問３　ライフゲームに関する次の記述を読んで，設問1～5に答えよ。

ライフゲームとは，数学者コンウェイが考案した，生命の誕生，生存，死滅などを再現したシミュレーションゲームである。マス目状の盤上の各マスに生命が存在でき，そのマス自身及び隣接するマスの状態によって次世代の誕生，生存，死滅が決まる。その条件を表1に示す。

なお，隣接するマスには，斜め方向のマスも含む。また，生命が存在するマスを“生のマス”，生命が存在しないマスを“死のマス”と呼ぶ。

表1　誕生，生存，死滅の条件

条件名	条件
誕生	死のマスに隣接する生のマスが三つならば，死のマスは次の世代では生のマスとなる。（死のマスに隣接する生のマスが二つ以下又は四つ以上ならば，死のマスは次の世代でも死のマスである。）
生存	生のマスに隣接する生のマスが二つ又は三つならば，生のマスは次の世代でも生のマスである。
過疎	生のマスに隣接する生のマスが一つ以下ならば，生のマスは過疎によって次の世代では死のマスとなる。
過密	生のマスに隣接する生のマスが四つ以上ならば，生のマスは過密によって次の世代では死のマスとなる。

〔4×4マスのシミュレーション〕

4×4マスの盤上における第3世代までのシミュレーションを図1に示す。

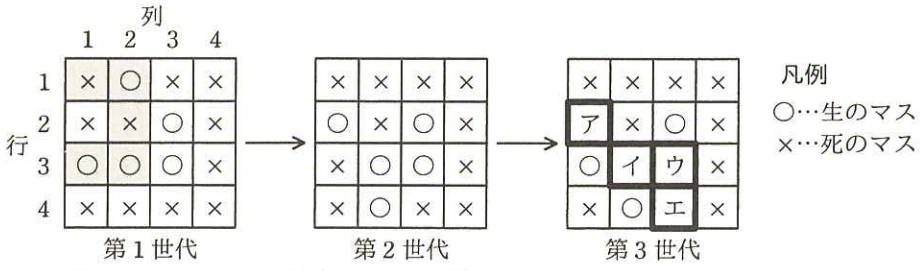

図1　4×4マスの盤上における第3世代までのシミュレーション

第1世代は，初期値として設定されたものである。例として，第1世代の2行目1列目のマスを考える。現在，このマスは死のマスである。このマスに隣接するマスを網掛けで示す。これら五つのマスの中に生のマスが三つある。これは表1の“誕

生”の条件に該当するので，第 2 世代の 2 行目 1 列目のマスは生のマスになる。同様に，第 1 世代の各マスについて，そのマス自身及び隣接するマスの状態を確認することで第 2 世代が決まる。次の世代への状態の更新は，全てのマスについて同時に行われる。

〔盤上のマスのデータ構造〕

N×N マスの盤上の状態を表現するデータ構造を考える。多次元配列が利用できないプログラム言語を考慮し，盤上の各マスの生死状態を管理するデータ構造として 1 次元配列 m を用いる。配列 m のデータ構造のイメージを図 2 に示す。

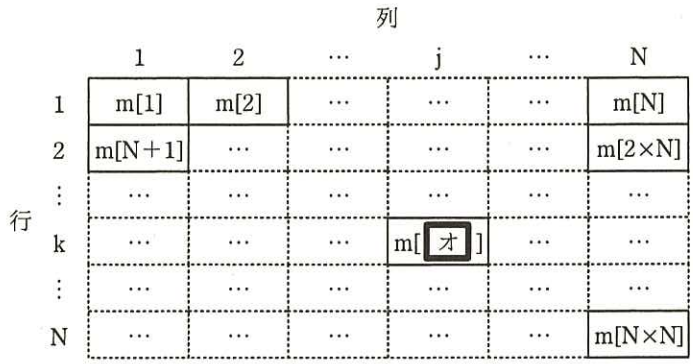

図 2　配列 m のデータ構造のイメージ

〔配列 m を次世代に更新するプログラム〕

使用する定数，配列及び関数を表 2 に，配列 m を次世代に更新する関数 update を図 3 に示す。

なお，関数に配列を引数として渡すときの方式は参照渡しである。

表 2　使用する定数，配列及び関数

名称	種類	説明
N	定数	盤の一辺の大きさ。3 以上の整数が入る。
m	配列	N×N マスの生死状態を管理する 1 次元配列。生の場合は 1 が，死の場合は 0 が入る。
temp	配列	配列 m と同じ大きさの 1 次元配列。
copy(array1, array2)	関数	配列 array1 の全ての要素を配列 array2 にコピーする。
clear(array)	関数	配列 array の全ての要素の値を 0 にする。

```
function update(m)
  copy(m, temp)       // 配列 m を配列 temp にコピーして退避する
  clear(m)            // 配列 m の全ての要素の値を 0 にする
  for( i を 1 から N×N まで 1 ずつ増やす )

    if( i-1 が N で割り切れる )
      a ← 0
      b ← 1
    elseif( i が N で割り切れる )
      a ← -1
      b ← 0
    else
      a ← -1
      b ← 1
    endif

    e ← 0
    for( y を -1 から 1 まで 1 ずつ増やす )
      for( x を a から b まで 1 ずつ増やす )
        if( (y と 0 が等しくない) or (x と 0 が等しくない) )
          c ← i + y×N + x
          if( (c が 1 以上) and (c が N×N 以下) )
            if( [        カ        ] )
              e ← e+1
            endif
          endif
        endif
      endfor
    endfor

    // 生死を判定する
    if( [        キ        ] )
      m[i] ← 1
    elseif( (temp[i]と 1 が等しい) and ( (e と 2 が等しい) or (e と 3 が等しい) ) )
      [        ク        ]
    endif

  endfor
endfunction
```

α

図 3　関数 update のプログラム

〔テストプログラム〕

図 3 のプログラムをテストするために，配列 m に第 1 世代が与えられたときの第 p 世代が，机上で作成した正しい結果である配列 r と等しいことを確認するプログラムを作成した。作成した関数 shouldBe を図 4 に示す。ここで，p には 2 以上の整数が入る。

```
 1: function shouldBe( m, p, r )
 2:   for( i を 1 から p まで 1 ずつ増やす )
 3:     update(m)
 4:   endfor
 5:   for( i を 1 から N×N まで 1 ずつ増やす )
 6:     if( m[i] と r[i] が等しくない )
 7:       return false    // テスト失敗
 8:     endif
 9:   endfor
10:   return true         // テスト成功
11: endfunction
```

図 4　関数 shouldBe のプログラム

図 3 のプログラムが正しく動作する状態で図 4 のプログラムを実行したところ，テストが失敗した。原因を調査した結果，図 4 の［　ケ　］行目に問題があることが判明したので，①プログラムを修正してテストを成功させることができた。

設問 1　図 1 中の［　ア　］～［　エ　］に入れる適切な生死状態を，図 1 の凡例に倣い答えよ。

設問 2　図 2 中の［　オ　］に入れる適切な字句を答えよ。

設問 3　図 3 中の［　カ　］～［　ク　］に入れる適切な字句を答えよ。

設問 4　図 3 中の α の二つの条件のいずれかを満たすのはどのような場合か。単語“盤”及び“マス”を用いて 30 字以内で述べよ。

設問 5　〔テストプログラム〕について，(1)，(2)に答えよ。

(1)　本文中の［　ケ　］に入れる適切な数値を答えよ。

(2)　本文中の下線①の修正後の［　ケ　］行目のプログラムを答えよ。