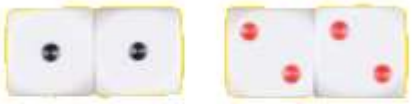
CSlam!

CSlam! is our simulation of **part** of a real board game named Yamslam. We will simulate rolling dice using a random number generator and our C program will determine which dice combination is created by the dice. Our player will get the option to reroll the dice to get a different combination. **For now**, our program will only roll all of the dice and will determine the combination and display it and allow the player to reroll up to 2 more times. Your program will be looking the following seven combinations.

2 Pairs
Two pairs of dice showing the same face.



Full House
Three-of-a-kind and a pair.

3-of-a-Kind
Three dice showing the same face.

4-of-a-Kind
Four dice showing the same face.

Small Straight
Four dice following each other.

Large Straight
Five dice following each other.

CSlam!
All five dice showing the same face.

Several videos are attached to this assignment in Canvas that show the game being played. Your goal is to follow the directions given in this assignment and create the same game as shown in the video. Read through the entire assignment to understand how the functions work and how they are called in `main`. The indention in the specification is intended to help you with creating your program – it matters, so pay attention to it so it can help you write your code. This assignment is more about exercising newly learned coding techniques (for loops and arrays) and not about figuring out how to code the rules of the game which is why I have provided the exact logic needed to create the game. Please follow it.

Vocabulary

"Die" refers to a single die – dice is the plural. The face of a die is a single side of the cube. The side facing up when a die is sitting on a flat surface is the "face" we use when playing games with dice.

Create your `Code5_xxxxxxxxxx.c` file.

At the top of the program, outside of any function (including `main`), add the following `#define` values

Create a `#define` named `NUMBEROFREROLLS` and set it to `3`. This number represents the number of rolls the player is allowed.

Create a `#define` named `NUMBEROFFACES` and set it to `6`. This number represents the number of faces on a die. CSlam uses a six sided die. There are lots of other types of dice, so we want our program to be flexible.

Create a `#define` named `NUMBEROFDICE` and set it to `5`. This number represents the number of dice used to play CSlam. CSlam uses 5 but other dice games use other quantities of dice, so we want our program to be flexible.

Create a function named `RollDice`. It has a return type of `void` and takes one parameter – an integer array named `dice`. This function will use the random number generator to generate numbers from 1 to `NUMBEROFFACES`. Use a `for` loop to fill up `dice` array. Your `for` loop must utilize `NUMBEROFDICE` and NOT use a hardcode value for the loop test.

```
for (int i = 0; i < NUMBEROFDICE; …
{
        Generate a random number between 1 and NUMBEROFFACES and store in dice[i]
}
```

Create a function named `PrintRoll`. It has a return type of `void` and takes one parameter – an integer array named `dice`. This function will use a `for` loop to print out the results of `dice`. Your `for` loop must utilize `NUMBEROFDICE` and NOT use a hardcode value for the loop test. This function should ONLY print the dice values and not anything else.

```
for (int i = 0; i < NUMBEROFDICE; …
{
        Print out dice[i]
}
```

Create a function named `HowManyFaces`. It has a return type of integer and takes two parameters – an integer array named `dice` and an integer named `SearchFace`. This function will calculate how many cells in `dice` are equal to the value passed in `SearchFace`. Your `for` loop must utilize `NUMBEROFDICE` and NOT use a hardcode value for the loop test.

create a variable named `NumberOfFaces` and initialize it to `0`

```
for (int i = 0; …
```

{

if dice[i] is equivalent to the value in SearchFace, then increment NumberOfFaces

}

return NumberOfFaces

For example, if you rolled the following

2 4 6 2 4

Then, if SearchFace is passed in as 4, then this function would return 2 since there are two 4's in this roll. This function will be called NUMBEROFFACES times in a for loop from main() – one time for each face of a die.

In the main function,

Declare the following variables

an integer array named dice. Create it using NUMBEROFDICE.

an integer array named howMany. Create it using NUMBEROFFACES.

an integer named numberOfRolls and initialize it to 0

an integer named numberOfReRolls and initialize it to NUMBEROFREROLLS

a character named answer and initialize to N

an integer named ofAKind and initialize it to 0

an integer named FullHouse and initialize it to 0

an integer named TwoPair and initialize it to 0

an integer named CSlam and initialize it to 0

an integer named LargeStraight and initialize it to 0

an integer named SmallStraight and initialize it to 0

an integer named i and initialize it to 0

Create a do-while loop

do

Call function RollDice and pass in the array dice. This function will use a random number generator to simulate the rolling of each die represented by each cell of the array dice.

Print the screen output as shown in the example output – "You rolled " is print in main() – not in PrintRoll().

Call the function PrintRoll and pass in array dice. This function will print out the values of all dice in the array.

Increment numberOfRolls

Use a for loop to call function HowManyFaces. This for loop will fill in each cell of howMany with the count of faces from the player's roll (the return value of function HowManyFaces). The for loop will call the

function HowManyFaces for the number of faces (NUMBEROFFACES) - use NUMBEROFFACES to create the loop test. The for loop iterator will be passed to the function HowManyFaces as the second parameter and will be used as the value for face in that function. 1 is added to the iterator when it is passed since a die does not have a face 0.

```
for(i = 0; …
{
      howMany[i] = HowManyFaces(dice, i+1);
}
```

For example, if the player rolls

```
1     6     4     3     6
```

then, after the for loop completes, the array howMany will have the following in it

0	1	2	3	4	5
1	0	1	1	0	2

Cell 0 in the array howMany contains how many 1's are in the player's roll – there's one.

Cell 1 in the array howMany contains how many 2's are in the player's roll – there's none.

Cell 2 in the array howMany contains how many 3's are in the player's roll – there's one.

Cell 3 in the array howMany contains how many 4's are in the player's roll – there's one.

Cell 4 in the array howMany contains how many 5's are in the player's roll – there's none.

Cell 5 in the array howMany contains how many 6's are in the player's roll – there's two.

Use a for loop to check each cell of howMany and run rules for each type of dice combination in order to detect them. Looping over array howMany for each die face will allow us to detect each type of combination.

```
for (i = 0; i < NUMBEROFFACES; i++)
{
```

if howMany[i] contains 3, then add 3 to variable FullHouse

if howMany[i] contains 2, then add 2 to variable FullHouse and increment variable TwoPair

if howMany[i] contains NUMBEROFDICE, then increment variable CSlam.

if howMany[i] contains 1, then increment variable LargeStraight

else if howMany[i] contains 0 AND variable LargeStraight is greater than 0 and less than 5, then set variable LargeStraight equal to 0

if howMany[i] is greater than or equal to 1, then increment variable SmallStraight

else if howMany[i] contains 0 AND variable SmallStraight is greater than 0 and less than 4, then set variable SmallStraight equal to 0

if howMany[i] contains 4, then set variable ofAKind to 4

if `howMany[i]` contains 3, then set variable `ofAKind` to 3

}

Now we look at the results of looping over the `howMany` array and print out which combination our looping found.

if variable `LargeStraight` is 5, then print "`Large Straight`"

else if variable `SmallStraight` is greater than or equal to 4, then print "`Small Straight`"

else if variable `FullHouse` is 5, then print "`Full House`"

else if variable `CSlam` is 1, then print "`CSlam!!`"

else if variable `ofAKind` is 4, then print "`Four of a kind`"

else if variable `ofAKind` is 3, then print "`Three of a kind`"

else if variable `TwoPair` is 2, then print "`Two Pair`"

else print "`You have nothing`"

After printing out the combination, if the player has any rerolls left, ask the player if they want to reroll.

if `numberOfRolls` is not equal to `NUMBEROFREROLLS`, then

ask the player "`Do you want to reroll? `" and store the answer in the variable `Answer`

Reset the following variables to 0 to prepare for the next roll

`ofAKind`

`FullHouse`

`TwoPair`

`CSlam`

`LargeStraight`

`SmallStraight`

while `Answer` is Y AND `numberOfRolls` is less than `numberOfReRolls`

Grading

When the GTA grades your program, they will be looking for specific rubric criterion, of course, but they will also be altering your program slightly. A prototype for function `TestRollDice()` will be added. They will then change the value of `NUMBEROFREROLLS` from 3 to 23 and change your function `RollDice()` to `TestRollDice()`. These alterations will allow your program to run 23 times and go through a subset of all possible dice combinations. All 7 combinations will be tested. Please see the video labeled “GTA Test Run” so see how it runs.

Sample Output

```
You rolled


Roll #1
1     3     3     6     6
Two Pair



Do you want to reroll? y

You rolled


Roll #2
1     3     4     5     5
You have nothing



Do you want to reroll? y

You rolled


Roll #3
1     1     1     3     5
Three of a kind
```