Sessie 6: Functie III (Records)

Wat is de vertaling van de volgende C-code?

```
struct rationaalgetal
{
  int teller;
  int noemer;
  int gehdeel;
};

struct rationaalgetal maakrationaal (register int gehdeel, int teller, int
noemer)
{
   struct rationaalgetal g;

   g.gehdeel = gehdeel;
   g.teller = teller;
   g.noemer = noemer;

   return (g);
}

struct rationaalgetal kopieerrationaal (register struct rationaal * g)
{
   return (*g);
}

void drukrationaal (struct rationaalgetal g) { ... } // zie vorige huistaak

struct rationaalgetal gq;

main ()
{
   struct rationaalgetal q;

   q = maakrationaal(20, 3, 10);
   gq = kopieerrationaal(&q);

   drukrationaal(gq);  // zou 20 3 10 moeten afdrukken
}
```

Oplossing:

```
ARs en toewijzingstabellen

AR van maakrationaal:         In registers:
  +----------------+
  | var. g.teller  |            Parameters: gehdeel
  +----------------+            Variabelen: ---
  | var. g.noemer  |
  +----------------+
  | var. g.gehdeel |
  +----------------+
  |   TKA          |
  +----------------+
  | oude R8        |  <-- R8
  +----------------+
  | par. teller    |
  +----------------+
```

```
  | par. noemer    |
  +----------------+
  | RES.teller     |
  +----------------+
  | RES.noemer     |
  +----------------+
  | RES.gehdeel    |
  +----------------+

 Toewijzingstabel (maakrationaal)
  Parameter |  Adres
  ----------+-------
   gehdeel  |  R1
  ----------+-------
   teller   |  1(R8)
  ----------+-------
   noemer   |  2(R8)
  ----------+-------
  Variabele |  Adres
  ----------+-------
   g        |  -4(R8)
  ----------+-------
  Resultaat |  Adres
  ----------+-------
  RES       |  3(R8)

AR van kopieerrationaal:         In registers:
  +----------------+              Parameters: g
  |   TKA          |              Variabelen: ---
  +----------------+
  | oude R8        |  <-- R8
  +----------------+
  | RES.teller     |
  +----------------+
  | RES.noemer     |
  +----------------+
  | RES.gehdeel    |
  +----------------+

 Toewijzingstabel (kopieerrationaal)
  Parameter |  Adres
  ----------+-------
     g      |  R1
  ----------+-------
  Variabele |  Adres
  ----------+-------
     ---    |  ---
  ----------+-------
  Resultaat |  Adres
  ----------+-------
  RES       |  1(R8)

AR van drukrationaal:         In registers:
  +----------------+              Parameters: ---
  |   TKA          |              Variabelen: ---
  +----------------+
  | oude R8        |  <-- R8
  +----------------+
  | par. g.teller  |
  +----------------+
  | par. g.noemer  |
```

```
  +----------------+
  | par. g.gehdeel |
  +----------------+

 Toewijzingstabel (drukrationaal)
  Parameter |  Adres
  ----------+-------
     g      |  1(R8)
  ----------+-------
  Variabele |  Adres
  ----------+-------
   ---      |  ---

AR en toewijzingstabel van main
AR van main:              In registers: ---

  +--------------+
  | q.teller     |
  +--------------+
  | q.noemer     |
  +--------------+
  | q.gehdeel    |
  +--------------+
  |   TKA = -1   |
  +--------------+
  | oude R8 = -1 |  <-- R8
  +--------------+

   Toewijzingstabel (main)

  Variabele |  Adres
  ----------+-------
     q      |  -4(R8)

   Globaal:
  Variabele |  Adres
  ----------+--------
     gq     |  gq


main:    HIA.w R1,-1
         BST   R0      | omg(main)
         HIA   R8,R9
         BST   R0      | onecht TKA

         AFT.w R9,3    | plaats voor var. q

         AFT.w R9,3    | plaats voor resultaat

         HIA.w R1,20     | par. gehdeel == 20

         HIA.w R0,3
         BST   R0        | par. teller == 3
         HIA.w R0,10
         BST   R0        | par. noemer == 10

         BST   R8
         HIA   R8,R9     | omg(maakrationaal)
         SBR   maakrationaal  | maakrationaal(20, 3, 10)
         HST   R8        | herstel omg(main)
```

```
        OPT.w R9,2      | parameters weg

        | resultaat kopieren naar q

        HST   R0
        BIG   R0,-4+0(R8)  | q.teller = RES.teller
        HST   R0
        BIG   R0,-4+1(R8)  | q.noemer = RES.noemer
        HST   R0
        BIG   R0,-4+2(R8)  | q.gehdeel = RES.gehdeel

        AFT.w R9,3    | plaats voor resultaat

        HIA.a R1,-4(R8) | par. g == &q;

        BST   R8
        HIA   R8,R9     | omg(kopieerrationaal)
        SBR   kopieerrationaal  | kopieerrationaal(&q)
        HST   R8        | herstel omg(main)

        OPT.w R9,2      | parameters weg

        | resultaat kopieren naar gq

        HST   R0
        BIG   R0,gq+0      | gq.teller = RES.teller
        HST   R0
        BIG   R0,gq+1      | gq.noemer = RES.noemer
        HST   R0
        BIG   R0,gq+2      | gq.gehdeel = RES.gehdeel

        | parameter g == gq op de stapel (begin met het laatste veld!)
        HIA   R0,gq+2      | g.gehdeel == gq.gehdeel
        BST   R0
        HIA   R0,gq+1      | g.noemer == gq.noemer
        BST   R0
        HIA   R0,gq+0      | g.teller == gq.teller
        BST   R0

        BST   R8
        HIA   R8,R9     | omg(drukrationaal)
        SBR   drukrationaal  | drukrationaal(q)
        HST   R8        | herstel omg(main)

        OPT.w R9,3      | parameter weg
        STP

gq:     RESGR 3         | globale variabele

MAAKRATIONAAL:

        AFT.w R9,3         | plaats voor var. g

        BIG   R1,-4+2(R8)  | g.gehdeel = gehdeel

        HIA   R0,2(R8)
        BIG   R0,-4+0(R8)  | g.teller = teller

        HIA   R0,1(R8)
        BIG   R0,-4+1(R8)  | g.noemer = noemer
```

```
        | return (g)

        HIA   R0,-4+0(R8)
        BIG   R0,3+0(R8)   | RES.teller = g.teller
        HIA   R0,-4+1(R8)
        BIG   R0,3+1(R8)   | RES.noemer = g.noemer
        HIA   R0,-4+2(R8)
        BIG   R0,3+2(R8)   | RES.gehdeel = g.gehdeel

        OPT.w R9,3         | var. g weg

        KTG

KOPIEERRATIONAAL:
        HIA   R0,0+0(R1)
        BIG   R0,-4+0(R8)  | RES.teller = g->teller
        HIA   R0,0+1(R1)
        BIG   R0,-4+1(R8)  | RES.noemer = g->noemer
        HIA   R0,0+2(R1)
        BIG   R0,-4+2(R8)  | RES.gehdeel = g->gehdeel
        KTG

DRUKRATIONAAL:
    ...
```

Sessie7: Functies IV (Activatie Records)

Veronderstel dat onderstaand programma vertaald is en in het geheugen geladen wordt vanaf adres 0.

```
0:        HIA.a R2,0
1:        HIA.a R1,CRIT
2:   LUS: LEZ
3:        VGL.i R0,0(R1+)
4:        VSP   GRG,END
5:        OPT   R2,R0
6:        SPR   LUS
7:   END: HIA   R0,R2
8:        DRU
9:        STP
10: CRIT: 13
11:       14
12:       15
13:       16
14:       17
15:       18
16:       19
17:       20
```

Indien de invoer de getallen 13, 16, 20, 40 en 100 bevat (in die volgorde):

- hoeveel getallen worden er dan ingelezen door dit programma?
- hoeveel instructies zijn er dan uitgevoerd?
- hoe vaak is er hierbij uit het DRAMA-geheugen gelezen? (Let op: het programma zit in het geheugen en elk bevel moet opgehaald worden om het te kunnen uitvoeren; het ophalen van een bevel telt ook als een leesoperatie!)

- welke waarde wordt er afgedrukt?

Oplossing:

```
                                  Uitv      Lees     R0     R1R2
0:          HIA.a R2,0             |         |                    0
1:          HIA.a R1,CRIT          |         |            10
2:    LUS:  LEZ                    |||       |||     13/16/20
3:          VGL.i R0,0(R1+)        |||       |||           11/12/13
4:          VSP   GRG,END          |||       |||
5:          OPT   R2,R0            ||        ||                   13/29
6:          SPR   LUS              ||        ||
7:    END:  HIA   R0,R2            |         |
8:          DRU                    |         |
9:          STP                    |         |
10: CRIT:   13                               |
11:         14                               |
12:         15                               |
13:         16                               |
14:         17                               |
15:         18                               |
16:         19
17:         20
```

Antwoord:

- aantal ingelezen getallen = 3
- aantal uitgevoerde instructie = 18
- aantal leesoperaties uit het geheugen = 18+6 = 24
- afgedrukte resultaat = 29

Sessie 8: Lijsten

Wat is de vertaling van het volgende C-programma? Het programma leest een aantal getallen in en bouwt een gesorteerde lijst op (van klein naar groot). De lijst heeft een kopelement.

```
struct elem
{
   int info;
   struct elem * volg;
};

void inlassen (struct elem * l, int g)
{
   register struct elem * p, *pp;
   struct elem * n;

   pp = l;
   p = pp->volg;
   while ((p != NULL) && (p->info < g))
   {
      pp = p;
      p = pp->volg;
   }
   // Het nieuwe element toevoegen tussen pp en p
```

```
   n = alloc(sizeof(struct elem));  // sizeof(...) geeft de grootte van het type
                                    // dit is GEEN functieoproep maar een
opdracht voor de compilator!
   n->info = g;
   n->volg = p;
   pp->volg = n;
}

void druklijst (register struct elem * l)
{
   l = l->volg;  // sla kopelement over
   while (l != NULL) {
     printint(l->info);
     l = l->volg;
   }
}

main ()
{
   register struct elem * lst;
   int getal;

   // maak lege lijst (met kopelement)
   lst = alloc(2);
   lst->info = 0;
   lst->volg = NULL;

   // inlezen van positieve getallen
   getal = getint();
   while (getal > 0)
   {
      inlassen(lst, getal);
      getal = getint();
   }
   druklijst(lst);
}
```

Oplossing:

```
AR van inlassen:
  +-----------+
  |    n      |               In registers:
  +-----------+                 Parameters: ---
  |   TKA     |                 Variabelen: p, pp
  +-----------+
  | oude R8   |  <-- R8
  +-----------+
  |    g      |
  +-----------+
  |    l      |
  +-----------+

 Toewijzingstabel (inlassen)
  Parameter |  Adres
  ----------+-------
     l      |  2(R8)
     g      |  1(R8)
  ----------+-------
  Variabele |  Adres
  ----------+-------
```

```
     p      |  R6
     pp     |  R5
     n      |  -2(R8)

AR van druklijst:
                              In registers:
  +-----------+                  Parameters: l
  |   TKA     |                  Variabelen: ---
  +-----------+
  | oude R8   |  <-- R8
  +-----------+

 Toewijzingstabel (druklijst)
  Parameter |  Adres
  ----------+-------
     l      |  R1
  ----------+-------
  Variabele |  Adres
  ----------+-------
     ---    |  ---

AR van main:              In registers: lst
  +--------------+
  |  getal       |
  +--------------+
  |   TKA = -1   |
  +--------------+
  | oude R8 = -1 |  <-- R8
  +--------------+

   Toewijzingstabel (main)

  Variabele |  Adres
  ----------+-------
     lst    |  R6
     getal  |  -2(R8)
```

Vertaling:

```
              | register struct elem * p, *pp;
              | struct elem * n;
    inlassen: AFT.w R9,1  | plaats voor R9

              BST   R6
              BST   R5    | R5, R6 bewaren (p, pp in reg)

              HIA   R5,2(R8)  | pp = l
              HIA   R6,1(R5)  | p = pp->volg;

       while: VGL.w R6,-1     | p != NULL ?
              VSP   GEL,ewh

              HIA   R0,0(R6)  | p->info
              VGL   R0,1(R8)  | p->info < g ?
              VSP   GRG,ewh

              HIA   R5,R6     | pp = p
              HIA   R6,1(R5)  | p = pp->volg

              SPR while
```

```
          | Het nieuwe element toevoegen tussen pp en p

    ewh:  BIG   R7,-2(R8)  | n = alloc(2)
          OPT.w R7,2

          HIA   R0,1(R8)
          BIG   R0,-2+0(R7)  | n->info = g

          BIG   R6,-2+1(R7)  | n->volg = p

          HIA   R0,-2(R8)
          BIG   R0,1(R5)     | pp->volg = n

          HST   R5
          HST   R6   | R5 en R6 herstellen

          OPT.w R9,1 | lokale var n v/d stapel halen

          KTG

druklst:  HIA   R1,1(R1)  | l = l->volg

    lus:  VGL.w R1,-1
          VSP   GEL,elus  | l != NULL ?

          HIA   R0,0(R1)
          DRU             | printint(l->info)

          HIA   R1,1(R1)  | l = l->volg

          SPR   lus

   elus:  KTG

main ()
{

main:   HIA.w R0,-1
        BST   R0
        HIA   R8,R9
        BST   R0

        HIA.a R7,HEAP

        | register struct elem * lst;
        | int getal;

        AFT.w R9,1  | plaats voor getal

        | maak lege lijst (met kopelement)
        HIA   R6,R7
        OPT.w R7,2   | lst = alloc(2);

        HIA.w R0,0
        BIG   R0,0(R6)  | lst->info = 0

        HIA.w R0,-1
        BIG   R0,1(R6)  | lst->volg = NULL

        | inlezen van positieve getallen
```

```
            LEZ
            BIG   R0,-2(R8)  | getal = getint()

     wlus:  VSP   NPOS,ewlus | getal > 0 ?

            BST   R6   | param l = lst
            BST   R0   | param g = getal

            BST   R8
            HIA   R8,R9
            SBR   inlassen
            HST   R8

            OPT.w R9,2  | param van stapel halen

            LEZ
            BIG   R0,-2(R8)  | getal = getint()

            SPR   wlus

    ewlus:  HIA   R1,R6   | param l = lst

            BST   R8
            HIA   R8,R9
            SBR   druklst
            HST   R8

            STP

    HEAP:   RESGR 100
```

Sessie 10: Macro's

Schrijf een macro NIEUW dat een nieuw element (struct verpl) alloceert op de HEAP en dit gepast initiliseert.

```
    | struct verpl {
    |   int van;
    |   int naar;
    |   int aantal;
    |   struct verpl * volg;
    | };
    | De macro nieuw alloceert een record van het type struct verpl op de HEAP
    | en initialiseert de velden met de meegegeven parameters
    | De parameters van, naar en aantal zijn *geheugenadressen* van de velden
waar
    |   deze waarden gevonden kunnen worden
    | De parameter Lijst is een *register* dat het adres van een lijst bevat
    | Het gealloceerde record wordt vooraan in de lijst geplaatst!
    | De parameter Reg geeft aan welk *register* mag gebruikt worden

    MACRO
       NIEUW van, naar, aantal, Lijst, Reg

       ...

    MCREINDE
```

Gebruik de macro in het vertaalde C-programma (torens van Hanoi van enkele weken geleden) waarbij de oproep naar de functie nieuw vervangen wordt door een oproep van de macro NIEUW. Welke versie van het programma zal sneller uitvoeren? Waarom? Vergelijk ook de lengte van de vertaalde programma's. Welk is langer? Verklaar!

```
    struct verpl {
      int van;
      int naar;
      int aantal;
      struct verpl * volg;
    };

    // De geschreven macro vervangt de volgende functie
    struct verpl * nieuw (int van, int naar, int aantal, register struct
verplaatsing * lijst)
    {
       register struct verpl * v = (struct verpl *) alloc(sizeof(struct verpl));
       v->van = van;
       v->naar = naar;
       v->aantal = aantal;
       v->volg = lijst;    // plaats het element vooraan in de lijst
       return (v);
    }

    main ()
    {
       int aantal;
       register struct verpl * lijst;
       register struct verpl * huidig;
       int via, aVerpl;

       aantal = getint();

       // verplaats AANTAL ringen van 1 naar 2
       // -----------------------------------
       // *Hier wordt de macro NIEUW gebruikt*
       lijst = nieuw(1, 2, aantal, NULL);
       aVerpl = 0;

       while (lijst != NULL)
       {
          huidig = lijst;
          lijst = lijst->volg;  // haal element uit de lijst

         if (huidig->aantal == 1) {
             printint(huidig->van, huidig->naar);
             aVerpl++;
          } else {
             via = 6 - huidig->van - huidig->naar;
             /* genereer in omgekeerde volgorde en voeg vooraan toe aan lijst */
             // *Hier zal telkens de macro gebruikt worden*
             lijst = nieuw (via, huidig->naar, huidig->aantal - 1, lijst);
             lijst = nieuw (huidig->van, huidig->naar, 1, lijst);
             lijst = nieuw (huidig->van, via, huidig->aantal - 1, lijst);
          }
       }
       printint(aVerpl);
    }
```

Oplossing

```
| struct verpl {
|   int van;
|   int naar;
|   int aantal;
|   struct verpl * volg;
| };
| De macro nieuw alloceert een record van het type struct verpl op de HEAP
| en initialiseert de velden met de meegegeven parameters
| De parameters van, naar en aantal zijn GEHEUGENADRESSEN van de velden waar
|   deze waarden gevonden kunnen worden
| De parameter Lijst is een register dat het adres van een lijst bevat
| Het gealloceerde record wordt vooraan in de lijst geplaatst!
| De parameter Reg geeft aan welk register mag gebruikt worden

MACRO
   NIEUW van, naar, aantal, Lijst, Reg
   | Bewaar eerst de inhoud van Lijst, anders ben je die kwijt!
   HIA   <Reg>,<Lijst>

   HIA   <Lijst>,R7
   OPT.w R7,4         | alloceer record op de HEAP!

   | Initialiseer de velden t.o.v. -4(R7)
   BIG   <Reg>,-4+3(R7) | v->volg = Lijst

   HIA   <Reg>,<van>
   BIG   <Reg>,-4+0(R7)    | v->van = van

   HIA   <Reg>,<naar>
   BIG   <Reg>,-4+1(R7)    | v->naar = naar

   HIA   <Reg>,<aantal>
   BIG   <Reg>,-4+2(R7)    | v->aantal = aantal
MCREINDE
```

De oproep van nieuw in het hoofdprogramma:

```
Toewijzingstabel (main)
Variabele | Adres
----------+---------
aantal    | -2(R8)
lijst     | R6
huidig    | R5
via       | -3(R8)
aVerpl    | -4(R8)

   | lijst = nieuw(1, 2, aantal, NULL);

   HIA.w R6,-1   | lijst = NULL
   | 1 en 2 moeten in het geheugen zitten (gebruik hiervoor de stapel)
   HIA.w R0,1
   BST   R0
   HIA.w R0,2
   BST   R0

   NIEUW 1(R9),0(R9),-2(R8),R6,R0
   OPT.w R9,2  | verwijder 1 en 2 van de stapel

   | aVerpl = 0;
   ...
```

```
   | while (lijst != NULL) {
   ...

   |  huidig = lijst;
   |  lijst = lijst->volg;  // haal element uit de lijst
   ...

   | if (huidig->aantal == 1) {
   |     printint(huidig->van, huidig->naar);
   |     aVerpl++;
   ...
   |   } else {
   |     via = 6 - huidig->van - huidig->naar;
   ...

   |     lijst = nieuw (via, huidig->naar, huidig->aantal - 1, lijst);
   HIA   R0,2(R5)
   AFT.w R0,1
   BST   R0      | huidig->aantal - 1 op de stapel

   NIEUW -3(R8),1(R5),0(R9),R6,R0
   | laat huidig->aantal - 1 nog even op de stapel staan, straks terug nodig!

   |     lijst = nieuw (huidig->van, huidig->naar, 1, lijst);

   HIA.w R0,1
   BST   R0    | 1 op de stapel

   NIEUW 0(R5),1(R5),0(R9),R6,R0
   OPT.w R9,1  | 1 van de stapel

   |     lijst = nieuw (huidig->van, via, huidig->aantal - 1, lijst);
   NIEUW 0(R5),-3(R8),0(R9),R6,R0
   OPT.w R9,1  | huidig->aantal - 1 van de stapel halen

   ...
```

Sessie 11: Herhaling

Vertaal het volgende C programma:

```
struct bkg {
  int teken; // 0 = +, 1 = -
  int mantisse; // decimale punt staat na het eerste cijfer (eerste cijfer is
steeds 0)
  int exponent; // bereik; [-500,499], in +500 notatie; basis is 10
};

#define ATEST 3
#define AMETING 5
#define HONDERDMILJOEN  100000000
#define MILJARD         1000000000
#define VERHOOG  500

// meting: 3 testen, elk van 5 metingen

struct bkg meting[ATEST][AMETING];

void lezMeting (struct bkg * g)
```

```
{
   register int m, e;

   g->teken = getint();
   m = getint(); // MAXIMAAL 9 cijfers,  "." staat net voor het eerste cijfer
   while (m < HONDERDMILJOEN)
     m *= 10;
   g->mantisse = m;
   e = getint();  // in domein [-500 .. +499]
   e += VERHOOG;
   g->exponent = e;
}

struct bkg bwk_som (struct bkg * g1, register struct bkg * g2)
{
   struct bkg tmp;

   tmp.teken = g1->teken;
   if (g1->teken == g2->teken)
      bwk_opt (&tmp, g1, g2);
   else
      bwk_aft (&tmp, g1, g2);

   normaliseer(&tmp);
   return tmp;
}

int aligneer (int exp1, register int * m1, int exp2, register int * m2)
{
   int aantPos;

   // maak kleinste exponent gelijk aan grootste en pas de mantisse aan
   // indien aantPos > 0: verschuif *m2 naar rechts, anders verschuif *m1 naar
rechts
   // geef als resultaat de grootste exponent terug

   aantPos = exp1 - exp2;

   if (aantPos > 0) {
     while (aantPos--)
       *m2 /= 10;
     return exp1;
   }
   else if (aantPos < 0) {
     while (aantPos++)
       *m1 /= 10;
     return exp2;
   }
   else
     return exp1; // reeds gealigneerd
}

void bwk_opt (struct bkg * res, register struct bkg * g1, struct bkg * g2)
{
   int m1, m2;

   m1 = g1->mantisse;
   m2 = g2->mantisse;

   res->exponent = aligneer (g1->exponent, &m1, g2->exponent, &m2);
```

```
   res->mantisse = m1 + m2;
}

void bwk_aft (struct bkg * res, register struct bkg * g1, struct bkg * g2)
{
   int m1, m2;

   m1 = g1->mantisse;
   m2 = g2->mantisse;

   res->exponent = aligneer (g1->exponent, &m1, g2->exponent, &m2);

   if (m1 > m2)
      res->mantisse = m1 - m2;
   else {
      res->mantisse = m2 - m1;
      res->teken = 1 - res->teken; // keer het teken om
   }
}

// normaliseer zorgt ervoor dat ofwel m = 0, ofwel 100.000.000 <= m <
1.000.000.000.000
void normaliseer (struct bkg * g)
{
   register int m, e;

   m = g->mantisse;
   e = g->exponent;

   if (m == 0) {
      g->exponent = VERHOOG;  // maak exponent = 0
      return;
   }
   if (m > MILJARD) { // te groot: deel door 10
      g->mantisse = m / 10;
      g->exponent++
      return;
   }
   while (m < HONDERDMILJOEN) {  // te klein: vermenigvuldig een aantal keer met
10
     m *= 10;
     e--;
   }
   g->exponent = e;
   g->mantisse = m;
}

main()
{
   int i, j;
   struct bkg som;

   for (i = 0; i < ATEST; i++)
    for (j = 0; j < AMETING; j++)
      lezMeting(&meting[i][j]);

   // bereken de som van de metingen voor elke test

   for (i = 0; i < ATEST; i++) {
      som.teken = 0;
      som.mantisse = 0;
```

```
      som.exponent = 500;
      for (j = 0; j < AMETING; j++)
         som = bwk_som (&som, &meting[i][j]);
      printint( som.teken, som.mantisse, som.exponent - VERHOOG);
   }
}
```

Activiatierecords en toewijzingstabellen:

```
struct bkg {
  int teken;    // 1 geheugenplaats
  int mantisse; // 1 geheugenplaats
  int exponent; // 1 geheugenplaats
};

De gegeven struct bkg heeft dus grootte 3.

-------------------------------------------------------------------------------
void lezMeting (struct bkg * g)
{
   register int m, e;
...

stack:
|----------|
|TKA       |
|----------|
|oude R8   | <- R8
|----------|
|g         |
|----------|

toekenningstabel:

g | 1(R8)
----------
m | R6
e | R5


-------------------------------------------------------------------------------
struct bkg bwk_som (struct bkg * g1, register struct bkg * g2)
{
   struct bkg tmp;

|------------------|
|tmp.teken         |
|------------------|
|tmp.mantisse      |
|------------------|
|tmp.exponent      |
|------------------|
|TKA               |
|------------------|
|oude R8           | <- R8
|------------------|
|g1                |
|------------------|
|RESULTAAT.teken   |
|------------------|
|RESULTAAT.mantisse|
```

```
|------------------|
|RESULTAAT.exponent|
|------------------|

g1         | 1(R8)
g2         | R1
-----------------
RESULTAAT | 2(R8)
-----------------
tmp       |-4(R8)


-------------------------------------------------------------------------------
int aligneer (int exp1, register int * m1, int exp2, register int * m2)
{
   int aantPos;

|-------|
|aantpos|
|-------|
|TKA    |
|-------|
|oude R8| <- R8
|-------|
|exp2   |
|-------|
|exp1   |
|-------|

exp1    | 2(R8)
m1      | R1
exp2    | 1(R8)
m2      | R2
-----------
aantpos | -2(R8)


-------------------------------------------------------------------------------
void bwk_opt (struct bkg * res, register struct bkg * g1, struct bkg * g2)
{
   int m1, m2;

|-------|
|m2     |
|-------|
|m1     |
|-------|
|TKA    |
|-------|
|oude R8| <- R8
|-------|
|g2     |
|-------|
|res    |
|-------|

res | 2(R8)
g1  | R1
g2  | 1(R8)
-----------
m1  | -2(R8)
```

```
m2  | -3(R8)

-------------------------------------------------------------------------------
void bwk_aft (struct bkg * res, register struct bkg * g1, struct bkg * g2)
{
   int m1, m2;

zie bwk_opt

-------------------------------------------------------------------------------
void normaliseer (struct bkg * g)
{
   register int m, e;

|-------|
|TKA    |
|-------|
|oude R8| <- R8
|-------|
|g      |
|-------|

g | 1(R8)
---------
m | R6
e | R6

-------------------------------------------------------------------------------
main()
{
   int i, j;
   struct bkg som;

|------------|
|som.teken   |
|------------|
|som.mantisse|
|------------|
|som.exponent|
|------------|
|j           |
|------------|
|i           |
|------------|
|TKA=-1      |
|------------|
|oude R8=-1  | <- R8
|------------|

i   | -2(R8)
j   | -3(R8)
som | -6(R8)
```

Vertaling:

```
| DRAMA computer begint uit te voeren vanaf adres 0x0.
SPR main

| #define ATEST 3
| #define AMETING 5
| #define VERHOOG  500
```

```
MEVA <ATEST>,   3
MEVA <AMETING>, 5
MEVA <VERHOOG>, 500

| Volgende defines zijn te groot om binnen het operand deel van een instructie
| te passen (4 cijfers), dus slaan we ze op in het geheugen.
HONDERDMILJOEN:  100000000
MILJARD:        1000000000

| struct bkg meting[ATEST][AMETING];
| Gegeven struct heeft grootte 3.
meting: RESGR <ATEST>*<AMETING>*3

| void lezMeting (struct bkg * g)
| {
lezMeting:
|    register int m, e;
    BST     R6
    BST     R5
|    g->teken = getint();
    LEZ
    HIA     R1,     1(R8)
    BIG     R0,     0(R1)
|    m = getint();
    LEZ
    HIA     R6,     R0
|    while (m < HONDERDMILJOEN)
lezMeting_while:
    VGL     R6,     HONDERDMILJOEN
    VSP     GRG,    lezMeting_eindwhile
|       m *= 10;
    VER.w   R6,     10
    SPR     lezMeting_while
lezMeting_eindwhile:
|    g->mantisse = m;
    HIA     R1,     1(R8)
    BIG     R6,     1(R1)
|    e = getint();  // in domein [-500 .. +499]
    LEZ
    HIA     R5,     R0
|    e += VERHOOG;
    OPT.w   R5,     <VERHOOG>
|    g->exponent = e;
    HIA     R1,     1(R8)
    BIG     R5,     2(R1)
| }
    HST     R5
    HST     R6
    KTG

| ------------------------------------------------------------------------------
| struct bkg bwk_som (struct bkg * g1, register struct bkg * g2)
| {
bwk_som:
|    struct bkg tmp;
    AFT.w   R9,     3
|   tmp.teken = g1->teken;
    HIA.i   R2,     1(R8)
    BIG     R2,     -4(R8)

|    if (g1->teken == g2->teken)
```

```
    VGL     R2,     0(R1)
    VSP     NGEL,   bwk_som_else
|       bwk_opt (&tmp, g1, g2);

    BST     R1
| parameter res
    HIA.a   R2,     -4(R8)
    BST     R2
| parameter g2 (zit nu nog in R1)
    BST     R1
| parameter g1
    HIA     R1,     1(R8)

    BST     R8
    HIA     R8,     R9
    SBR     bwk_opt
    HST     R8

    OPT.w   R9,2

    HST     R1
    SPR     bwk_som_eindif

bwk_som_else:
|    else
|       bwk_aft (&tmp, g1, g2);
    BST     R1
| parameter res
    HIA.a   R2,     -4(R8)
    BST     R2
| parameter g2 (zit nu nog in R1)
    BST     R1
| parameter g1
    HIA     R1,     1(R8)

    BST     R8
    HIA     R8,     R9
    SBR     bwk_aft
    HST     R8

    OPT.w   R9,2

    HST     R1

bwk_som_eindif:
|    normaliseer(&tmp);
| parameter g
    HIA.a   R0,     -4(R8)
    BST     R0

    BST     R8
    HIA     R8,     R9
    SBR     normaliseer
    HST     R8
    OPT.w   R9,1

|    return tmp;
    HIA     R0,     -4+0(R8)
    BIG     R0,     2+0(R8)
    HIA     R0,     -4+1(R8)
    BIG     R0,     2+1(R8)
```

```
    HIA     R0,     -4+2(R8)
    BIG     R0,     2+2(R8)
| }
    OPT.w   R9,     3
    KTG

| ------------------------------------------------------------------------------
| Hulpmacro om code duplicatie in aligneer functie te minimaliseren.
MACRO
    ALIGN   memAantPos, regPtrMantisse, regRes, memRes, doAdd, lblRet
|     while (aantPos-- OR ++)
$align_wh:
    HIA     <regRes>,   <memAantPos>
    VGL.w   <regRes>,   0
    VSP     NNUL,   $align_nawh

    MVGL    <doAdd>,    0
    MVSP    NGEL, $do_dec
    OPT.w   <regRes>,   1
    MSPR    $do_store
$do_dec: MNTS
    AFT.w   <regRes>,   1
$do_store: MNTS
    BIG     <regRes>,   <memAantPos>

|       *m2 OR 1 /= 10;
    HIA     <regRes>,   0(<regPtrMantisse>)
    DEL.w   <regRes>,   10
    BIG     <regRes>,   0(<regPtrMantisse>)
    SPR     $align_wh

$align_nawh:
|     return exp1 OR exp2;
    HIA     <regRes>, <memRes>
    SPR     <lblRet>
MCREINDE

| ------------------------------------------------------------------------------
| int aligneer (int exp1, register int * m1, int exp2, register int * m2)
| {
|   int aantPos;
    AFT.w   R9,     1

|   aantPos = exp1 - exp2;
    HIA     R0,     2(R8)
    AFT     R0,     1(R8)
    BIG     R0,     -2(R8)

|   if (aantPos > 0) {
    VGL.w   R0,     0
    VSP     KLG,    aligneer_naif
    ALIGN   -2(R8), R2, R0, 2(R8), 0, aligneer_ret

|   else if (aantPos < 0) {
aligneer_naif:
    VSP     NUL,    aligneer_else
    ALIGN   -2(R8), R1, R0, 1(R8), 1, aligneer_ret

|   else
|     return exp1; // reeds gealigneerd
aligneer_else:
```

```
    HIA     R0,     2(R8)

aligneer_ret:
    OPT.w   R9,     1
    KTG
}

| ------------------------------------------------------------------------------
| void bwk_opt (struct bkg * res, register struct bkg * g1, struct bkg * g2)
| {
|   int m1, m2;
    AFT.w   R9,     2

|   m1 = g1->mantisse;
    HIA     R0,     1(R1)
    BIG     R0,     -2(R8)
|   m2 = g2->mantisse;
    HIA     R0,     1(R8)
    HIA     R0,     1(R0)
    BIG     R0,     -3(R8)

|   res->exponent = aligneer (g1->exponent, &m1, g2->exponent, &m2);
| backup R1; R0 en R2 niet in gebruik
    BST     R1
| parameter g1->exponent
    HIA     R0,     2(R1)
    BST     R0
| parameter &m1
    HIA.a   R1,     -2(R8)
| parameter g2->exponent
    HIA     R0,     1(R8)
    HIA     R0,     2(R0)
    BST     R0
| parameter &m2
    HIA.a   R2,     -3(R8)

    BST     R8
    HIA     R8,     R9
    SBR     aligneer
    HST     R8

    OPT.w   R9,2
    HST     R1
| res->exponent = functieresultaat
    HIA     R2,     2(R8)
    BIG     R0,     2(R2)

| res->mantisse = m1 + m2;
    HIA     R0,     -3(R8)
    OPT     R0,     -2(R8)
    BIG     R0,     1(R2)

    OPT.w   R9,     2
    KTG

| ------------------------------------------------------------------------------
| void bwk_aft (struct bkg * res, register struct bkg * g1, struct bkg * g2)
| {
|   int m1, m2;
|
| Vul implementatie bwk_aft zelf verder aan. Probeer het gemeenschappelijke deel
```

```
| met bwk_opt eventueel m.b.v. een macro af te zonderen.


| ------------------------------------------------------------------------------
| void normaliseer (struct bkg * g)
| {
|   register int m, e;
    BST     R6
    BST     R5

|   m = g->mantisse;
|   e = g->exponent;
    HIA     R0,     1(R8)
    HIA     R6,     1(R0)
    HIA     R5,     2(R0)

|   if (m == 0) {
    VLG.w   R6,     0
    VSP     NGEL,   normaliseer_naif1
|      g->exponent = VERHOOG;
|      return;
    HIA.w   R1,     <VERHOOG>
    HIA     R0,     1(R8)
    BIG     R1,     2(R0)
    SPR     normaliseer_ret

|   if (m > MILJARD) {
normaliseer_naif1:
    VGL     R6,     MILJARD
    VSP     KLG,    normaliseer_wh
|      g->mantisse = m / 10;
    HIA     R0,     R6
    DEL.w   R0,     10
    HIA     R1,     1(R8)
    BIG     R0,     1(R1)
|      g->exponent++
    HIA     R0,     2(R1)
    OPT.w   R0,     1
    BIG     R0,     2(R1)
|      return;
    SPR     normaliseer_ret

normaliseer_wh:
|   while (m < HONDERDMILJOEN) {
    VGL     R6,     HONDERDMILJOEN
    VSP     GRG,    ewh
|     m *= 10;
|     e--;
    VER.w   R6,     10
    AFT.W   R5,     1
    SPR     normaliseer_wh

normaliseer_ewh:
|   g->exponent = e;
    HIA     R1,     1(R8)
    BIG     R5,     2(R1)
|   g->mantisse = m;
    BIG     R6,     1(R1)

normaliseer_ret:
    HST     R5
```

```
    HST     R6
    KTG
}

| ------------------------------------------------------------------------------
| main()
| {
|   int i, j;
|   struct bkg som;
|
| Vooreest moet je het gegeven C programma lineariseren (i.e., omzetten naar een
| equivalent C programma dat met eendimensionale rijen werkt). De gegeven matrix
| heeft grootte ATEST * AMETING:
|
|     struct bg meting[ATEST*AMETING];
|
| De for-loops kunnen dan vertaald worden als volgt (rij-linearisatie):
|
|     for (i = 0; i < ATEST; i++)
|         for (j = 0; j < AMETING; j++)
|             lezMeting(&meting[i*AMETING + j]);
|
| Vertaal zelf de gelineariseerde main functie.

EINDPR
```