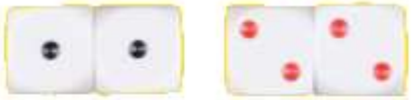
C++Slam!

C++Slam! is our simulation of **part** of a real board game named Yamslam. We will simulate rolling 5 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. The player will be allowed to roll 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

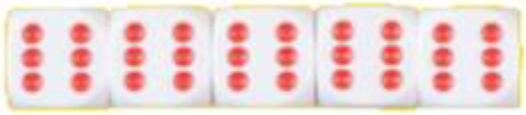
Four dice following each other.

Large Straight

Five dice following each other.

C++Slam

All five dice showing the same face.



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.

A video is attached to this assignment in Canvas that shows 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.

Create your `Code2_xxxxxxxxxx.cpp` file where `xxxxxxxxxx` is your 10 digit student id number.

Create a `makefile` that can compile and link `Code2_xxxxxxxxxx.cpp` and `Test.cpp`.

Use the version of `Test.cpp` provided with the assignment. Do not make any changes to `Test.cpp` – you need to add to your `Code2_xxxxxxxxxx.cpp` file in order to use `Test.cpp`. Your code will be graded with the provided `Test.cpp` – not with the copy you submit; therefore, any changes you make to `Test.cpp` to make your program work will not be there when your code is graded.

When your program is run with a command line parameter of `TEST`, it should call the `TestRollDice()` function in `Test.cpp` rather than the regular `RollDice()` function. The total number of allowed rolls should be set to define for the number of test rolls. When the program is running in TEST mode, it should print the number of passed and failed tests and the `PickDice()` function should not be called. Use the return value of the `TestRollDice()` function to determine if a test passed or failed. See video for details. Note : you should only try to use `argv[1]` if `argc` is 2 and you need to verify that the command line parameter is `TEST` – not some other word, spelling or capitalization – see video for example.

Create four defines

the number of dice – set to 5

the number of faces per die – set to 6

the number of rolls – set to 3

the number of test rolls – set to 23

Use these defines throughout your program. Do not hardcode these values anywhere in your program.

Your code should contain the following functions

`RollDice`

Return value : `void`

Parameters : dice vector

Generate enough random numbers to fill the vector back to the number of dice being used. Use `push_back()` to add each die value to the vector.

`PrintRoll`

Return value : `void`

Parameters : dice vector

Prints the value of the dice (as shown in the video) using a range based for loop.

`FillHowMany`

Return value : `void`

Parameters : dice vector and how many faces vector

Uses range based for loops ONLY. Initialize `howMany` vector to 0. Loop over dice vector and determine how many of each face are showing.

For example, if the player rolls

```
1 6 4 3 6
```

then, when the function completes, the vector `howMany` will have the following in it

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

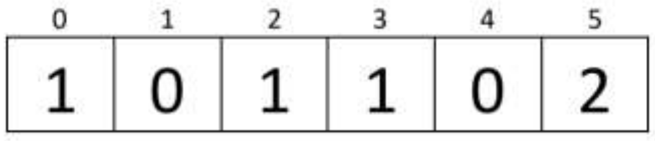
Cell 1 in the vector `howMany` contains how many 2's are in the player's roll – there's none.

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

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

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

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



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

`PickDice`

Return value: `void`

Parameters: dice vector

Print instructions on picking which dice to keep and which to reroll. Prompts user for each die and uses `erase()` to remove a die from the dice vector.

Part of this assignment is detecting when certain combinations occur. Creating that logic from scratch is not part of the assignment. I am giving you that logic here. If you choose to not use the provided logic, then make sure your logic works and satisfies all test cases.

Given these integer variables initialized to 0

`ofAKind`

`FullHouse`

`TwoPair`

`CPPSlam`

`LargeStraight`

`SmallStraight`

Set them according to the values stored in the `howMany` vector.

```
Loop over howMany

      if howMany[i] is 3, then add 3 to FullHouse

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

      if howMany[i] is 5, then increment CPPSlam

      if howMany[i] is 1, then increment LargeStraight
      else if howMany[i] is 0 and LargeStraight is greater than 0 and LargeStraight is
      less than 5, then set LargeStraight equal to 0

      if howMany[i] is greater than or equal to 1, then increment SmallStraight
      else if howMany[i] is 0 and SmallStraight is greater than 0 and SmallStraight is
      less than 4, then set SmallStraight equal to 0

      if howMany[i] is 4, then set ofAKind to 4

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

After checking every element of `howMany`

```
      if LargeStraight is 5, then "Large Straight"
      else if SmallStraight is greater than or equal to 4, then "Small Straight"
      else if FullHouse is  5, then "Full House"
      else if CPPSlam is 1, then "C++Slam!"
      else if ofAKind is 4, then "Four of a Kind"
      else if ofAKind is 3, then "Three of a Kind"
      else if TwoPair == 2, then "Two Pair"
      else "You have nothing"
```