Nota:

- indien 'register' in de declaratie van de lokale variabele of formele parameter staat, dan moet die variabele of parameter in een register worden bijgehouden (resp. doorgegeven); anders moet deze variabele op de stapel wordt bijgehouden (resp. via de stapel worden doorgegeven);
- een functie geeft zijn resultaat terug via R0 tenzij het niet in een register past (bijv. record): dan wordt de stapel gebruikt.

1. Teken de activatierecords en stel de toewijzingstabellen op voor de functies/procedures en het hoofdprogramma.

```
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);
   }
}
```

- lineariseer de matrix meting (rijlinearisatie)

Oefeningen

1. Vertaal het C-programma (zie huistaak)
2. Teken de volledige stapel op het ogenblik dat normaliseer wordt opgeroepen.