Exercise 8
Applying Pointers & Virtual Functions: Rolling Dice Simulation

Introduction:
Pointers are data type in which the content is not a numerical value but rather an address of a memory item. When declaring and initializing a pointer follow the syntax below:

```
datatype pointerName = new datatype;
```

Since a pointer directs to memory address, it is also possible that it can handle the limitation of static arrays by allocating its size during runtime. Refer to the syntax below

```
int size;
pointerName = new datatype [size];
```

After using the dynamic array use the delete operator to deallocate all the memory location associated with the pointer. You must use the delete operation when implementing a destructor of a class with pointer members.

```
delete[] pointerName;
```

Virtual functions is a feature in C++ in which the binding of an object with respect to its function is done on runtime rather than on compile time. Thus enabling an attribute in a derived class to have the same form but of different function from the base class, otherwise known as polymorphism. To do this just add the virtual reserved word prior to the return value of the function [1].

The concept of probability goes in line with common sense. For example the probability of a coin toss is 50-50 (i.e. 50% heads or 50% tails). It doesn't mean that when tossing a coin 10 times 5 would be heads and 5 would be tail. But, by tossing it a million times the proportionality of getting a head or a tail is close to 50% [2]. Simulating the likelihood of a probable scenario is an important study in software modelling.

Procedure:

1. Create a project that would accommodate the necessary headers, implementation details and client code.

2. Make a header name `dice.h` with the class declaration indicated below

```
class Dice
{
    public:
      //default number of sides is 6
      Dice();
      Dice( int);
      ~Dice()
      void setDiceSides(int);
      virtual int rollDice() const;
    protected:
      int numSides;
};
```

3. Implement the class definition of the `Dice` class on another cpp file. Use the `srand (time(NULL))` function on the constructors to seed the `rand()` function on the `virtual int rollDice()`. Note: include the `<cstdlib >` and `<time.h>` libraries so that these function would work. You may test the functionality of this class on the client code.

4. Create a derived class from Dice, named `RiggedDice()` that would modify the `int RollDice ()const` function so that it would return 25% of the time the maximum number of sides. Implement the function definition on another .cpp file.

5. Create another class named `twoDice` with the functions indicated below

```
class TwoDice
{
   public:
         /*default suffleSize is 10, number of size 6. Allocate the
         dynamic arrays *dice1Suffle and *dice2Suffle at constructors */
         TwoDice();
         TwoDice(int numSides, int sufSize);

         //destructor
          ~TwoDice();

         /*roll dice 1 and rigged dice 2 and store it on dynamic arrays
         *dice1Suffle and *dice2Suffle respectively*/
         void suffle();

        //returns dice 1 suffle result at index
         int dice1SuffleAt(int);

         //returns dice 2 suffle result at index
         int dice2SuffleAt(int);

         //returns the suffle size
         int getSuffleSize();
   private:
         int suffleSize;
         int *dice1Suffle;
         int *dice2Suffle;
         Dice d1;
         RiggedDice d2;
};
```

6. Implement the definition of these functions then test you `TwoDice` class on the client code. Preferably try the code below.

```
 TwoDice TD(6,15);
TD.suffle();
cout<<"Dice 1 Results:"<<endl;
for (int i=0;i<TD.getSuffleSize();i++)
{
    cout<<TD.dice1SuffleAt(i)<<",";
}
cout<<endl;

cout<<"Dice 2 Results:"<<endl;
```

```
for (int i=0;i<TD.getSuffleSize();i++)
{
   cout<<TD.dice2SuffleAt(i)<<",";
}
```