Graduate students registered for one unit should complete the following project by April 27. The project is an extension of MP2. The objective is to apply the following two source-to-source transformations to YAPL programs. In other words, this project calls for modifying the abstract syntax tree generated by MP2. There is no need to print the result in YAPL. It suffices to print the result in C using MP3. Make sure that you handle all cases correctly.

The two transformations are:

1. **Loop unrolling.** All for loops in YAPL will be unrolled. If the number of iterations is a constant, the for loop will be fully unrolled. For example:

   ```yapl
   for i := 1 to 3 do
     begin
       a[i] := b[i];
       c := c + a[i];
     end
   ```

   would be transformed into

   ```c
   a[1] := b[1];
   c := c + a[1];
   a[2] := b[2];
   c := c + a[2];
   a[3] := b[3];
   c := c + a[3];
   ```

   If the number of iterations is unknown, the loop will be partially unrolled into sequences of $k$ iterations, where $k$ is a UNIX command parameter to the compiler. Thus, if $k=3$, the loop:

   ```yapl
   for i := 1 to n do
     begin
       a[i] := b[i];
       c := c + a[i];
     end
   ```

   would be transformed into

   ```c
   if n < 3 then
     for i := 1 to n do
       begin
         a[i] := b[i];
         c := c + a[i];
       end
   else
     begin
       for i := 1 to n-3 step 3 do
         begin
           a[i] := b[i];
           c := c + a[i];
           a[i+1] := b[i+1];
           c := c + a[i+1];
         end
   ```
\[
a[i+2] := b[i+2]; \\
c := c + a[i+2]; \\
end
\]

\[
\text{if } \text{mod}(n, 3) = 2 \text{ then} \\
\begin{align*}
\text{begin} \\
a[n-1] &:= b[n-1]; \\
c &:= c + a[n-1]; \\
\text{end}
\end{align*}
\]

\[
\text{if } \text{mod}(n, 3) \geq 1 \text{ then} \\
\begin{align*}
\text{begin} \\
a[n] &:= b[n]; \\
c &:= c + a[n]; \\
\text{end}
\end{align*}
\]

end

2. **Procedure integration.** All non-recursive function calls will be replaced with the body of the procedure. Notice that for this to work properly your translator may need to rename variables to maintain the original behavior of the program. For example:

```plaintext
program
integer r

function integer mod(integer p, integer q)
integer r
begin 
\begin{align*}
r &= n-m*(n/m); \\
\text{return } r;
\end{align*}
end

... 

if mod(n, 3) \geq 1 then
...
```

would be transformed into:

```plaintext
program
integer r
integer rnew
...

rnew = 3*(n/3);
if n-rnew \geq 1 then
...
```