▶ **Problem DS-03-10** The *Josephus problem* is the following game: N people, numbered 1 to N, are sitting in a circle. Starting at person 1, a hot potato is passed. After M passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins. Thus, if $M = 0$ and $N = 5$, players are eliminated in order, and player 5 wins. If $M = 1$ and $N = 5$, the order of elimination is 2, 4, 1, 5.

(a) Write a program to solve the Josephus problem for general values of M and N. Try to make your program as efficient as possible. Make sure you dispose of cells.

(b) What is the running time of your program?

(c) If $M = 1$, what is is the running time of your program? How is the actual speed affected by the *free* routine for large values of N ($N > 10,000$)?

Solution. (a) To solve this problem, we shall create a circularly linked list L with N nodes to simulate N people sitting in a circle. Without loss of generality we assume that $N \geq 2$. Initially, every node has a number for representing its order in L. Specially, the list without a header node. Starting a count with node number 1, we repeatedly perform an eliminating procedure to delete the node from L in every M passes until only one node is left. The following is the complete list of the algorithm.

```
struct Node;
typedef struct Node* List;
typedef struct Node* Position;

struct Node
{
   int         Number;
   Position    Next;
};

void Insert( int X , Position P )
{
    Position TmpCell;
    TmpCell =  (struct Node*) malloc(sizeof(struct Node));
    TmpCell->Number = X;
    TmpCell->Next = P->Next;
    P->Next = TmpCell;
}

void DeleteNext( Position P )
{
     Position TmpCell;
     TmpCell = P->Next;
     P->Next = TmpCell->Next;
     free(TmpCell);
}
```

```
int Josephus( int N , int M )
{
      List L;
      Position LastCell;
      int i, count=0;

      // Create a circularly linked list with N nodes
      L =  (struct Node*) malloc(sizeof(struct Node));
      L->Number = 1;
      LastCell = L->Next = L;
      for ( i = 2 ; i <= N ; i++ ) {
            Insert( i , LastCell );
            LastCell = LastCell->Next;
      }

      // Perform the elimination process
      while ( L != L->Next ) {
          if ( ++count == M ) {
               DeleteNext(L);
               count = 0;
          }
          L = L->Next;
      }
      return L->Number;
}

void main()
{
      int N,M;
      scanf("%d %d" , &N , &M );
      printf("The winner is %d\n" , Josephus( N , M ));
}
```

(b) In the above implementation of circularly linked list, the complexity is dominated in the elimination process. Since there are $N-1$ nodes to be deleted in the procedure and exactly M passes are needed for every elimination of a node, the running time is $O(NM)$.

(c) For the special case $M = 1$, the above algorithm is clearly linear, i.e., $O(N)$ time. The actual speed affected by the `free` routine for large N depends on the C compiler and the memory management routine of operating system. In fact, to speed up the execution of the program for $M = 1$, there exist alternate ways to calculate the function `Josephus`(N) (We omit the parameter M in this case.) Consider the following recursive function:

```
int Josephus( int N )
{
    if ( N == 1 ) return 1;
    if ( (N/2)*2 == N ) return 2*Josephus(N/2)-1;
    return 2*Josephus(N/2)+1;
}
```

Obviously, the function can be run in $O(\log N)$ time. To understand why this function works correctly, it is easy to see that to get the initial position of a person, we simply need to multiply his new position by two and subtract one. That is,

$$\texttt{Josephus}(2k) = 2 \cdot \texttt{Josephus}(k) - 1$$

Let us now consider the case of an odd N ($N > 1$), i.e., $N = 2k + 1$. The first pass eliminates people in all even positions. If we add to this the elimination of the person in position 1 right after that, we are left with an instance of size k. Here, to get the initial position that corresponds to the new position numbering, we have to multiply the new position number by two and add one. Thus, for odd N, we get

$$\texttt{Josephus}(2k+1) = 2 \cdot \texttt{Josephus}(k) + 1$$

Alternatively, if we consider the binary representation of N, `Josephus(`N`)` can also be obtained by one-bit left cyclic shift of N itself. For example,

$$\texttt{Josephus}(6) = \texttt{Josephus}(110_2) = 101_2 = 5$$

and

$$\texttt{Josephus}(7) = \texttt{Josephus}(111_2) = 111_2 = 7$$

Because we can compute one-bit left cyclic shift of N by $2N - (2^{\lfloor \log N \rfloor + 1} - 1)$, the function can be implemented as follows.

```
int Josephus( int N )
{
    return 2*N-(Pow(2,Log(N)+1)-1);
}
```

where `Pow`(a, N) is the function to calculate the value of a^N which has the running time $O(\log N)$ (see Page 32 of the textbook). Also, we are easily to implement the function `Log`(N) that converts N into $\lfloor \log_2 N \rfloor$ with running time $O(\log N)$. Therefore, the time complexity of the last implementation of `Josephus(`N`)` is $O(\log \log N)$. $\square$