Assignment 1
Pass the Pigs

by Prof. Darrell Long
edited by Dr. Kerry Veenstra and Jess Srinivas
CSE 13S, Spring 2023
Document version 3 (changes in Section 10)

Due April 26, 2023, at 11:59 pm

1 Introduction

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

The simplified version of Pass the Pigs that we will implement can be played with any k players, such that $2 \leq k \leq 10$. Players 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 positions listed in the following table. The game ends when one player gets 100 points or more.

Position	**Probability**	**Points**	**Next Action**
Jowler	2/7	5	Player rolls again
Razorback	1/7	10	Player rolls again
Trotter	1/7	10	Player rolls again
Snouter	1/7	15	Player rolls again
Side	2/7	0	Next player rolls

3 Game Abstractions

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 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 $0 \leq n \leq 6$, 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,

[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`.

`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:

```
/*
* Create a new type called "Position".
* Any variable of this type acts like an int.
* The enum defines contants with the names SIDE, RAZORBACK, etc.
* where SIDE == 0, RAZORBACK == 1, etc.
*/
typedef enum {SIDE, RAZORBACK, TROTTER, SNOUTER, JOWLER} Position;

/*
* Define a constant array pig that assigns a pig position to each value
* of a pseudorandom number from 0 through 6.
* SIDE appears twice because it has 2/7 chance of being rolled.
* The same goes for JOWLER.
*/
const Position pig[7] = {
    SIDE,
    SIDE,
    RAZORBACK,
    TROTTER,
    SNOUTER,
    JOWLER,
    JOWLER,
};
```

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 one of the seven elements of the pig array.

3.2 Generating Pseudorandom Numbers

To generate the pseudorandom numbers that are required to simulate the rolling of the pig, you will need a pseudorandom number generator (PRNG). You will use the one defined by the standard C library, 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()` sets 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, that the pseudorandom numbers that are generated by `random()` always appear in the same order.

Try out the following test program, `prng.c`:

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

#define SEED 2023

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

```
    return 0;
}
```

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

```
./prng
Set the random seed.
 - generated 1033193930
 - generated 278388770
 - generated 1574118255
 - generated 1189045628
 - generated 28839610
Set the random seed.
 - generated 1033193930
 - generated 278388770
 - generated 1574118255
 - generated 1189045628
 - generated 28839610
Set the random seed.
 - generated 1033193930
 - generated 278388770
 - generated 1574118255
 - generated 1189045628
 - generated 28839610
```

4 Your Tasks

You will be performing three main tasks:

1. Design your program
2. Write your program in the C programming language
3. Write a report about your program

I'm distinguishing between the first two tasks because they really are separate. The first task is to understand the data and the algorithms that are needed to, in this case, simulate the Pass the Pigs game as described in the rules of Section 2. You confirm your understanding of the data and algorithms of that design by writing a design report describing what you know. Then, when you have an idea of what you want to do, you translate that idea into the C programming language.

As encouragement to start on your design report, you will be submitting a *draft* of your design report early! For points! (See the Assignment on Canvas.) Turning in your draft means following the sequence *add / commit / push / submit commit ID* for your draft document before the draft's deadline.

Yes, this assignment has two deadlines.

After you have a draft of your design, you write your program, placing the implementation in the source code file `pig.c`. You will find starter files for this assignment in the course resources repository on `git.ucsc.edu`. The starter files are `names.h` and `Makefile`.

Finally, you submit a final design report that reports how well your program works and mentions any unintended shortcomings. You submit this final design report along with your source code by the final assignment deadline.

5 Program Structure

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.

   ```
   int num_players = 2;
   printf("Number of players (2 to 10)? ");
   int scanf_result = scanf("%d", &num_players);
   ```

   The variable `scanf_result` stores the return value of `scanf`. `scanf` returns the number of elements read successfully, which in this case would either be 0 or 1. If the retun value is 0, we have not read successfully, and must throw and error. We must also validate that the input is a number in the required range. In the case 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:

   ```
   if (scanf_result < 1 || num_players < 2 || num_players > 10) {
       fprintf(stderr, "Invalid number of players. Using 2 instead.\n");
   }
   ```

   (What is `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.

   ```
   unsigned seed = 2023;
   printf("Random-number seed? ");
   num_assignments = scanf("%u", &seed);
   ```

   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, and then use the default value of 2023 as the random seed:

   ```
   if (num_assignments < 1) {
       fprintf(stderr, "Invalid seed. Using 2023 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 player 0, index 1 is the name of player 1, and so on and so forth. Print the name using `printf()`, not `fprintf()` (since the player name is not an error). Use the `printf()` format string `"%s\n"`.

   (b) Roll the pig, increasing the player's point score until they either win the game or the pig lands on one of its two sides. Report the roll and the player's new score using the `printf()` format string `" rolls %d, has %d\n"`. This is the step where you use the PRNG program `random()` and the modulus (`%`) operator to generate a random number that is used to access the `pig[]` array (see Section 3.1).

(c) If the player has greater or equal to 100 points, then they win the game and a congratulatory message is printed celebrating their achievement. Use the `printf()` format string `"%s won!\n"`.

(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.

6 Testing Your Program

To receive full credit, the output of your program must match the output of a reference program that we provide. You can test your program by comparing its output to the output of the reference program automatically. Below I describe how to do that.

The reference program (often called a "binary") can be found in the course resources repository on `git.ucsc.edu`. **Since new Apple laptops use a different microprocessor than older Apple laptops and Windows laptops, we are providing two different binaries. Run the binary that is appropriate for your laptop.**

1. Using `vim` or whichever editor you have been using to write your C program, create an input file called `input1.txt`. The file contains only two lines, which are the responses to the `pig` program's two prompts. Those prompts are for the number of players and for the seed of the PRNG. Let's use 2 players and a random seed of 3.

   ```
   2
   3
   ```

2. Run the reference program using `input1.txt` for input, and save the output for later comparison to the file `expect1.txt`.

   If you have a new Apple laptop with an M1 or M2 processor, run this reference program:

   ```
   ./pig_arm < input1.txt > expect1.txt
   ```

   otherwise run this reference program:

   ```
   ./pig_x86_64 < input1.txt > expect1.txt
   ```

3. Run your own program and save the output for later comparison. Name the output file `output1.txt`.

   ```
   ./pig < input1.txt > output1.txt
   ```

4. You now have two output files: `expect1.txt` and `output1.txt`. Compare the two output files using the command-line program `diff`.

   ```
   diff expect1.txt output1.txt
   ```

   The `diff` program will display any differences between the output of the reference program (`pig_arm` or `pig_x86_64`) and the output of your `pig` program. If the files are identical, then diff will print nothing.

You can run more than one test. For example, to make a second test, create an input file `input2.txt`, the expected-output file `expect2.txt`, and your program's output file `output2.txt`. Then run the `diff` program on `expect2.txt` and `output2.txt` to check whether they are the same.

7 Deliverables

You will need to turn in the following source code and header files. **Note: do not turn in the executable file `pig`. In this class you never add an executable file to your repository. Add only source code.** You can do this by running `make clean` before you make every commit.

1. `pig.c`: This C source-code file contains your implementation of the game. It must include the definition of `main()` and any supporting functions that you write.
2. `report.pdf`: This document must be a proper PDF that is well formatted and readable. This report must describe your program as a whole and the design process that you used to create it. It will also describe the results of the program. More information about it can be found on the template. 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.
3. `Makefile`: **Do not edit this file.** We provide this file, but you will turn it in to ensure that we can compile your program. This file directs program compilation, building the program `pig` from `pig.c`.
4. `names.h`: **Do not edit this file.** We provide this file, but you will be turning it in to ensure that we can compile your program. This file contains the array of player names to be used in your implementation of the game. Do not change the names of the players. **The automatic grading system is looking for exact output from your program, including the player names.**

8 Submission

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

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.

Remember that you have two deadlines: You will be turning in a draft of your design report early, and then you will be turning in a final version of your design report along with your source code.

Be sure to format your code using the `make format` rule in the Makefile provided. This requires installing the `clang-format` package, if you have not already. Not running `clang-format` before submitting your code will result in reduced points. Your `.clang-format` file should already be located in the root directory of your git repository so that it can be used for future assignments also.

9 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`

10 Revisions

Version 1 Original.

Version 2 Correct a few grammar errors. Add information about `make format`, change design to report, and include information about the new requirements.

Version 3 Clarify usage of `scanf`