lab12

```
$ gcc roll.o lab12.c
lab12.c:155:1: warning: non-void function does not return a value in all control pat
hs [-Wreturn-type]
}
^
1 warning generated.

$ ./a.out
RollDiceFair 390000 times
Occurrences  1     2     3     4     5     6
  die 1:  64750 65201 65392 64936 64652 65069
  die 2:  64906 65095 65001 65088 64777 65133
  die 3:  65234 65103 65034 65218 64650 64761
  die 4:  64978 65001 64948 65303 64845 64925
  die 5:  65442 64978 65351 64835 64320 65074

RollDiceUnfair 390000 times
Occurrences  1     2     3     4     5     6
  die 1:  59496 60328 59834 60033 60018 90291
  die 2:  59932 60537 59668 59500 59861 90502
  die 3:  65209 64757 64868 65064 65114 64988
  die 4:  65029 64737 64771 64497 65551 65415
  die 5:  64724 64841 64762 65071 65541 65061

Player 2 using unfair dice playing 10000000 games:
  Winning percentage: 49.87%
  Losing percentage: 50.13%
```

```
score: 76.0
o. [Output] Program output is correct, good.
o. [Coding] lab12.c spelling errors: dimentions(1), occured(2)
o. [Format] Program format can be improved.
o. [Compiler] warnings should be eliminated.
o. [Efficiency] can be improved.
o. [Program] legibility can be improved.
```

lab12.c

```c
// EE231002 Lab12. Poker Dice
// 110060007, 黃俊穎
// 2021/12/29

#include <stdio.h>
#include <stdlib.h>
#include "roll.h"

// function to count how many times dice's points have occured
void counter(int dice[5], int occur[5][6]);

int main(void)
{
    int dice[5];    // array to record fair dice's points
    int i, j, k;    // variables for loops
    int undice[5];  // array to record unfair dice's points
    // initialize all elements in "occurrence array" toward output form
    int occur[5][6] = {{0}};
    int fair, unfair;   // corresponded result from 5 dice' points
    double win = 0;     // initialize win number by testing many times

    printf("RollDiceFair 390000 times\n");
    printf("Occurrences  1     2     3     4     5     6\n");
    // loop to call functions to observe occurrences of fair dice
    for (i = 0; i < 390000; i++) {
        rollDiceFair(dice);
        counter(dice, occur);
    }
    // loop to print out point occurrences of each fair die
    for (i = 0; i < 5; i++) {
        printf("  die %d:  %5d %5d %5d %5d %5d %5d\n", i + 1, occur[i][0],
        occur[i][1], occur[i][2], occur[i][3], occur[i][4],occur[i][5]);
            occur[i][1], occur[i][2], occur[i][3], occur[i][4], occur[i][5]);
    }
    printf("\n");

    printf("RollDiceUnfair 390000 times\n");
    printf("Occurrences  1     2     3     4     5     6\n");
    // initialize 2-dimentions array to record point occurrences
    // of each fair die
    for (j = 0; j < 5; j++) {
        for (k = 0; k < 6; k++) {
            occur[j][k] = 0;
        }
    }
    // loop to call functions to observe occurrences of unfair dice
    for (i = 1; i <= 390000; i++) {
        rollDiceUnfair(undice);
        counter(undice, occur);
    }
    // loop to print out point occurrences of each unfair die
    for (j = 0; j < 5; j++) {
        printf("  die %d:  %5d %5d %5d %5d %5d %5d\n", j + 1, occur[j][0],
        occur[j][1], occur[j][2], occur[j][3], occur[j][4],occur[j][5]);
            occur[j][1], occur[j][2], occur[j][3], occur[j][4], occur[j][5]);
    }
    printf("\n");

    printf("Player 2 using unfair dice playing 10000000 games:\n");
    // set up loop to play 10000000 times
    for (k = 1; k <= 10000000; k++) {
        rollDiceFair(dice);
        fair = rank(dice);
        rollDiceUnfair(undice);
        unfair = rank(undice);
        // if unfair dice win, add number of winning times
        if (unfair < fair)  win++;
        // if game is ended in a tie, then play again
        if (unfair == fair) k--;
    }
    // print out winning and losing percentage of unfair dice
    printf("  Winning percentage: %2.2lf%%\n", win / 100000);
    printf("  Losing percentage: %2.2lf%%\n", 100 - win / 100000);

    return 0;
}

// function to count how many times dice's points have occured
void counter(int dice[5], int occur[5][6])
{
    int i;  // variable for loop
    // loop to record final times of each point
    for (i = 0; i < 5; i++) {
        switch (dice[i]) {
          case 0:  occur[i][0]++;
            case 0:  occur[i][0]++;
                   break;
          case 1:  occur[i][1]++;
            case 1:  occur[i][1]++;
                   break;
          case 2:  occur[i][2]++;
            case 2:  occur[i][2]++;
                   break;
          case 3:  occur[i][3]++;
            case 3:  occur[i][3]++;
                   break;
          case 4:  occur[i][4]++;
            case 4:  occur[i][4]++;
                   break;
          case 5:  occur[i][5]++;
            case 5:  occur[i][5]++;
        }
    }
}

// function to make dice's point by mod 6 of random number
void rollDiceFair(int dice[5])
{
    int i;
    for (i = 0; i < 5; i++) {
        dice[i] = rand() % 6;
    }
}

// function to arrange and give condition of each dice type
int rank(int dice[5])
{
    int i, j;   // variables for loops
    int count = 0;
    // counter to record how many times 5 dice's number are the same
    // by comparing each other
    int sum = dice[4];
    // sum of all dice

    // loop to record how many times 5 dice's number are the same
    // and sum up the points of dice
    for (i = 0; i < 4; i++) {
        for (j = i + 1; j < 5; j++) {
            if (dice[i] == dice[j])
                count++;
        }
        sum += dice[i];
    }

    switch (count) {
        // if no point is same, it may be Straight or Bust
        // 10 = 1 + 2 + 3 + 4; 14 = 2 + 3 + 4 + 5
        // otherwise, rest answers are Bust
        case 0:  if (sum == 10 || sum == 14)
                    return Straight;
                 else
                    return Bust;
                 break;
        // C represent combination sign
        // C(2,2) = 1
        case 1:  return OnePair;
                 break;
        // C(2,2) * 2 = 2
        case 2:  return TwoPair;
                 break;
        // C(3,2) = 3
        case 3:  return ThreeKind;
                 break;
        // C(3,2) + C(2,2) = 4
        case 4:  return FullHouse;
                 break;
        // C(4,2) = 6
        case 6:  return FourKind;
                 break;
        // C(5,2) = 10
        case 10:  return FiveKind;
    }
}
```