Assignment 1
Pass the Pigs

Prof. Darrell Long
CSE 13S – Fall 2021

Due: October 3rd at 11:59 pm

1 Introduction

> *The creatures outside looked from pig to man, and from man to pig, and from pig to man again; but already it was impossible to say which was which.*
>
> —George Orwell, *Animal Farm*

We are going to implement a simplified version of David Moffat's dice game[1] *Pass the Pigs*. The game itself requires no skill and no real decision-making, making it a great game for the purpose of introductory programming in **C**. It will involve using all the basic programming language facilities: primitive types, loops, conditionals, arrays, and handling user input.

2 The Rules of the Game

> *No one believes more firmly than Comrade Napoleon that all animals are equal. He would be only too happy to let you make your decisions for yourselves. But sometimes you might make the wrong decisions, comrades, and then where should we be?*
>
> —George Orwell, *Animal Farm*

The simplified version of *Pass the Pigs* that we will implement can be played with any k players, such that $2 \leq k \leq 10$. The players are arranged in a cyclic fashion. The players will take turns rolling an asymmetrical die, affectionately named the *pig*, to earn points. Rolling the pig can result in it landing in any of the following positions:

1. **Side**: The pig lands on one of its sides ($\frac{2}{7}$ chance).
2. **Razorback**: The pig lands on its back ($\frac{1}{7}$ chance).
3. **Trotter**: The pig lands upright ($\frac{1}{7}$ chance).
4. **Snouter**: The pig lands on its snout ($\frac{1}{7}$ chance).
5. **Jowler**: The pig lands on one of its ears ($\frac{2}{7}$ chance).

[1] The traditional game *Pig* (`https://en.wikipedia.org/wiki/Pig_(dice_game)`) is played with a single six-sided die. This simple dice game was first described in print by John Scarne in 1945. Players take turns to roll a single die as many times as they wish, adding all roll results to a running total, but losing their gained score for the turn if they roll a 1. It has similarities with the traditional Mongolian game *Shagai* (`https://en.wikipedia.org/wiki/Shagai`).

Rolling **Side** yields 0 points and immediately ends the current player's turn, resulting in the pig being passed to the next player in the ring of players. Assuming k players and 0-based indexing, the next player to go after player 0 is player 1. After player 1 is player 2. Who is the player after player $k-1$? That would be player 0.

Rolling either **Razorback** or **Trotter** earns 10 points for the player. Rolling **Snouter** earns 15 points. Lastly, rolling **Jowler** earns 5 points. The game ends when any player has earned 100 or more points.

John Scarne

3 Game Abstractions

All Animals Are Equal. But Some Animals Are More Equal Than Others.

—George Orwell, *Animal Farm*

One of the most important skills to polish as Computer Scientists is the ability to create abstractions for the problems that we need to solve. In the case of "Pass the Pigs," the most immediate problem is to provide an abstraction for is the pig itself. Without some way of representing the pig, and the rolling of the pig, we cannot even begin to hope to implement the game in its entirety.

3.1 Enumerating the Positions

Your first thought might be to simply pseudorandomly generate a random number n such that $1 \leq n \leq 7$, then assign mappings from those integers to each of the positions the pig can land in. Generating pseudorandom numbers is required, but we will elect to use *enumerations* to represent each of the positions. Enumerations, `enum` in **C**, are used to provide names for integer constants. Using this, we can represent the positions and the pig in the following manner:

```
 1 typedef enum { SIDE, RAZORBACK, TROTTER, SNOUTER, JOWLER } Position;
 2 const Position pig[7] = {
 3   SIDE,
 4   SIDE,
 5   RAZORBACK,
 6   TROTTER,
 7   SNOUTER,
 8   JOWLER,
 9   JOWLER
10 };
```

The `typedef` is used to give a new name to a type. In this case, we used `typedef` to name the enumeration of positions as `Position`. The pig, then, can be represented as an *array* of positions. The act of "rolling" the pig can be achieved by randomly selecting 1 of the 7 elements of the `pig` array.

3.2 Generating Pseudorandom Numbers

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.

—John von Neumann, 1951

To generate pseudorandom numbers, the ones required to simulate the rolling of the pig, you will need a *pseudorandom number generator* (PRNG). You will the utilize the one defined by the standard **C** library (`libc`) which is interfaced with the two functions `srandom()` and `random()`. You will need to include `<stdlib.h>` in order to use these functions.

The `srandom()` function is essential in order to make your program *reproducible*. `srandom()` is used to *set the random seed*, which effectively establishes the start point of the pseudorandom number sequence that is generated. This means, after calling `srandom()` with a seed to set, that the pseudorandom numbers that are generated with `random()` *always* appear in the same order. Try out the following snippet of code:

```c
#include <stdio.h>
#include <stdlib.h>

#define SEED 2021

int main(void) {
    for (int i = 0; i < 3; i += 1) {
        puts("Set the random seed.");
        srandom(SEED);
        for (int i = 0; i < 5; i += 1) {
            printf(" - generated %lu\n", random());
        }
    }
}
```

The output of running the pseudorandom number generation example is given here so you can check if the numbers produced on your system matches the ones produced by the system in which your program will be graded on.

```
$ ./prng
Set the random seed.
 - generated 1644198542
 - generated 653251166
 - generated 273593510
 - generated 1964717451
 - generated 314322227
Set the random seed.
 - generated 1644198542
 - generated 653251166
 - generated 273593510
 - generated 1964717451
 - generated 314322227
Set the random seed.
 - generated 1644198542
 - generated 653251166
 - generated 273593510
 - generated 1964717451
 - generated 314322227
```

4 Your Task

> *"You have been my friend," replied Charlotte, "That in itself is a tremendous thing."*
>
> —E.B. White, *Charlotte's Web*

You will implement the version of *Pass the Pigs* using the rules presented in §2, placing the implementation in the source code file `pig.c`. The structure of your program should follow these steps:

1. Prompt the user to input the number of players, scanning in their input from `stdin`. You will want to use `scanf()` for this. To scan an `int` from `stdin`:

```
int input = 0;
scanf("%d", &input);
```

In the event that the user inputs anything other than a valid integer between 2 and 10 inclusive, print the following error to `stderr` informing them of improper program usage, then use the default value of 2 as the number of players:

```
fprintf(stderr, "Invalid number of players. Using 2 instead.\n");
```

What are `stdin` and `stderr`? In UNIX, every process on creation has access to the following input/output (I/O) streams: `stdin`, `stdout`, and `stderr`. A running program is a process. `stdin`, or *standard input*, is the input stream in which data is sent to be read by a process. `stdout`, or *standard output*, is the output stream where data written by a process is written to. The last stream, `stderr`, or *standard error*, is an output stream like `stdout`, but is typically used for error messages.

2. Prompt the user to input the random seed for this run of *Pass the Pigs*. In the event that the user inputs anything other than a valid seed, print the following error to `stderr` informing them of improper program usage, the use the default value of 2021 as the random seed:

```
fprintf(stderr, "Invalid random seed. Using 2021 instead.\n");
```

3. Set the random seed and make sure that each player starts off with 0 points. Note that it isn't made explicit how you keep track of each player's points. There are, of course, many ways to go about this. You are encouraged to keep things simple, however.

4. Proceed around the circle starting from player 0. For each player:

   (a) Print out the name of the player currently rolling the pig. An array of player names that you *must use* will be provided in the header file `names.h`. Index 0 of this names array is the name of the player 0, index 1 is the name of the player 1, and so on and so forth.

   (b) Roll the pig, increasing the player's point count until they either win the game or the pig lands on one of its two sides.

   (c) If the player has greater or equal to 100 points, then the win the game and a congratulatory message is printed to `stdout` celebrating their achievement.

   (d) If the rolled pig lands on one of its two sides, then the player's turn ends and the next player in the circle gets to have their shot at rolling the pig.

A working reference program, or *binary*, can be found in the course resources repository on `git.ucsc.edu`. It is also in this repository that you will find the header file of player names, `names.h`. To receive full credit, the output of your program *must* match the output of the reference program.

5 Deliverables

That'll do pig, that'll do.

—Farmer Hoggett, *Babe*

You will need to turn in the following source code and header files:

1. `names.h`: This contains the array of player names to be used in your implementation of the game. This file will be provided in the course resource repository and *may not* be modified.

2. `pig.c`: This contains the implementation of the game.

You will also need to turn in the following:

1. `Makefile`: This file will be provided in the course resource repository. It is a file that directs program compilation, building the program `pig` from `pig.c`. Make sure to attend section to learn about using a `Makefile` and how to write your own.
2. `README.md`: This must use proper Markdown syntax. It must describe how to use your program. Note down any known bugs or errors in this file as well for the graders.
3. `DESIGN.pdf`: This document *must* be a proper PDF. This design document must describe your design and design process for your program with enough detail such that a sufficiently knowledgeable programmer would be able to replicate your implementation. This does not mean copying your entire program in verbatim. You should instead describe how your program works with supporting pseudocode.

6 Submission

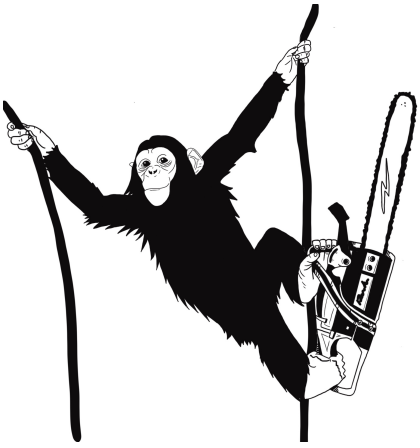
> *This work was strictly voluntary, but any animal who absented himself from it would have his rations reduced by half.*
>
> —George Orwell, *Animal Farm*

Refer back assignment 0 for the instructions on how to properly submit your assignment through `git`. Remember: *add*, *commit*, and *push*!.

Your assignment is turned in *only* after you have pushed and submitted the commit ID you want graded on Canvas. "I forgot to push" and "I forgot to submit my commit ID" are not valid excuses. It is *highly* recommended to commit and push your changes *often*.

7 Supplemental Readings

- *The C Programming Language* by Kernighan & Ritchie
  - Chapters 2 – 4
  - Chapter 7 §7.2, §7.4, §7.6
  - `man 3 random`
  - `man 3 printf`



I learned long ago, never to wrestle with a pig. You get dirty, and besides, the pig likes it. —George Bernard Shaw