

**1. Introduction.** This program implements the coroutines of Algorithms 7.2.1.1R and 7.2.1.1D, in the important case  $m = 2$ .

```
#define nn 10      /* we will test this value of n */
#include <stdio.h>
int p[nn];        /* program locations */
int x[nn], y[nn], t[nn], xp[nn], yp[nn], tp[nn];    /* local variables */
int n[nn];        /* the value of 'n' in each coroutine */
<Subroutines 2>;
main()
{
    register int k, kp;
    <Initialize the coroutines 3>;
    for (k = 0; k < (1 << nn); k++) printf("%d", co(nn - 1));
    printf("\n");
}
```

**2.** We simulate the behavior of recursive coroutines, in such a way that repeated calls on  $co(n - 1)$  will yield a cyclic sequence of period  $2^{nn}$  in which each  $nn$ -tuple occurs exactly once.

The coroutines are of types S (simple), R (recursive), and D (doubly recursive), as explained in the book. There are  $nn - 1$  coroutines altogether (see exercise 96); the main one will be number  $nn - 1$ .

Each coroutine  $q$ , for  $1 \leq q < nn$ , has a current position  $p[q]$ , as well as local variables  $x[q]$ ,  $y[q]$ , and so on; and it generates a de Bruijn sequence of length  $2^{n[q]}$ .

If  $n = 2$ , the coroutine for order  $n$  simply outputs the sequence 0, 0, 1, 1. Otherwise, if  $n$  is odd, coroutine  $q = n - 1$  invokes coroutine  $q - 1 = n - 2$  and doubles its length. Otherwise coroutine  $q = 2n' - 1$  invokes coroutines  $q - 1 = 2n' - 2$  and  $(q - 1)/2 = n' - 1$ , where coroutines  $2n' - 1$  through  $n'$  are “clones” of coroutines  $n' - 1$  through 1; the effect is to square the length of the cycles output by those coroutines.

```
#define S 0        /* base for positions of an S coroutine */
#define R 10       /* base for positions of an R coroutine */
#define D 20       /* base for positions of a D coroutine */
<Subroutines 2> ≡
void init(int r)
{
    register q = r - 1;
    n[q] = r;
    if (r ≡ 2) p[q] = S + 1;
    else if (r & 1) {
        p[q] = R;
        x[q] = 0;
        init(q);
    } else {
        register int k, qq;
        qq = q >> 1;
        p[q] = D + 1;
        x[q] = xp[q] = 2;
        init(qq + 1);
        for (k = q - 1; k > qq; k--) p[k] = p[k - qq], x[k] = x[k - qq], xp[k] = xp[k - qq], n[k] = n[k - qq];
    }
}
```

See also section 4.

This code is used in section 1.

3.  $\langle \text{Initialize the coroutines 3} \rangle \equiv$   
 $\text{init}(nn);$

This code is used in section 1.

4. Now here's how we invoke a coroutine and obtain its next value.

$\langle \text{Subroutines 2} \rangle + \equiv$   
**int**  $\text{co}(\text{int } q)$   
{  
  **switch** ( $p[q]$ ) {  
     $\langle \text{Cases for individual coroutines 5} \rangle$   
  }  
}

5. Each coroutine resets its  $p$  before returning a value. For example, type S is the simplest.

$\langle \text{Cases for individual coroutines 5} \rangle \equiv$   
**case**  $S + 1$ :  $p[q] = S + 2$ ; **return** 0;  
**case**  $S + 2$ :  $p[q] = S + 3$ ; **return** 0;  
**case**  $S + 3$ :  $p[q] = S + 4$ ; **return** 1;  
**case**  $S + 4$ :  $p[q] = S + 1$ ; **return** 1;

See also sections 6 and 7.

This code is used in section 4.

6. Type R is next in difficulty. I change the numbering slightly here, so that case  $R$  does the first part of the text's step R1. The text's  $n$  is  $n[q - 1]$  in this code, because of the initialization we've done.

$\langle \text{Cases for individual coroutines 5} \rangle + \equiv$   
R1: **case**  $R$ :  $p[q] = R + 1$ ; **return**  $x[q]$ ;  
**case**  $R + 1$ : **if** ( $x[q] \neq 0 \wedge t[q] \geq n[q - 1]$ ) **goto** R3;  
R2:  $y[q] = \text{co}(q - 1)$ ;  
R3:  $t[q] = (y[q] \equiv 1 ? t[q] + 1 : 0)$ ;  
R4: **if** ( $t[q] \equiv n[q - 1] \wedge x[q] \neq 0$ ) **goto** R2;  
R5:  $x[q] = (x[q] + y[q]) \& 1$ ; **goto** R1;

7. And finally there's the coroutine of type D. Again the text's parameter  $n$  is our variable  $n[q - 1]$ .

$\langle \text{Cases for individual coroutines 5} \rangle + \equiv$   
D1: **case**  $D + 1$ : **if** ( $t[q] \neq n[q - 1] \vee x[q] \geq 2$ )  $y[q] = \text{co}(q - (n[q] \gg 1))$ ;  
D2: **if** ( $x[q] \neq y[q]$ )  $x[q] = y[q]$ ,  $t[q] = 1$ ; **else**  $t[q]++$ ;  
D3:  $p[q] = D + 4$ ; **return**  $x[q]$ ;  
D4: **case**  $D + 4$ :  $yp[q] = \text{co}(q - 1)$ ;  
D5: **if** ( $xp[q] \neq yp[q]$ )  $xp[q] = yp[q]$ ,  $tp[q] = 1$ ; **else**  $tp[q]++$ ;  
D6: **if** ( $tp[q] \equiv n[q - 1] \wedge xp[q] < 2$ ) {  
  **if** ( $t[q] < n[q - 1] \vee xp[q] < x[q]$ ) **goto** D4;  
  **if** ( $xp[q] \equiv x[q]$ ) **goto** D3;  
}  
D7:  $p[q] = D + 8$ ; **return** ( $xp[q]$ );  
**case**  $D + 8$ : **if** ( $tp[q] \equiv n[q - 1] \wedge xp[q] < 2$ ) **goto** D3;  
**goto** D1;

**8. Index.***co*: [1](#), [2](#), [4](#), [6](#), [7](#).*D*: [2](#).*D1*: [7](#).*D2*: [7](#).*D3*: [7](#).*D4*: [7](#).*D5*: [7](#).*D6*: [7](#).*D7*: [7](#).*init*: [2](#), [3](#).*k*: [1](#), [2](#).*kp*: [1](#).*main*: [1](#).*n*: [1](#).*nn*: [1](#), [2](#), [3](#).*p*: [1](#).*printf*: [1](#).*q*: [2](#), [4](#).*qq*: [2](#).*R*: [2](#).*r*: [2](#).*R1*: [6](#).*R2*: [6](#).*R3*: [6](#).*R4*: [6](#).*R5*: [6](#).*S*: [2](#).*t*: [1](#).*tp*: [1](#), [7](#).*x*: [1](#).*xp*: [1](#), [2](#), [7](#).*y*: [1](#).*yp*: [1](#), [7](#).

- ⟨ Cases for individual coroutines [5](#), [6](#), [7](#) ⟩ Used in section [4](#).
- ⟨ Initialize the coroutines [3](#) ⟩ Used in section [1](#).
- ⟨ Subroutines [2](#), [4](#) ⟩ Used in section [1](#).

CO-DEBRUIJN

	Section	Page
Introduction .....	1	1
Index .....	8	3