```
IF HRS_WORKED <= 40 THEN
   CALL REGPAY;
ELSE
   CALL OTPAY;
```

Use `IF-ELSE` *to emphasize that*
only one of two actions is to be performed.

Another major aspect of control flow is looping. We are already familiar with the indexed loop, the

```
DO I = 1 TO N
```

of PL/I and the

```
DO 10 I = 1, N
```

of Fortran. But even more frequent are loops which are not arithmetic progressions, as in this sorting procedure:

```
SORT: PROCEDURE OPTIONS(MAIN);
      DECLARE  (NAMES(50),SPARE)CHARACTER(10),
      SWITCH BIT(1),(I,N) FIXED BINARY;
      /*READ IN ALL 50 NAMES                              */
      GET LIST(NAMES);
      N=50;
AGAIN:SWITCH='0'B;                      /*CLEAR THE SWITCH*/
      DO I=1 TO N-1;       /*SET THE NUMBER OF COMPARISONS*/
         IF NAMES(I)>NAMES(I+1) THEN    /*SWAP THE PAIR   */
             DO;                        /*USING SPARE,AND */
                SWITCH='1'B;            /*  SET THE SWITCH*/
                SPARE=NAMES(I);
                NAMES(I)=NAMES(I+1);
                NAMES(I+1)=SPARE;
             END;
      END;
      N=N-1;             /*DECREASE NUMBER OF  COMPARISONS*/
      IF SWITCH THEN GOTO AGAIN; /*REPEAT IF SWAP WAS MADE*/
      PUT LIST(NAMES);
END;
```

There are actually two loops here, although it takes a bit of work to find that out. The inner loop is clear enough; it runs from 1 to `N-1`. The outer loop is executed so long as an interchange has been made during a pass through the list of items. This is recorded by `SWITCH`, which is `'1'B` if an exchange has been made, and `'0'B` otherwise.

The PL/I `DO-WHILE` statement provides a way to write this loop that makes it instantly obvious to the reader that there *is* a loop, and what controls it.