*/
}*point,*start,*vorige;   /* Definiren van 3 pointer, geen aanmaak van variabelen*/
int index;
                /* De eerste is altijd speciaal */
    start=(struct dier*)malloc(sizeof(struct dier));
    strcpy(start->naam,"Kristien");
    strcpy(start->ras,"Poes");
    start->leeftijd=4;
    vorige=start;
              /* Met een 'for' lus vullen we de andere in */
    for(index=0;index<RECORDS;index++) {
       point=(struct dier*)malloc(sizeof(struct dier));
       strcpy(point->naam,"Frank");
       strcpy(point->ras,"Hond");
       point->leeftijd=3;
       vorige->volgende=point;/* Het vorige record wijst nu het huidige aan */
       point->volgende=NULL;  /* Hiermee geven we het einde aan */
       vorige=point;          /* point is nu het 'vorige' record */
}
             /* Het afdrukken van de data */
     point=start;
     do {
        vorige=point->volgende;
        printf("%15s is een %10s, en is %d jaar oud\n",point->naam,
               point->ras,point->leeftijd);
        point=point->volgende;
     } while (vorige != NULL);
  /* Het is een goede programmeer gewoonte om het bezette geheugen altijd terug
vrij te geven */
     point=start;                  /* Het eerste geheugengebied        */
     do {
        vorige=point->volgende;     /* Het volgende geheugengebied      */
        free(point);               /* Vrijgeven van het geheugengebied */
        point=vorige;              /* point = adres v/h volgende       */
     } while (vorige!=NULL);
}
```

Het programma start op ongeveer dezelfde manier als de vorige programma's. We met het definiren van een konstante "RECORDS" welke het aantal variabelen aan geeft. Hierna maken we een struktuur aan welke er bijna hetzelfde uitziet als in de twee vorige programma's, enkel hebben we via struct dier *volgende een ekstra "pointer" in de struktuur opgenomen. Deze zal het adres

bevatten van de volgende variabele. We definiren drie "pointers" naar deze struktuur plus een integer "index" welke we later zullen gebruiken als teller.

Via de "malloc" funktie maken zoals in de vorige voorbeeld een eerste variabele aan, het adres van deze variabele wordt in de pointer "start" bewaard. Dit adres onthouden we ook de pointer "vorige".

Met de eerste "for" lus maken we onze lijst aan, telkens de lus doorlopen wordt bezetten we geheugen, kopiren we data in de juist aangemaakte variabele, en vullen we de pointers met de juiste adressen. De pointer "vorige" bevat het adres van de vorige variabele, met vorige->volgende=point kopiren we het adres van de nieuwe struktuur in de pointer van de oude. Telkens we een nieuwe struktuur variabele aan maken stellen we de pointer hiervan gelijk aan "NULL", dit doen we om het einde van de lijst aan te geven.

We doorlopen de lus zes keer, na het einde van de deze "for" hebben we een lijst aangemaakte welke er zo uitziet.

1. De pointer "start" wijst naar de eerste struktuur in de lijst.

2. Elke struktuur bevat een pointer welke naar de volgende struktuur wijst.

3. De laatste struktuur bevat een pointer welke gelijk is aan "NULL", hiermee geven we het einde van de lijst aan.

start->struktuur1

naam

ras

leeftijd

volgende->struktuur2

naam

ras

leeftijd

volgende-> struktuur7

naam

ras

leeftijd

volgende->"NULL"

Het is duidelijk dat we via een dergelijke opbouw het niet mogelijk is om naar bv. het derde element in de lijst te gaan. De enige manier om dit element te bereiken is via het eerste te beginnen, en zo in de lijst "af te dalen". Niet alle datatypes zijn dus geschikt om bewaard te worden in een dergelijke lijst, omdat het te veel tijd vraagt om zo een bepaald element op te zoeken. In principe kunnen we de strukturen in de lijst zo aanpassen dat ze twee pointers bevatten, n naar het volgende element, en een andere naar het vorige om het opzoekingswerk te vereenvoudigen.

Met de "do ... while" lus drukken we de data op het scherm, de metode die we hiervoor gebruiken is analoog met hoe we de data genereert hebben. De lus wordt doorlopen totdat de pointer van de struktuur gelijk is aan "NULL".

Met de laatste "for" lus geven we het bezetten geheugen terug, dit was ook nu niet echt nodig, maar het is een goede gewoonte dit wel te doen. Ja ... Ik val in herhaling ;-)

11.10 De "calloc" funktie.

De "calloc" funktie lijkt hard op de "malloc" funktie, enkel vult het de bezette datablok met nullen, wat praktisch kan zijn in sommige gevallen.

11.11 Programmeer opdrachten!

1.Herschrijf het eerste programma van hoofdstuk 10 zo dat de twee strukturen dynamische aangemaakt worden.

2.Herschrijf het tweede programma van hoofdstuk 10 zo dat de 12 strukturen dynamisch worden aangemaakt.

12 Karakter & bit manipulatie

In dit laatste hoofdstuk gaan we nog enkel funkties bekijken voor karakter en bit manipulatie. Zo beschikt C over funkties voor konversie naar grootte en kleine letters, schuif ("shift") instrukties...

12.1 Grote & kleine letters.

Grote en kleine letters hebben een andere ASCII - waarde, zo komt "a" overeen met 97 en "A" met 65. Dit kan erg vervelend zijn voor sommige toepassingen zoals bv. sorteren. C heeft hiervoor enkele funkties om deze problemen op te lossen.

isupper(<char>)	:	Is het een kleine letter?
islower(<char>)	:	Is het een grote letter?
toupper(<char>)	:	Maak van het karakter een grote letter.
tolower(<char>)	:	Maak van het karakter een kleine letter.

Het volgende programma illustreert deze vier funkties, het leest een bestand regel per regel en drukt dit bestand op het scherm af. Alle grote letters worden verandert in kleine, alle kleine letters worden verandert grootte. Indien je de vorige hoofdstukken goed begrepen hebt zal het niet moeilijk zijn de werking van het programma te begrijpen.

```
#include <stdio.h>
#include <ctype.h>
void verander(char lijn[]);
main()
{
FILE *fp;
char lijn[80],bestand[24];
char *c;
   printf("Geef een bestandsnaam ->");scanf("%s",bestand);
   fp=fopen(bestand,"r");
   do {
      c=fgets(lijn,80,fp); /* haal een lijn tekst uit het bestand */
      if (c!=NULL) verander(lijn);
   } while (c!=NULL);
   fclose(fp);
}
/* Deze procedure maakt van alle kleine letters grootte, en van alle grootte
   kleine letters */
void verander(char lijn[])
{
int index;
   for(index=0;lijn[index]!=0;index++) {
      if (isupper(lijn[index]))   /* 1 indien een grootte letter */
         lijn[index]=tolower(lijn[index]);
      else {
         if (islower(lijn[index])) /* 1 indien een kleine letter */
            lijn[index]=toupper(lijn[index]);
      }
   }
   printf("%s",lijn);
}
```

12.2 Verschillende soorten van karakters.

C maakt onderscheidt tussen bepaalde soorten van karakters, zo hebben we bv. kontrole karakter welke we al in "printf" gebruikt hebben. Misschien heb je al afgevraagd hoe we een ' " ' met "printf" kunnen afdrukken, zonder dit aanzien wordt als - einde van de data -, of hoe het mogelijk we is een "\" af te drukken. Ook hier is een oplossing voor onderstaande lijst vervolledig de lijst van kontrole karakters.

\n	volgende lijn
\t	tab
\b	"backspace"
\\	"backslash"
\"	aanhallingstekens
\0	NULL

C heeft funkties om te testen met wat voor type van karakter we te maken hebben.

isalpha(<char>)	:	is het een normaal karakter?
isdigit(<char>)	:	is het een cijfer (digit)?
isspace(<char>)	:	is het spatie of een \n , \t?

Het volgende programma laadt opnieuw een bestand in, en telt per regel het aantal karakters, cijfers en spaties.

```
#include <stdio.h>
#include <ctype.h>
void tel_de_data(char lijn[]);
main()
{
FILE *fp;
char lijn[80],bestand[24];
char *c;
   printf("Geef een bestandsnaam -> ");scanf("%s",bestand);
   if((fp=fopen(bestand,"r"))==NULL) {
      printf("Sorry, ik kan het bestand niet laden");exit(1);}
   do {
      c=fgets(lijn,80,fp);  /* haal een regel op */
      if (c!=NULL) { tel_de_data(lijn); }
   } while (c!=NULL);
   fclose(fp);
}
void tel_de_data(char lijn[])
{
int spatie,kar,cijfer;
int index;
  spatie=kar=cijfer=0;
  for(index=0;lijn[index]!=0;index++) {
     if (isalpha(lijn[index])) kar++;
     if (isdigit(lijn[index])) cijfer++;
     if (isspace(lijn[index])) spatie++;
  }
  printf("%3d%3d%3d %s",spatie,kar,cijfer,lijn);
}
```

12.3 Logische bewerkingen.

In de volgende lijst staan enkele logische bewerken:

&	Logische AND	:	indien alle bits 1 zijn is het resultaat ook 1
|	Logische OR	:	indien n van bits 1 is, is het resultaat ook 1
^	Logische XOR	:	indien n en allen n bit 1 is, is het resultaat ook 1
~	Inverteer	:	maakt van 0 een 1 en van 1 een 0

Het volgende programma demonstreert bovenstaande bewerkingen.

```
#include <stdio.h>
main()
{
char mask;
char getal[6];
char and,or,xor,inv,index;
   getal[0]=0X00;
   getal[1]=0X11;
   getal[2]=0X22;
   getal[3]=0X33;
   getal[4]=0X88;
   getal[5]=0XFF;
   printf(" getal     mask     and     or     xor     inv\n");
   mask=0X0F;
   for (index=0;index<=5;index++) {
      and = mask & getal[index];
      or  = mask | getal[index];
      xor = mask ^ getal[index];
      inv = ~getal[index];
      printf("%5x %5x %5x %5x %5x %5x\n",getal[index],mask,and,or,xor,inv);
   }
   printf("\n");
   mask=0X22;
   for (index=0;index<6;index++) {
      and = mask & getal[index];
      or  = mask | getal[index];
      xor = mask ^ getal[index];
      inv = ~getal[index];
      printf("%5x %5x %5x %5x %5x %5x\n",getal[index],mask,and,or,xor,inv);
   }
}
```

12.4 Schuif bewerkingen.

Ook is het mogelijk om de bits van een getal naar links of rechts te verschuiven, dit gebeurt op de volgende manier.

<<n	Schuif n plaatsen naar links
>>n	Schuif n plaatsen naar rechts

Het volgende programma illustreert dit.

```
main()
{
int klein,groot,index,teller;
   printf("       shift left       shift right\n\n");
   klein=1;groot=0X4000;
   for(index=0;index<17;index++) {
      printf("%8d %8d %8d %8d\n",klein,klein,groot,groot);
      klein=klein<<1;groot=groot>>1;
   }
   printf("\n");
   teller=2;klein=1;   groot=0X4000;
   for(index=0;index<9;index++) {
      printf("%8d %8d %8d %8d\n",klein,klein,groot,groot);
      klein=klein<<teller;groot=groot>>teller;
   }
}
```

13 Gebruikte Software

Debian Linux 1.3

LyX 0.10.7 Beta

LaTeX

Ghostscript

Ghostview

XEmacs

GNU C (gcc)