Consider the following simplified variant of the card game “Blackjack”. Two players are alternately dealt cards, which are random integers between 1 and 10, one at a time. (Assume that each deal is independent; the numbers are dealt by rolling a 10-sided die, not by dealing from a finite deck.) At each stage, a player may either request a card, or may pass and not get assigned a card. If a player’s total is exactly 21, he immediately wins. If a player’s total exceeds 21, then he immediately loses. If a player has passed on one round, he is permitted to continue drawing cards on later rounds. If both players have passed, then the player with the higher total wins. In the case of a tie, the second player to pass wins.

For instance, consider the following sequences with two players:

Sequence 1:
- Player A draws 8.
- Player B draws 3.
- Player A draws 7.
- Player B draws 5.
- Player A draws 8, and loses (exceeds 21).

Sequence 2:
- Player A draws 8.
- Player B draws 3.
- Player A draws 9.
- Player B draws 7.
- Player A passes.
- Player B draws 9.
- Player A draws 3.
- Player B draws 7 and loses (exceeds 21).

We now generalize the above in two ways. First, the number of card values $N_{Cards}$ may be different from 10. Second, instead of a single target value 21, we have a target range, from $L_{Target}$ to $U_{Target}$. If the player reaches a total between $L_{Target}$ and $U_{Target}$, inclusive, he immediately wins. If the player’s total exceeds $U_{Target}$, he immediately loses.

The problem is a combination of a Markov decision problem with an adversary game.

The strategy can be computed using a dynamic programming implementation of a probabilistic calculation. First, we note the following:

A. If it is player X’s turn to play, then his optimal move is determined by a game state consisting of three parts: Whether or not the player Y has just passed, X’s total points, and Y’s total points.

B. If it is X’s turn to play and X’s total is less than or equal to Y’s, then X should definitely draw, because if he passes, Y can immediately pass and win.

C. It will never happen that the game ends with both players passing, because whichever player would lose will do better to take a chance on drawing.
In view of these observations, the optimal strategy and the player's chances of winning in any given situation can be expressed in two arrays. The Boolean array \( \text{Play}[X, Y] \) of size \( L\text{Target} \times L\text{Target} \), indexed from 0 to \( L\text{Target}-1 \), gives the optimal move for player X in the case where player Y did not pass on the previous move, where \( X \) and \( Y \) are the current totals for X and Y. \( \text{Play}[X, Y]=1 \) if X should draw, 0 if he should pass. The three-dimensional Boolean array \( \text{Prob}[X, Y, P] \) of size \( L\text{Target} \times L\text{Target} \times 2 \), 0-indexed, is the probability that X will win from state \( X, Y \) if he makes move \( P \).

The assignment, then, is to write a function \( \text{Blackjack(NCards,LTarget,UTarget)} \) returning the two arrays \( \text{Play} \) and \( \text{Prob} \).

The two arrays are filled in together working backward, from the game's end to its beginning. For instance, if \( L\text{Target} = 21 \) the algorithm computes first \( \text{Prob}[20, 20, 1] \), then \( \text{ProbDraw}[20, 20, 0] \) and \( \text{Play}[20, 20] \); then \( [19, 20] \) and \( [20, 19] \); then \( [18, 20] \), \( [19, 19] \), and \( [20, 18] \); and so on.

The value of the three arrays is filled in as follows:

```plaintext
for (SUM = 2*LTarget-2 downto 0) {
    if (SUM >= LTarget-1) {
        XTOP = LTarget-1;
        XBOT = SUM+1-LTarget;
    } else {
        XBOT = 0;
        XTOP = SUM;
    }
    for (XT = XBOT to XTOP)
        YT = SUM - XT;
        \text{Prob}[XT,YT,0] = the probability of winning if X passes in state XT,YT;
        \text{Prob}[XT,YT,1] = the probability of winning if X draws in state XT,YT;
        \text{Play}[XT,YT] = (\text{Prob}[XT,YT,1] > \text{Prob}[XT,YT,0]);
} }
```

The probability that X will win if he draws in state \( XT,YT \) can be computed by considering each possible deal. If X draws a card with value \( \text{CARD} \) and neither wins nor loses, then it will be Y's turn, and Y will be in the state \( YT, XT+\text{CARD} \). Y will make the move \( \text{Play}[YT,XT] \). The probability that Y wins in that state is \( \text{Prob}[YT,XT+\text{CARD},\text{Play}[YT,XT]] \); hence the probability that X wins if Y is in that state is \( 1 - \text{Prob}[YT,XT+\text{CARD},\text{Play}[YT,XT]] \).

Computing the probability that X will win if he draws in state \( XT,YT \):

```plaintext
{ ProbWinning = 0.0
  for (CARD=1:NCards) {
    if (XT+CARD > UTarget) then ProbYWins = 1;
    elseif (XT+CARD >= LTarget) then ProbYWins = 0;
    else ProbYWins = Prob[YT,XT+CARD,Play[YT,XT+CARD]];
    ProbWinning = ProbWinning + (1-ProbYWins)/NCards;
  } endfor
  return ProbWinning
}
```

If X passes, then, if his score is less than Y, he has zero probability of winning; if his score is at least equal to Y, then Y must draw.

2
Computing the probability that X will win if he passes in state $X_T, Y_T$
(assuming Y has drawn on the previous move):
if ($X_T \leq Y_T$) then 0 else $1 - \text{Prob}[Y_T, X_T, 1]$

In a program that actually played the game, there would be an additional simple routine that
instructs X what do to if Y has just passed,

\begin{verbatim}
if (Y has just passed) then
    if ($X_T \geq Y_T$) then pass else draw
\end{verbatim}

but we do not need an additional data structure to apply that rule; and the presumption that X
will use that rule is built into our algorithms above.

**Input and Output**

The input is just the three numbers: $N_{Cards}, L_{Target}, U_{Target}$. You may use any reasonable
input format (e.g. command line, input file, standard input, text window, arguments to function in
an interpreter) but DO NOT HARD CODE THEM.

The output is the two of $L_{Target} \times L_{Target}$ arrays, $\text{Prob}$ and $\text{Play}$ described above.