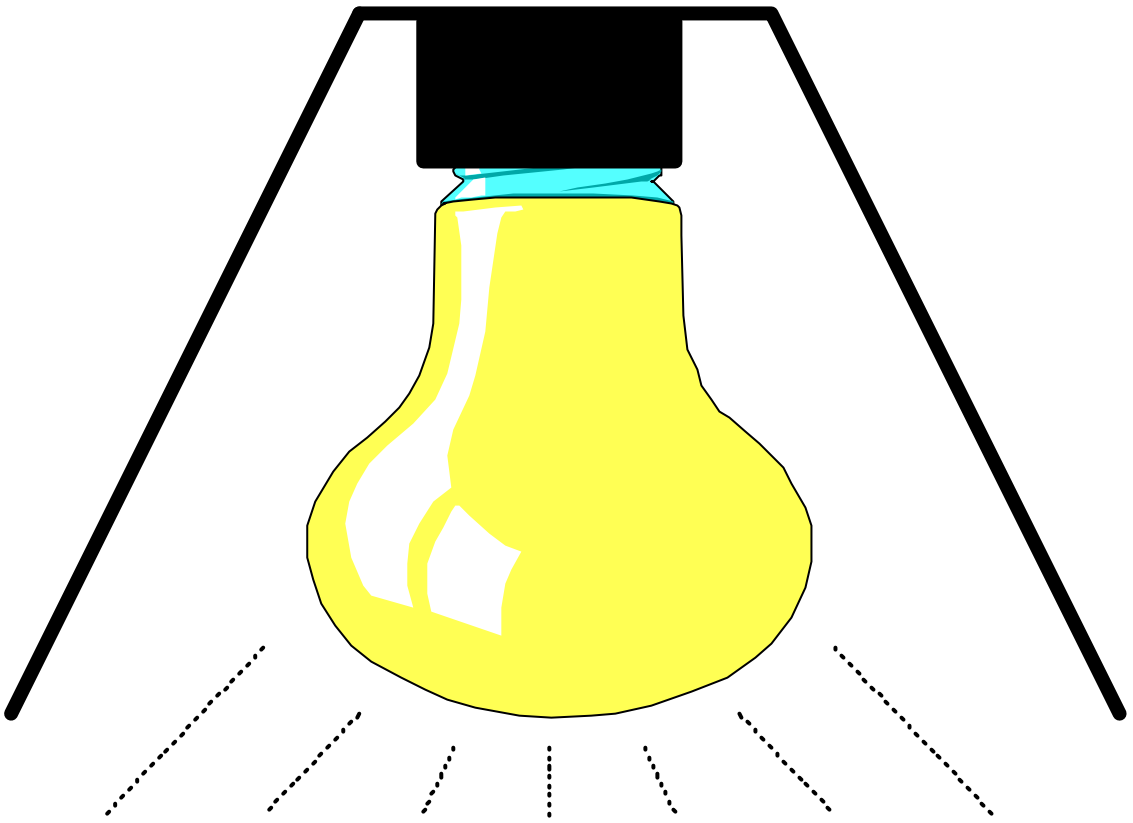
C
in vogelvlucht



```
void wissel(int *, int *);  /* prototypering */

void main()
{ int i =10, j =20;
  wissel( &i, &j);
  printf("%2d %2d\n", i, j);
}

void wissel(int *px, int *py) /* declaratie */
{ int hulp;
  hulp = *px; *px = *py; *py = hulp;
}
```

7 november 2000

Inhoudsopgave

1 Een Pascal en C programma

```
program koopbon;

const PERCENTAGE = 0.185;

var kp, btwb, tot: real;

function btw(bedrag: real): real;
begin
  btw := bedrag * PERCENTAGE;
end;


begin
  write('Ex-btw-prijs: ');
  readln(kp);

  btwb := btw( kp );
  tot := kp + btwb;

  writeln;
  writeln(' Prijs: fl ', kp:0:2);
  writeln('   BTW: fl ', btwb:0:2);
  writeln('Totaal: fl ', tot:0:2);
  end.
```

```
#define PERCENTAGE 0.185
float btw(float);
float kp, btwb, totaal;

float btw(float bedrag)
{
  return bedrag * PERCENTAGE;
}

void main()
{
   printf("Ex-btw-prijs: ");
   scanf("%f", &kp);

   btwb = btw(kp);
   tot = kp + btwb;

   printf("\n");
   printf(" Prijs: fl %.2f\n", kp);
   printf("\n   BTW: fl %.2f\n", btwb);
   printf("Totaal: fl %.2f\n", tot);
}
```

2 Inleiding

De globale structuur van een C-programma is eenvoudig. Het is een verzameling procedures (functies). Deze worden niet genest zoals bij Pascal, maar simpelweg achter elkaar gezet. Het skelet van zo'n functie ziet er grofweg als volgt uit:

```
[Type] Naam_van_de_functie( [ parameters ] )
{
  [ declaratie van lokale variabelen ]

  [ C statements ]
}
```

Wat tussen de accolades staat, heet het 'blok' van de functie. Alleen de naam (vrij te kiezen), de haakjes en de accolades zijn verplicht.
Ook het 'hoofdprogramma' is een functie genaamd 'main'. Vanuit deze hoofdfunctie kunnen de andere functies aangeroepen worden.

De voordelen van de functie-structuur zijn dat ze de mogelijkheid bieden een groot complex probleem te verdelen in kleine overzichtelijke problemen.

Opvallend is de overeenkomst van de vorm van het compound-statement met die van de body van een functie:

```
{
  [ declaratie van lokale variabelen ]

  [ C statements ]
}
```

Wat tussen de accolades staat, heet het 'blok' van het statement.

Globale variabelen worden aan het begin van het programma gedeclareerd, dus buiten de functies.

Samenvattend het skelet van een C programma:

```
prototyperingen
declaratie van globale variabelen

main()
{
  blok0
}

functie1( parameters )
{
  blok1
}
        .
        .

functieN( parameters )
{
  blokN
}
```

3 Types en constanten

3.1 Types (enkelvoudige)

int	:standaard-integer
short	:korte integer
long	:lange integer
char	:karakter
float	:drijvende komma, kort
double	:drijvende komma, lang
void	:'procedure'-type, zie later

Er is dus geen boolean type. Dit wordt gecompenseerd door het feit dat elke expressie na evaluatie een waarde krijgt. Is die waarde ongelijk aan 0 dan is de logische uitkomst TRUE, anders FALSE.

Het voorvoegsel 'unsigned' mag gebruikt worden voor de types 'int', 'short' en 'char'. In dat geval wordt het hoogste orde bit niet als tekenbit geinterpreteerd.

3.2 Constanten

		Voorbeeld	
- gehele getallen:	decimale	127	
	octale	0127	
	hexadecimale	0X127	
- drijvende komma:		127.	
		.721	
		127.721	
		1.27721E2	
		1277.21E-1	
- karakters:	tussen enkelvoudige aanhalingstekens	'a'	
		'1'	
- speciale karakters:		'\n'	nieuwe regel
		'\t'	tabulatie
		'\b'	backspace
		'\r'	terugloop wagen
		'\f'	nieuwe pagina
		'\\'	backslash
		'\"	aanhalingsteken

- strings: verzameling karakters tussen dubbele aanhalingstekens
  V.B.: "A\"Bc\\D" betekent A"Bc\D

4 Expressies en operatoren

Een 'expressie' is opgebouwd uit door de compiler herkenbare eenheden van karakters. Deze eenheden zijn:

- identifiers: namen waarmee variabelen worden aangeduid.
- sleutelwoorden: gereserveerde woorden die voor de C-compiler een speciale betekenis hebben.
- constanten
- strings: een rij karakters tussen dubbele aanhalingstekens.
- operatoren
- separatoren: zoals spaties, tabulaties, commentaar (alles wat tussen /* en */ staat).

De term 'expressie' is in C een breed begrip: C kent zowel 'wiskundige' als 'conditionele' expressies. Daarom zijn vergelijkingstekens als < en > ook operatoren. In de syntax van C vallen op zichzelf staande identifiers, constanten en strings onder de term 'expressie'. Deze zgn 'primaire' expressies zijn triviaal en blijven daarom buiten beschouwing.

De behandeling van expressies gebeurt hier aan de hand van operatoren die gebruikelijk zijn in expressies. Er wordt onder scheid gemaakt tussen unaire, binaire en drievoudige operatoren.

4.1 Operatoren met 1 operand

Een unaire operator wordt gebruikt voor de variabele of expressie waar ze betrekking op hebben. De unaire operatoren zijn:

!	ontkenning van conditie
~	1-complement van binaire waarde van variabele
++	incrementeer de variabele
--	decrementeer de variabele
-	tegengestelde van de variabele
(type_id)	converteer het type van de variabele
*	inhoud van het veld waar de betrokken wijzer-variabele naar wijst
&	adres van de betrokken variabele
sizeof(Var)	geef lengte van Var

De werking van ++ en -- is, als enige uit de rij, positie-afhankelijk. Stel een variabele komt in een expressie voor met als voorvoegsel ++, dan wordt de variabele eerst geincrementeerd daarna wordt de expressie uitgewerkt. Staat de operator achter de variabele, dan wordt eerst de expressie uitgewerkt en daarna de variabele geincrementeerd. Dit geldt ook voor --.

De (type_id) operator wordt behandeld in paragraaf Type-conversie.

De * en & operatoren worden behandeld in paragraaf Wijzers.

4.2 Operatoren met 2 operanden

De binaire operatoren zijn:

- rekenkundig:	+	optellen
	-	aftrekken
	*	vermenigvuldigen
	/	delen (afhankelijk van type)
	%	modulo (rest na integer deling)
- logisch:	<	kleiner
	>	groter
	<=	kleiner gelijk
	>=	groter gelijk
	==	gelijk
	!=	niet gelijk
	&&	logische en
	||	logische of
- bit-gewijs:	&	and
	|	or
	^	xor
	<<	schuif links
	>>	schuif rechts
- toewijzing:	=	als := in pascal
	+=	LL = LL + RL
	-= *= /= %= <<= >>= &= ^= |=	

De meeste operatoren spreken voor zich. De schuif- en toewijzings-operatoren behoeven nadere toelichting.

Met betrekking tot de schuif verklaart het volgende voorbeeld deze operatie: het resultaat van de expressie E1 << E2 is het bitpatroon van E1 E2 plaatsen naar links geschoven.

De toewijzing heeft in C een extra dimensie. De normale toewijzing is als volgt: Var = E1. In Pascal zou dit een statement zijn maar let op, in C is dit een expressie en kan dus onderdeel zijn van samengestelde expressies. Bij evaluatie van de samengestelde expressie wordt gewerkt met Var nadat die eerst de waarde van E1 gekregen heeft.
Een voorbeeld: y = 3 * (x += 5 + (x = u = 7)) + 1
De evaluatie van deze expressie gaat als volgt:
u <- 7, x <- 7, 5 + 7 = 12, x <- 19, 3 * 19 = 57, 57 + 1 = 58, y <- 58
de waarde van de expressie na evaluatie is 58.

Met betrekking tot de += operator het volgende: in de praktijk komen veel toewijzingen voor van het type Var = Var 'oper' E1. Deze expressie kan in C gecomprimeerd worden tot Var 'oper'= E1, als 'oper' een van de operatoren uit boven vermelde lijst.

Merk op dat de volgende expressies gelijkwaardig zijn:
Var = Var + 1 Var += 1 ++Var

4.3 Operator met 3 operanden

De drievoudige operator is: ? :

Deze wordt gebruikt in de context: E1 ? E2 : E3 .
De werking is: als E1 dan E2 anders E3 .

Voorbeeld: Max = (a<b) ? b : a
dus als a<b dan Max <- b en de expressie evalueert tot b
anders Max <- a en de expressie evalueert tot a .

4.4 Prioriteitsregels voor operatoren

De prioriteit van operatoren is vermeld in onderstaande tabel.

prioriteit	operatoren
1	() [] -> .
2	! ~ ++ -- - (type_id) * & sizeof
3	* / %
4	+ -
5	<< >>
6	< > <= >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= <<= >>= &= ^= |=
15	,

4.5 Identifiers

bouwstenen:
- hoofdletters
- kleine letters
- cijfers
- onderstreepte spatie (underscore)

Let op: C maakt onderscheid tussen hoofd- en kleine letters.

4.6 Sleutelwoorden

K&R auto, break, case, char, continue, default, do, double, else, extern, float, for, goto, if, int, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, while.

ANSI const, enum, signed, void, volatile.

5 Statements

Een statement ontstaat door achter een expressie een punt-komma te zetten. In C is dit de zgn 'statement-terminator' (in tegenstelling tot de term 'statement-separator' in Algol-achtige talen).

De statements zijn hieronder opgesomd.

- toewijzingsstatement: toewijzingsexpressie;

- if statement: if (E) S1 [else S2]

- while statement: while (E) S

- for statement: for ([E1] ; [E2] ; [E3]) S
  init cond stap

- do while statement: do S while (E);

- break statement: break; verlaat een lus of switch-statement

- continue statement: continue; ga naar volgende lus

- return statement: return [E]; verlaat de functie [en geef de waarde van E terug aan de aanroeper]

- goto statement: goto Label;

- switch statement:
```
switch ( E )
{ case CONST: S1 ... Sn
  case CONST: Sn1 ... Sn2
                .
                .
  default: Sn3 ... Sn4
}
```

- compound statement: { S1 ... Sn }

Let op: een statement eindigt altijd met ; of is omsloten door {}

Het break statement is zeer goed te gebruiken in lus en switch statements. Als in zulk een statement een 'break' uitgevoerd wordt, dan wordt een sprong geforceerd naar het eerste statement na de lus of switch. Dit is vooral bij switch van belang. Het switch-statement evalueert eerst E en springt naar de berekende 'case' of naar 'default' indien aanwezig. Alle resterende statements van het switch-blok zullen dan verwerkt worden tot een 'break' uitgevoerd wordt.

6 Type-conversie

In tegenstelling tot Pascal kan een variabele van het ene type toegekend worden aan een variabele van een ander type. Indien conversie mogelijk is, dan wordt impliciet een type-conversie uitgevoerd. Dit is gevaarlijk daar zo ook ongewilde of foute conversies plaats kunnen vinden. Veiliger is goed op te letten of er conversie nodig is en zo ja, dit expliciet aan te geven met behulp van de (type_id) operator.

Voorbeeld: stel F is een float en, I en J zijn integers met waarden resp. 3 en 2.
Het statement F = I / J; levert F <- 1.0 .
Het statement F = (float)I / J; levert F <- 1.5 .
Het statement F = I / (float)J; levert F <- 1.5 .
Het statement F = (float)I / (float)J; levert F <- 1.5 .

7 Standaard-I/O-mogelijkheden

In C zijn geen standaard-statements opgenomen om invoer/uitvoer te verwerken. Dit wordt daarom gerealiseerd mbv standaard-functies. De meest gebruikte zijn hieronder opgesomd.

```
ch = getchar()  /* haal 1 karakter van standaard-I/O */

putchar(ch)     /* zend 1 karakter naar standaard-I/O */
```

Geformatteerde invoer/uitvoer:

```
scanf( "formaatreeks", Adr1, Adr2, ...)

printf( "formaatreeks", Var1, Var2, ...)
```

Let op dat scanf als parameters niet de variabelen zelf verwacht maar de adressen van de variabelen (wijzer als parameter).

formaten:	%d	decimaal formaat
	%o	octaal formaat
	%x	hexadecimaal formaat
	%c	karakter formaat
	%s	string formaat
	%f	drijvende komma formaat
	.	
	.	

Hoe deze gebruikt worden: zie de voorbeeld-programma's.

8 Voorbeeld en opgave

Voorbeeld 1

Onderstaand programma drukt van een aantal ASCII waarden het betrokken karakter, de decimale-, de octale- en de hexadecimale-waarde.
Bijzonderheden: variabele I krijgt een beginwaarde bij declaratie,
scanf vereist als parameter het adres van een variabele,
in statement Val = Str & 0x00ff vindt impliciete type-conversie plaats.

```
#include <stdio.h>
void main()
{     int  e, i =0;
      char ch;
      printf("Vanaf (decimaal) ");
      scanf("%d", &e); ch = e;
      printf("Hoeveel ? ");
      scanf("%d", &e);
      printf("\n");
      do {
            int j;
            for (j = 1; j < 6; j++) {
                  char str, hstr;
                  int val;
                  str = ch++;
                  hstr = (unsigned)str<32 ? '^' : str;
                  printf("%c ", hstr);
                  val = str & 0x00ff;
                  printf("%3d ", val);
                  printf("%3o ", val);
                  printf("%2x   ", val);
                  i += 1;
            }
            printf("\n");
      } while (i < e);
      printf("%d\n",i);
}
```

Opgave 1

Schrijf een C-programma dat het kleinste gehele getal n berekent waarvoor geldt:

$$1 + 1/2 + 1/3 + 1/4 + ... + 1/n >= M,\ \ M \text{ is variabel}$$

Het programma moet de maximale waarde van M (Mmax) inlezen en daarna de berekening uitvoeren voor M = 1, 2, 3, ..., Mmax.

9 Arrays, strings en wijzers

9.1 Arrays

Arrays zijn samengesteld uit elk willekeurig type. Zo zijn ook arrays met meer dimensies, dus arrays van arrays, mogelijk.

declaratie: type_id Arrayid[Aantal];

Wordt een array gedeclareerd, dan wordt voor Aantal elementen geheugenruimte gereserveerd. De array-index loopt van 0 tot en met Aantal-1.
De elementen kunnen gerefereerd worden met Arrayid[Nr]. Wordt Arrayid zonder index gebruikt, dan functioneert dit als wijzer naar het eerste element van het array (Arrayid[0]).

Voorbeelden:
```
int Myarray[25]; /* serie van 25 integers */
char Str[256];          /* een string van maximaal 255 karakters */
int Tabel[3][5];  /* twee-dimensionaal array met integers */
```

Het laatste voorbeeld is de declaratie van een twee-dimensionaal array. Dit moet geinterpreteerd worden als 3 arrays van 5 elementen. De eerste index geeft het aantal rijen weer en de tweede index het aantal kolommen als bij een twee-dimensionale matrix. De elementen komen rij-gewijs in het geheugen.
De eerste index loopt van 0 t/m 2, de tweede van 0 t/m 4.

Voorbeeld: stel int Tabel[3][5] is gevuld als volgt

```
0    1    2    3    4
10   11   12   13   14
20   21   22   23   24
```

Het element waarin 13, is Tabel[1][3].
De volgorde waarin de elementen in het geheugen staan:

```
 0  1  2  3  4 10 11 12 13 14 20 21 22 23 24
```

Een string wordt gedeclareerd als een array van char. Hierbij moet echter rekening gehouden worden met het feit dat de geheugenplaats volgend op het laatste karakter gevuld moet zijn (of kunnen worden) met het NULL-karakter (0x00). Daarom is er in de bij de voorbeelden vermelde string Str ruimte voor ten hoogste 255 karakters.

9.2 Wijzers

C bevat uitgebreide mogelijkheden voor manipulatie met wijzers. De declaratie van een wijzer komt tot stand door expliciet een variabele als wijzer naar een bepaald type te declareren. Dit gebeurt als bij declaratie voor de variabele een * geplaatst wordt.

```
declaratie:   type_id *P;     /* P is eenwijzer naar variabele van het type type_id */
voorbeeld:    int *Pint;      /* Pint is een wijzer naar een integer */
```

In het voorbeeld wordt Pint gedeclareerd als wijzer naar een integer. Let op dat de integer zelf hiermee niet gedeclareerd is.
Door variabelen en wijzers te voorzien van de voorvoegsels
* en/of & ontstaan nog meer mogelijkheden

Voorbeeld: als Pint is een wijzer naar een integer,
dan is *Pint de integer zelf en &*Pint is Pint;

als INTid is een integer,
dan is &INTid een wijzer naar INTid
en *&INTid is INTid.

De naam van een 1-dimensionaal array (zonder indexhaken) en van een functie (zonder haken) kan als wijzer gebruikt worden. Letop dat er dan voor zo'n wijzer geen geheugen gealloceerd wordt zodat hij slechts in rechter leden van toewijzingen mag voorkomen.
Voorbeeld (functie-wijzer):

```
int f()
{ ...
}
int (*p_f)();
p_f = f;
(*p_f)(); /* de functie f wordt aangeroepen */
```

Als de naam van een 2-dimensionaal array gevolgd wordt door één index, dan wordt dat opgevat als een wijzer naar het eerste element van de betrokken rij.

Voorbeeld:
Als gedeclareerd is int Tabel[3][5] , dan wordt Tabel[2] opgevat als wijzer naar element Tabel[2][0]. (zie eerder voorbeeld.)

N.B.: De naam van een meer-dimensionaal array zonder indices heeft geen betekenis.

9.3 Wijzer-aritmetiek

Als met wijzers bewerkingen worden uitgevoerd, dan geschiedt dat volgens de zg wijzer-aritmetiek. Dat betekent dat als bij een wijzer een integer opgeteld wordt, dan wordt in werkelijkheid bij deze wijzer opgeteld: integer * lengte_van_wijzer_type.

Voorbeeld: Stel Pint is een wijzer naar een integer en integers beslaan twee bytes in het geheugen. Het statement Pint = Pint + 1; verhoogt dan Pint met 2.

10 Voorbeeld en opgave

Voorbeeld 2

Onderstaande programma's lezen 20 getallen in en bepalen of het 20e getal ook voorkomt in de rij voorafgaande 19 getallen.

```
/* versie 1 arraynotatie */

void main()
{ int a[20], i, laatste;
  printf("Geef 20 gehele getallen: \n");
  for (i = 0; i < 20; i++) scanf("%d", &a[i]);
  laatste = a[19];
  for (i = 0; a[i] != laatste; i++);
  printf(i == 19 ? "Nieuw\n" : "Komt voor\n");
}

/* versie 2 met wijzers */

void main()
{ int a[20], *p, laatste;
  printf("Geef 20 gehele getallen: \n");
  for (p = a; p < a+20; p++) scanf("%d", p);
  laatste = *(a+19);
  for (p = a; *p != laatste; p++);
  printf(p == a+19 ? "Nieuw\n" : "Komt voor\n");
}
```

Opgave 2

Schrijf een programma dat een regel tekst inleest bestaande uit woorden gescheiden door spaties. Druk deze regel af waarbij per woord de lettervolgorde omgekeerd is.

Vb: invoer: Dit is een tekst
uitvoer: tiD si nee tsket

11 Samengestelde typen

11.1 Structure

Een structure is een samengesteld type (vergelijkbaar met het record in Pascal). Een structure kan alle soorten gegevens bevatten, zelfs een andere structure of een array. De opmaak van een structure wordt vastgelegd volgens de volgende syntax:

```
struct Ref {
        declaratie van velden
};
```

waarin struct een sleutelwoord is, Ref een optionele referentie waarnaar later verwijzingen mogelijk zijn, en veld-declaraties gewone declaraties van variabelen zijn.
Een structure van bovenstaande vorm kan gedeclareerd worden volgens de volgende syntax:

```
struct Ref NAAM;
```

waarin NAAM de naam is van de variabele die gedeclareerd wordt.

Beschouw bijvoorbeeld de declaraties die nodig om gegevens bij te kunnen houden van de frequentie van woorden die in een tekst voorkomen. Een implementatie zou kunnen zijn het bewaren van het betrokken woord en het aantal malen dat dat woord voorkomt.
Het statement

```
struct Woordteller {
        char Woord[WOORDLENGTE];
        int Frequentie;
};
```

legt vast een enkelvoudige structure waarin twee velden.

De declaratie

```
struct Woordteller Woordfreq[AANTALWOORDEN];
```

declareert dan een array van zulke structures.
Het is ook mogelijk structure-variabelen te declareren in hetzelfde statement als waarin de opmaak van de structure vastgelegd wordt. De twee vorige statements kunnen bijvoorbeeld gecombineerd worden tot

```
struct Woordteller {
        char Woord[WOORDLENGTE];
        int Frequentie;
} Woordfreq[AANTALWOORDEN];
```

Als in het vervolg van het programma geen gebruik meer gemaakt wordt van de structure, kan de referentie weg gelaten worden. Dit gebeurt veel bij geneste structures.

Bijvoorbeeld:

```
struct Werknemerdata {
        struct {
                char Straat[16];
                char Plaats[8];
        } Adres;
        struct {
                int Salaris;
                int Dienstjaren;
        } Divers;
};
```

Een structure wordt meestal gemanipuleerd via zijn velden. Dit gebeurt door namen te met behulp van de '.' (punt), zoals in

```
Varnaam.Veld
```

Dit toepassend op structure Woordfreq, wordt gerefereerd naar de frequentie van het eerste woord door

```
Woordfreq[0].Frequentie
```

Het eerste karakter van het eerste woord wordt gerefereerd door

```
Woordfreq[0].Woord[0]
```

Als gedeclareerd wordt

```
struct Werknemerdata D;
```

dan wordt naar het salaris gerefereerd door

```
D.Divers.Salaris
```

Geformatteerde in- en uitvoer van een complete structure is niet mogelijk. Dit kan slechts met individuele velden. Bijvoorbeeld:

```
scanf("%s %d", Woordfreq[0].Woord, &Woordfreq[0].Frequentie);
```

Wel is het in moderne C-implementaties mogelijk een gehele structure te kopieëren van de ene naar de andere variabele, zoals in

```
Woordfreq[1] = Woordfreq[0];
```

Een andere manier om velden te benaderen is met behulp van een wijzer. Als voorbeeld dient de structure Woordteller.

```
struct Woordteller *P_Woordteller;
int i;

P_Woordteller = &Woordfreq[1];
i = P_Woordteller->Frequentie;
```

De frequentie van het tweede woord gaat naar variabele i.

De syntax is dus: wijzer_naar_structure->veldnaam

De wijzer-aritmetiek is ook voor structures geldig.
Bijvoorbeeld:

```
P_Woordteller = &Woordfreq[0];
P_Woordteller = P_Woordteller + 1;
```

P_Woordteller wijst nu naar de structure van het tweede woord.

Nog een voorbeeld:

```
#include <stdio.h>
struct boek { char titel[20], schrijver[20]; int blz;};
typedef struct boek BOEK;

void main()
{ BOEK a, b, *p;
  p = &b;
  scanf("%s", &b.titel[0]);
  scanf("%s", p->schrijver);
  scanf("%d", &b.blz);
  a = b; p = &a;
  printf("%s\n%s\n%d\n", p->titel, a.schrijver, p->blz);
}
```

11.2 Union

Bij de structure wordt voor elk veld geheugen gereserveerd, bij de union wordt slechts ruimte gereserveerd voor het langste (in bytes) veld zodat elke referentie naar een veld van een union dezelfde (fysieke) geheugenplaats benaderd. Het woord struct moet vervangen worden door union.

Een voorbeeld van het gebruik van structure en union om de referentie van de 8088-cpu te vergemakkelijken:

```
struct   WORDREGS    {
         unsigned int ax, bx, cx, dx, si, di, cflag, flags;
};

struct   BYTEREGS    {
         unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};

union    REGS {
         struct   WORDREGS x;
         struct   BYTEREGS h;
};

union    REGS r;

r.x.cx   = 10;
r.h.al   = 0X01;
```

12 Voorbeeld en opgave

Voorbeeld 3

```
/* LINKEDLIST: An ordered linked list is built.
               Additions, deletions and changes
               can be given in any order */

void main()
{ struct node { int num; float x,y; struct node *next; };
  typedef struct node NODE;
  NODE *start, *p, *q, *sentinel;
  char *malloc();
  int k = sizeof(NODE), n, nabs;
  float x, y;

  start = sentinel = (NODE*)malloc(k);
  printf("Enter triples n x y to add or change, \n");
  printf("or -n to delete items in the list. \n");
  printf("Enter # to signal end of input. \n\n");

  while (scanf("%d", &n))
  { nabs = abs(n);
    sentinel->num = nabs;
    p = start;
    while (p->num < nabs) p = p->next;
    if (n >= 0)
    { scanf("%f %f", &x, &y);
      if (p == sentinel) sentinel = p->next = (NODE*)malloc(k);
      if (p->num == n) { p->x = x; p->y = y;}
      else
      { q = (NODE*)malloc(k); *q = *p;
        p->num = n; p->x = x; p->y = y; p->next = q;
      }
    } else
    { if (p != sentinel)
      { q = p->next; *p = *q;
        if (sentinel == q) sentinel = p;
        free(q);
      }
    }
  }
  printf("\nResult: \n");
  for ( p = start; p != sentinel; p = p->next )
    printf("%6d %5.1f %5.1f\n", p->num, p->x, p->y);
}
```

Opgave 3

Documenteer bovenstaand programma.

13 Programma-structuur

Een C-programma is een verzameling functies.

Een functie is van een zeker type (als weggelaten: int). De argumenten zijn ook van een zeker type.

Belangrijk is te beseffen dat een C-compiler de invoer (een programma dus) volstrekt sequentiëel verwerkt. Dat betekent bv dat, als de compiler de aanroep van een functie tegenkomt die niet eerder gedeclareerd of geprototypeerd is, de standaard-typeringen gebruikt zullen worden bij de compilatie van de aanroep. Blijkt later dat de functie niet met het standaard-type gedeclareerd is, dan kunnen de eerdere (foute) compilaties niet meer gecorrigeerd worden. Daarom verdient het aanbeveling alle aan te roepen functies aan het begin van het programma te vermelden. Dit wordt "prototypering" genoemd. Zie onderstaand voorbeeld.

VB:
```
double invert(int); /* prototypering */

void main()
{
  printf("%f\n", invert(3));
}

double invert(int x) /* functie declaratie */
{
  return 1.0/x;
}
```

Het is aanbevelenswaardig functies die geen waarde terug geven, het type 'void' mee te geven.

VB:
```
void f(int x, int y)
{
  printf( x < y ? "x" : "y" );
}
```

Parameter-overdracht:

altijd 'by value';
mbv wijzers 'by reference' mogelijk.

VB:
```
void wissel(int *, int *);  /* prototypering */

void main()
{ int i =10, j =20;
  wissel( &i, &j);
  printf("%2d %2d\n", i, j);
}

void wissel(int *px, int *py) /* declaratie */
{ int hulp;
  hulp = *px; *px = *py; *py = hulp;
}
```

NB: Let op bij array en string parameters.

14 Variabelen: declaratie en prototypering

In C onderscheiden we de begrippen '**declaratie**' en '**prototypering**'. Bij de prototypering van een variabele worden slechts de kenmerken van die variabele vastgelegd (ten behoeve van de compiler). Als een variabele gedeclareerd wordt, dan wordt, al of niet gebruik makend van een eerdere prototypering, vastgelegd waar en wanneer geheugen gealloceerd wordt en hoe dit geheugen geinitialiseerd wordt. Daarom kan een variabele slechts éénmaal gedeclareerd worden en zoveel als nodig geprototypeerd worden.

dus: declaratie = [prototypering +] geheugenallocatie + initialisatie

Er bestaan twee soorten variabelen: globale en locale.
In C heten deze resp. '**external**' en '**automatic**'.
Het wezenlijke verschil zit in het feit wanneer en waar geheugen voor de variabele gealloceerd wordt: voor external variabelen wordt geheugen gealloceerd tijdens compilatie en op de 'heap' zodat ze permanent aanwezig zijn, voor automatic variabelen wordt geheugen gealloceerd tijdens executie en op de 'stack' zodat ze slechts aanwezig zijn gedurende de executie in het betrokken blok.

- automatic (vergelijk Pascal lokale variabelen)

declaratie: aan het begin van een blok

bereik: het betrokken blok

VB:
```
{ int i;                                  |
        .                                 |
        .                                 |
  { int i;          |
          .                 |
          .                 |
  }
        .                                 |
        .                                 |
}
```

- external (vergelijk Pascal globale variabelen)

declaratie: buiten een blok (gebruikelijk is vooraan het bestand zodat ze in het gehele bestand bekend zijn)

bereik: het betrokken bestand

VB:
```
void f(void); /* prototypering */
int e; /* declaratie */
void main()
{ e = 1; f();
}
void f() /* declaratie */
{ printf("%d\n", e);
}
```

Functies worden 'buiten een blok geprototypeerd of gedeclareerd' zodat hiervoor vergelijkbare regels gelden als voor external variabelen.

External variabelen kunnen ook in andere bestanden dan waarin ze gedeclareerd zijn, gebruikt worden. In die andere bestanden moeten ze dan geprototypeerd worden met het voorvoegsel '**extern**'. Er wordt dan geen geheugen voor gereserveerd maar tijdens het linken van het totale programma worden de adressen ingevuld.

VB: bestand1

```
extern void f(); /* prototypering */
int e; /* declaratie */
void main()
{
  e = 1; f();
}
```

bestand2

```
void int f() /* declaratie */
{ extern int e; /* prototypering */
  printf("%d\n", e);
}
```

Zowel external als automatic variabelen kunnen bij declaratie het voorvoegsel '**static**' krijgen. Bij external variabelen betekent dit dat de betrokken variabele lokaal gemaakt wordt voor het bestand, dus de variabele kan niet extern verklaard worden in andere bestanden. Voor een automatic variabele betekent het dat het benodigd geheugen niet uit de stack komt maar uit de heap zodat de variabele een eenmaal verkregen waarde behoudt.

VB:

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

VB:

```
static void Open(int, int); /* prototypering */
static int E;
void main()
{ ... Open( A, B) ...
}
static void Open(int x, int y) /* declaratie */
{ ...
}
```

Bij de declaratie van een automatic variabele van het type int, short of char kan het voorvoegsel '**register**' gebruikt worden. De compiler probeert dan de betrokken variabele zoveel mogelijk in een register te bewaren.

15 Voorbeeld en opgave

Voorbeeld 4

```
/* BINTREE: This program builds a binary search tree
            and prints its contents in ascending order. */
#define NULL 0
struct node { int num; struct node *left, *right; };
typedef struct node NODE;
NODE *build(NODE *, int);
void printtree(Node *);
NODE *found(NODE *, int);

void main()
{ int i;
  NODE *root, *ptr;
  root = NULL;
  printf("Enter nonzero integers, followed by 0: \n");
  while (scanf("%d", &i), i) root = build(root,i);
  printf("\nIn ascending order we have:\n");
  printtree(root);
  printf("\nEnter a number to be searched for: ");
  scanf("%d", &i);
  if (ptr = found(root,i), ptr == NULL)
     printf("Not stored. \n"); else printf("Found!\n");
}
NODE *build(NODE *p,int i)
{ char *malloc();
  if (p == NULL)
  { p = (NODE*)malloc(sizeof(NODE));
    p->num = i;
    p->left = p->right = NULL;
  }
  else
    if (i < p->num) p->left = build(p->left,i);
    else p->right = build(p->right,i);
  return p;
}
void printtree(NODE *p)
{ if (p != NULL)
  { printtree(p->left);
    printf("%9d\n", p->num);
    printtree(p->right);
  }
}
NODE *found(NODE *p,int i)
{ return p == NULL || i == p->num ? p :
             found((i < p->num ? p->left : p->right),i);
}
```

Opgave 4

Documenteer bovenstaand programma.

16 C-preprocessor

Elke C-compiler kent een eerste doorgang waarin compiler directieven verwerkt worden. Een directief heeft als eerste teken '#'. Hieronder staan een aantal mogelijke directieven:

DIRECTIE	BETEKENIS
#define A B	vervang identifier A door B
#undef A	hef betrokken define op
#include<filename>	substitueer betrokken include-file
#if constant-expression ... #else ... #endif	voorwaardelijke compilatie
#ifdef Id	als Id gedefiniëerd is, dan ..
#ifndef Id	als Id niet gedefiniëerd is, dan ..

Voorbeeld 5:

includebestand A.H

```
#define N 1000
```

bronbestand B.C

```
#include <A.H>
main()
{ #ifndef N
   printf("Geen define voor N\n");
  #else
        #if N <= 100
         float Matrix[N][N]
              .
              .
              .
        #else
         printf("Matrix te groot\n");
        #endif
  #endif
}
```

Opgave 5:

Bepaal het C-programma als include-file A.H bevat

1/ #define N 1000
2/ #define n 1000
3/ #define N 60

17 Een sorteer-programma in Pascal en in C

```
program Dosort;
const
  LMax = 100;
type
  Item = integer;
  List = array[1..LMax] of Item;
var
  myList    : List;
  Count, I  : Integer;
  Ch                  : Char;

procedure Sortlist(var L : List; C : Integer);
var
  Top, Min, K : Integer;


procedure SwapItem(var I, J : Item);
var  Temp : Item;
begin
  Temp := I; I := J; J := Temp;
end; { of proc SwapItem }

begin { mainbody of SortList }
  for Top := 1 to C-1 do
  begin
    Min := Top;
    for K := Top + 1 to C do
      if L[K] < L[Min] then Min := K;
    SwapItem(L[Top], L[Min]);
  end
end; { of proc SortList }


procedure DumpList(L : List; C : Integer);
var I : Integer;
begin
  for I := 1 to C do
    Writeln('L[',I:3,'] = ',L[I]:4)
end; { end of proc DumpList }


begin { mainbody of DoSort }
  for I := 1 to LMax do
    myList[I] := Random(1000);
  Count := LMax;
  DumpList(myList, Count);
  Read(Kbd, Ch);
  SortList(myList, Count);
  DumpList(myList, Count);
  Read(Kbd, Ch);
end. { of DoSort }
```

```
#define LMax 100

typedef int Item;
typedef Item List[LMax];

List myList;
int Count, I;
char Ch;

void SwapItem(Item*,Item*);
void SortList(List,int);
void DumpList(List,int);


void SwapItem(Item *I, Item *J)

{ Item Temp;
  Temp = *I; *I = *J; *J = Temp;
} /* SwapItem */

void SortList(List L, int C)
{ int Top, Min, K;
  for (Top = 0; Top < C-1; Top++)
  {
    Min = Top;
    for (K = Top + 1; K <= C; K++)
      if (L[K] < L[Min]) Min = K;
    SwapItem(&L[Top], &L[Min]);
  }
} /* SortList */

void DumpList(List L, int C)

{ int I;
  for ( I = 0; I <= C; I++)
    printf("L[%3d] = %4d\n",I,L[I]);
} /* DumpList */


void main()
{ for (I = 0; I < LMax; I++)
    myList[I] = rand() % 1000;
  Count = LMax;
  DumpList(myList, Count);
  Ch = getch();
  SortList(myList, Count);
  DumpList(myList, Count);
  Ch = getch();
} /* main */
```

Opgave

De voorgaande programmalistings komen uit de Turbo-C (1) User's Guide.
De C versie van het sorteerprogramma werkt niet goed.
Verbeter deze.

18 Literatuur

Het 'standaardboek' mbt de taal C is:

Titel C handboek (NB vertaling van onderstaand boek)
Auteurs Kernighan & Ritchie
Uitgever P-H/Academic Service ISBN 90 6233 488 1

Titel The C Programming Language (second edition)
Auteurs Kernighan & Ritchie
Uitgever Prentice-Hall ISBN 13-110362-8

Titel The C Answer Book (second edition)
Auteurs Tondo & Gimpel
Uitgever Prentice-Hall ISBN 13-109653-2

Titel Basishandleiding (beperkt maar goedkoop en handig)