問３　パズルの解答を求めるプログラムに関する次の記述を読んで，設問 1～3 に答えよ。

太線で 3×3 の枠に区切られた 9×9 のマスから成る正方形の盤面に，1～9 の数字を入れるパズルの解答を求めるプログラムを考える。このパズルは，図 1 に示すように幾つかのマスに数字が入れられている状態から，数字の入っていない各マスに，1～9 のうちのどれか一つの数字を入れていく。このとき，盤面の横 1 行，縦 1 列，及び太線で囲まれた 3×3 の枠内の全てにおいて，1～9 の数字が一つずつ入ることが，このパズルのルールである。パズルの問題例を図 1 に，図 1 の解答を図 2 に示す。

2		1		9		7		
	4		2			3		
5					8		2	9
	9		6	7		2		
6			3		5			4
		7		4	9		1	
7	6		9					3
		9			6		4	
		4		1		6		

図 1　問題例

2	8	1	4	9	3	7	6	5
9	4	6	2	5	7	3	8	1
5	7	3	1	6	8	4	2	9
4	9	5	6	7	1	2	3	8
6	1	8	3	2	5	9	7	4
3	2	7	8	4	9	5	1	6
7	6	2	9	8	4	1	5	3
1	5	9	7	3	6	8	4	2
8	3	4	5	1	2	6	9	7

図 2　図 1 の解答

このパズルを解くための方針を次に示す。

方針：数字が入っていない空白のマスに，1～9 の数字を入れて，パズルのルールにのっとって全部のマスを埋めることができる解答を探索する。

この方針に沿ってパズルを解く手順を考える。

〔パズルを解く手順〕

(1)　盤面の左上端から探索を開始する。マスは左端から順に右方向に探索し，右端に達したら一行下がり，左端から順に探索する。

(2)　空白のマスを見つける。

(3)　(2)で見つけた空白のマスに，1～9 の数字を順番に入れる。

(4)　数字を入れたときに，その状態がパズルのルールにのっとっているかどうかをチェックする。

(4-1) ルールにのっとっている場合は，(2)に進んで次の空白のマスを見つける。

(4-2) ルールにのっとっていない場合は，(3)に戻って次の数字を入れる。このとき，入れる数字がない場合には，マスを空白に戻して一つ前に数字を入れたマスに戻り，(3)から再開する。

(5) 最後のマスまで数字が入り，空白のマスがなくなったら，それが解答となる。

〔盤面の表現〕

この手順をプログラムに実装するために，9×9 の盤面を次のデータ構造で表現することにした。

・9×9 の盤面を 81 個の要素をもつ 1 次元配列 board で表現する。添字は 0 から始まる。各要素にはマスに入れられた数字が格納され，空白の場合は 0 を格納する。

配列 board による盤面の表現を図 3 に示す。ここで括弧内の数字は配列 board の添字を表す。

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]
[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]
[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]	[35]
[72]	[73]	[74]	[75]	[76]	[77]	[78]	[79]	[80]

図 3　配列 board による盤面の表現

〔ルールのチェック方法〕

パズルのルールにのっとっているかどうかのチェックでは，数字を入れたマスが含まれる横 1 行の左端のマス，縦 1 列の上端のマス，3×3 の枠内の左上端のマスを特定し，行，列，枠内のマスに既に格納されている数字と，入れた数字がそれぞれ重複していないことを確認する。このチェックを“重複チェック”という。

〔解法のプログラム〕

プログラムで使用する配列，関数，変数及び定数の一部を表 1 に示す。なお，表 1 の配列及び変数は大域変数とする。

表1　プログラムで使用する配列，関数，変数及び定数の一部

名称	種類	内容
board[ ]	配列	盤面の情報を格納する配列。 初期化時には問題に合わせて要素に数字が設定される。
solve(x)	関数	パズルを解くための手順を実行する関数。 盤面を表す board[ ]の添字 x を引数とする。
row_ok(n, x)	関数	横 1 行の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
column_ok(n, x)	関数	縦 1 列の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
frame_ok(n, x)	関数	3×3 の枠内の重複チェックを行う関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 数字の重複がない場合は true，重複がある場合は false を返す。
check_ok(n, x)	関数	row_ok，column_ok，frame_ok を呼び出し，全ての重複チェックを実行する関数。チェック対象の数字 n，チェック対象のマスを示す添字 x を引数とする。 全てのチェックで数字の重複がない場合は true，一つ以上のチェックで数字の重複がある場合は false を返す。
div(n, m)	関数	整数 n を整数 m で割った商を求める関数。
mod(n, m)	関数	整数 n を整数 m で割った剰余を求める関数。
print_board( )	関数	board[ ]の内容を 9×9 の形に出力する関数。
row_top	変数	数字を入れようとするマスが含まれる横 1 行の左端のマスを示す添字を格納する変数。
column_top	変数	数字を入れようとするマスが含まれる縦 1 列の上端のマスを示す添字を格納する変数。
frame_top	変数	数字を入れようとするマスが含まれる 3×3 の枠内の左上端のマスを示す添字を格納する変数。
MAX_BOARD	定数	盤面に含まれるマスの数を表す定数で 81。

解法のプログラムのメインプログラムを図 4 に，関数 solve のプログラムを図 5 に，重複チェックを行うプログラムの一部を図 6 に示す。

```
function main()
  board[]を初期化する    //問題を盤面に設定する
  solve(0)               //盤面の左上端のマスを示す添字を引数として関数 solve を呼び出す
endfunction
```

図4　メインプログラム

```
function solve(x)
  if (x が MAX_BOARD−1 より大きい)
    print_board()                          //解答を出力する
    exit()                                 //メインプログラムの処理を終了する
  else
    if ( [ア] )                            //対象のマスが空白でない場合
      solve ( [イ] )                       //次の探索
    else
      for (n を 1 から 9 まで 1 ずつ増やす)   //1〜9 の数字を順にマスに入れる
        if ( [ウ] )
          board[x] ← n
          solve ( [イ] )                   //次の探索
          board[x] ← [エ]                  //再帰から戻った場合のマスの初期化
        endif
      endfor
    endif
  endif
endfunction
```

図 5　関数 solve のプログラム

```
function row_ok(n, x)                      //横 1 行の重複チェック
  row_top ← [オ]                           //行の左端のマスを示す添字を求める
  for (i を 0 から 8 まで 1 ずつ増やす)
    if ( [カ] )
      return false
    endif
  endfor
  return true
endfunction

function column_ok(n, x)                   //縦 1 列の重複チェック
  column_top ← [キ]                        //列の上端のマスを示す添字を求める
  for (i を 0 から 8 まで 1 ずつ増やす)
    if ( [ク] )
      return false
    endif
  endfor
  return true
endfunction

function frame_ok(n, x)                    //3×3 の枠内の重複チェック
  frame_top ← x − [ケ] − mod(x, 3)         //枠内の左上端のマスを示す添字を求める
  for (i を 0 から 2 まで 1 ずつ増やす)
    for (j を 0 から 2 まで 1 ずつ増やす)
      if (board[frame_top + 9 * i + j]が n と等しい)
        return false
      endif
    endfor
  endfor
  return true
endfunction
```

図 6　重複チェックを行うプログラムの一部

〔プログラムの改善〕

解法のプログラムは深さ優先探索であり，探索の範囲が広くなるほど，再帰呼出しの回数が指数関数的に増加し，重複チェックの実行回数も増加する。

そこで，重複チェックの実行回数を少なくするために，各マスに入れることができる数字を保持するためのデータ構造 Z を考える。データ構造 Z は盤面のマスの数×9の要素をもち，添字 x は 0 から，添字 n は 1 から始まる 2 次元配列とする。Z[x][n]は，ゲームのルールにのっとって board[x]に数字 n を入れることができる場合は要素に 1 を，できない場合は要素に 0 を格納する。データ構造 Z の初期化処理と更新処理を表 2 のように定義した。

なお，データ構造 Z は大域変数として導入する。

表 2　データ構造 Z の初期化処理と更新処理

処理の名称	処理の内容
初期化処理	初期化時の盤面に対し，個々の空白のマスについて 1～9 の数字を入れた場合の重複チェックを行う。 重複チェックの結果によって，初期化時の盤面の状態で個々の空白のマスに入れることができない数字は，データ構造 Z の該当する数字の要素に 0 を設定する。それ以外の要素には 1 を設定する。
更新処理	空白のマスに数字を入れたとき，そのマスが含まれる横 1 行，縦 1 列，3×3 の枠内の全てのマスを対象に，データ構造 Z の該当する数字の要素を 0 に更新する。

〔パズルを解く手順〕の(1)の前にデータ構造 Z の初期化処理を追加し，〔パズルを解く手順〕の(2)～(5)を次の(2)～(4)のように変更した。

(2)　空白のマスを見つける。

(3)　データ構造 Z を参照し，(2)で見つけた空白のマスに入れることができる数字のリストを取得し，リストの数字を順番に入れる。

(3-1) 入れる数字がある場合，①処理 A を行った後，マスに数字を入れる。その後，データ構造 Z の更新処理を行い，(2)に進んで次の空白のマスを見つける。

(3-2) 入れる数字がない場合，マスを空白に戻し，②処理 B を行った後，一つ前に数字を入れたマスに戻り，戻ったマスで取得したリストの次の数字から再開する。

(4)　最後のマスまで数字が入り，空白のマスがなくなったら，それが解答となる。

設問1　図5中の［ア］～［エ］に入れる適切な字句を答えよ。

設問2　図6中の［オ］～［ケ］に入れる適切な字句を答えよ。

設問3　〔プログラムの改善〕について，下線①の処理A及び下線②の処理Bの内容を，“データ構造Z”という字句を含めて，それぞれ20字以内で述べよ。