# ← The Game of Craps

**Project 5**

You will be implementing a dice game, called Craps. I have no idea why it's called that. It's not much of a game, either. More like a... fancy way of flipping a coin? Haha.

It will read random numbers from a file that you specify on the command line, meaning that once you make your kernel module, you will be able to use it to generate the random numbers!

---

## Game Rules

Craps is played with **two regular dice with numbers 1-6** which are rolled at the same time. The sum of these two dice is computed. If the sum is:

- 2, 3, or 12, **the player immediately loses.**
- 7 or 11, **the player immediately wins.**
- Any other number, this number becomes **"the point."** In this case, the player **keeps rolling dice** until:
  - The player rolls a 7, in which case **the player loses.**
  - The player rolls **the point** again, in which case **the player wins.**
  - Any other number means nothing; keep rolling dice automatically until one of the two conditions above occurs.

---

# Your Program

1. Ask the player their name, and display a welcome message.
2. Ask if the player would like to play or quit.
3. Roll **two dice,** and display their values and the total.
    - If the player won, congratulate the player.
    - If the player lost, tell them.
    - Otherwise, store the first roll as **the point** and keep rolling until the player wins or loses. The point is only set **on the first roll.**
4. Go back to step 2.

**Use string inputs from the player. Do not make an integer menu or use single characters.**

As an example interaction:

```
root@tiny ~ # ./craps /dev/dice

Welcome to Jarrett's Casino! Please enter your name: Jarrett
Jarrett, would you like to Play or Quit? play
You have rolled 5 + 2 = 7
You Win!
Would you like to play again? no
Goodbye, Jarrett!

root@tiny ~ # _
```

---

# Reading the dice rolls from the file

Your program should **check if the user has given a command-line argument** and then **try to open that argument for reading binary data.** Be sure to check for errors.

You can read a single die roll from the file with `fread` into an `unsigned char` variable.

When writing and testing your program on thoth, you can use `/dev/urandom` as a random number generator. However, each byte it generates is in the range `0 .. 255` so you will have to use modulo to get it into the correct range. It's fine to leave the modulo in your program, since it will still work with `/dev/dice`.

---

# Writing and building your program

You can write and test this program entirely on thoth. Compile it as usual, with `-Wall -Werror --std=c99` and all that stuff.

You should test it in your VM after you write your device driver, too. However, the VM doesn't have all the dynamic libraries and is only 32-bit, so use this command line to compile it:

```
gcc -m32 -static -Wall -Werror --std=c99 -o craps craps.c
```

Now you have the executable, and you need to get it into your VM... this is a job for `scp`. `scp` means "secure copy" and uses SFTP to copy a file between computers.

Let's say your craps executable is **on thoth** at the path `~/private/proj5/craps`.

Inside the VM (using your username, not USERNAME, lol):

```
scp USERNAME@thoth.cs.pitt.edu:~/private/proj5/craps .
```

It will ask for your password, and if you enter it correctly, it will copy the `craps` executable into the current directory.

Now you can test it with `/dev/dice` (or even `/dev/urandom`; that exists in the VM too):

```
root@tiny ~ # ls
craps dice_dev.ko
root@tiny ~ # ./craps /dev/dice

Welcome to Jarrett's Casino! Please enter your name: _
```