void multiply(int a[], int b[], int c[])  
/* This is to multiply radix-r multiple precision numbers a and b, and store the result in c */  
{
    int i, j;
    for (i=0; i<=2*n-1; i++) c[i]=0;
    for (i=0; i<=n-1; i++) {
        for (j=0; j<=n-1; j++) {
            c[i+j+1]=c[i+j+1] + (c[i+j]+a[i]*b[j]) / r;
            c[i+j]=(c[i+j] + a[i]*b[j]) % r;
        };
    };
}

This is to multiply a and b, and the product is stored into c. Indices i and j go through multiple precision numbers a and b respectively.

Example

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6</th>
<th>7</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>i goes on a</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>j goes on b</td>
</tr>
</tbody>
</table>

-----------
40 48 56 i=0; j=0, 1, 2

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>5</td>
<td>6</td>
<td>carry 5, remainder 6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-----------
35 42 49 i=1; j=0, 1, 2

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

-----------
4 4 4 4

-----------
3 8 4 4 2 6
void divide(int a[], int b[], int c[], int n, int m)
/* This is to divide radix-r multiple precision numbers.
a is divided by b and the quotient is stored in c.
The remainder is in a */
{
  int i, j, q, x;
  int a1[100];
  while (b[m-1]==0) m=m-1;
  a[n]=0;
  n=n+1; /* inserted 14 Aug. 2001 */
  // The following while loop normalizes b by multiplying it by 2
  // We need to multiply a as well to have correct quotient
  // At the end a is not the correct remainder due to normalization
  while (b[m-1]<=(r / 2)) {
    for(i=0;i<=m-1;i++) b[i]=2*b[i];
    for(i=0;i<=m-1;i++) {
      b[i+1]=b[i+1]+b[i] / r;
      b[i]=b[i] % r;
    }
    for(j=0;j<=n-1;j++) a[j]=2*a[j];
    for(j=0;j<=n-1;j++) {
      a[j+1]=a[j+1]+a[j] / r;
      a[j]=a[j] % r;
    }
  }
  for(j=n-m-1;j>=0;j--) {
    x=r*a[j+m]+a[j+m-1]; // when a[j+i] is negative, d=-a[j+i]deficit
    a[j+m-1]=x; // ...-d....-2r...-r....0.........
    a[j+m]=0; //     |     |     |     |     |     |
    q=x/b[m-1] + 1; //Guess q
    do{
      for(i=m-1;i>=0;i--) a[i]=a[i+1];
      /* if q is too large, decrease it by 1 */
      q=q-1;
      count=count+1;
      for(i=m-1;i>=0;i--) a[i]=a[i]-q*b[i];
      for(i=0;i<=m-2;i++) {
        // Try to subtract q*b from a
        if(a[i+1]<0){
          a[i+1]=a[i+1]+((r-1-a[i+1])/r)*r; //remaining at i+1
          a[i]=a[i]+((r-1-a[i+1])/r)*r; //borrow from left
          a[i]=a[i]+((r-1-a[i+1])/r)*r; //remaining at i+1
        }
      }
    } while(a[j+m-1]<0);
    c[j]=q;
    for(i=m-1;i>=0;i--) a[i]=a[i+1];
  } // for j

Until the leading digit of b becomes greater than or equal to r/2, we keep multiplying n-digit number a and m-digit number b. Then we guess the first digit for the quotient expressed by q. We copy the relevant portion of a into a1, and try to subtract q*b. If we hit negative, q is too large, and so subtract 1 from q, and follow the same procedure. If we can subtract, we go to the next position for the quotient. This idea comes from the simple mathematical fact that a/b = (ax)/(bx), where x is a power of 2 in our present problem.

Note that the radix, r, is very large, typically r = 2^14. Thus good guess of q will lead to vast speed-up.
Example. \( r=10, m=3, n=6 \).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 7 9</td>
<td>1 3 7 9</td>
</tr>
<tr>
<td>3 3 1</td>
<td>4 5 6 7 1 2</td>
</tr>
<tr>
<td>6 6 2</td>
<td>0 9 1 3 4 2 4</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
3 3 1 \\
6 6 2 \\
r = 263
\end{array}
\]

\[
\begin{array}{c}
2 5 1 4 \\
2 6 4 8 \\
1 9 8 6
\end{array}
\]

\[
\begin{array}{c}
x = 9, q = 1 \\
x = 25, q = 4 \\
\text{aborted}
\end{array}
\]

\[
\begin{array}{c}
5 2 8 2 \\
5 2 9 6 \\
4 6 3 4
\end{array}
\]

\[
\begin{array}{c}
x = 52, q = 8 \\
\text{aborted}
\end{array}
\]

\[
\begin{array}{c}
5 9 5 8
\end{array}
\]

\[
\begin{array}{c}
q = 7
\end{array}
\]

\[
\begin{array}{c}
x = 54, q = 9
\end{array}
\]

5 2 6 526 / 2 is correct remainder

General picture of subtraction

\[ j \text{ goes on dividend a and I goes on divisor b} \]

\[
\begin{array}{c|c|c|c}
\text{j} & \text{i} & \text{j+m-1} & \text{0} \\
\hline
\text{a1} & 2 & 5 & 1 & 4 \\
\text{-} & \text{q*b} & 2 & 6 & 4 & 8 \\
\hline
\text{a1} & 0 & -1 & -3 & -4 & \text{i=0, negative. Borrow 10 from neighbour} \\
0 & -1 & -4 & 6 & \text{i=1, negative. Borrow 10 from neighbour} \\
0 & -2 & 6 & 6 & \text{i=2, negative. Borrow 10 from neighbour} \\
-1 & 8 & 6 & 6 & \text{i=3} \\
\end{array}
\]

\( a1[j+m-1]<0 \), and so repeat the process with a new \( q \)

In this example, \( q*b \) is in a radix-10 form. In the program, \( q*b \) is given in crude form, e.g.,

\[
q*b = 4*(0, 6, 6, 2) = (0, 24, 24, 8) \\
a1 - q*b = (2, -19, -23, -4)
\]

For \( a1[0]=-4 \), we can borrow 1 from left neighbour, resulting \( a1[0]=6 \).
Now for \( a1[1]=-24 \), we borrow \((9+24)/10=3\) from left neighbour, resulting \( a1[1]=30+24=6 \), etc. The logic for \((r+1 + a1[j+i])\)/\( r \) for borrow is this is the minimum borrow to make the current position non-negative. For example, if \( a1[i+j]=-20 \), we borrow \((9+20)/2 = 2 \) from neighbour and the current value becomes 0.