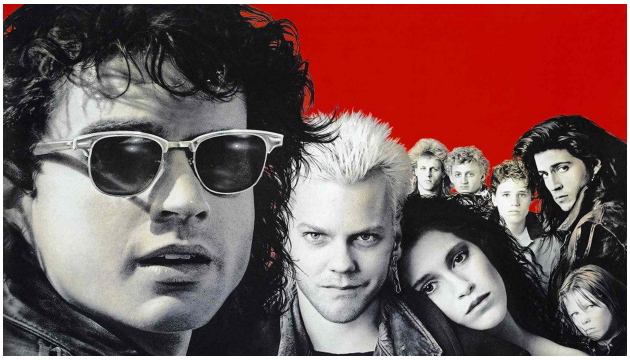
Assignment 1
The Garlic Game



Prof. Darrell Long
CSE 13S – Winter 2021

Due: January 17th at 11:59 pm

1 Introduction

> *Garlic does not usually destroy vampires but gives them a hell of a burn. Use it when wanting to deter them, but do not place all your faith in an herb that goes into spaghetti; it is just an inherently wrong philosophy and most likely damaging to your health. Yet, if garlic is ingested or injected into a vampire, theory suggests that they will slowly begin to disintegrate internally and become nothing but dust in the wind.*
>
> —R. M. Foreman, *Vampire Slaying for Dummies 101*

Vampire teens are quite similar to human teens: rebellious, moody, and prone to engaging in dangerous activities. Vampire parents are all about "free blood" and "drink blood, not wine." Vampire teens tend to disregard their parents and instead smoke wolfsbane and play risky games such as *the Garlic Game*. In underground vampire clubs, the Garlic Game is a spectator sport. Spectators watch excitedly as the thick sent of garlic wafts through the air, with the pungent scent of vampire corpses disintegrating from the inside out filling their nostrils. Being the victor of the Garlic Game is every vampire teen's dream come true. But how does one win the Garlic Game? Simple: by being lucky. You will soon be so well-versed with the Garlic Game that you can even *implement the game in* ***C***.

© 2021 Darrell Long

2 Playing the Game

> *In America, the young are always ready to give those who are older than themselves the full benefits of their inexperience.*
>
> —Oscar Wilde, *The American Invasion*, 1887

The Garlic Game is played with anywhere from 2 to 10 vampires, inclusive. The vampires stand in a circle and each have 3 lives to start with. The game progresses through *rounds*. Here is what occurs during a round:

- Each vampire will roll *two 6-sided* dice if they are alive (perhaps we should say *undead*), going around the circle in sequence. Only undead vampires roll. **Dead vampires** do not roll dice. It is possible, however, for vampires to be resurrected, so death does not mean permanent ejection from the game.
- The vampire who has the *lowest* roll during the round loses a life as a result of being forced to ingest garlic.
  - Be careful—if there are two or more vampires who get the same lowest roll, it is the *first* unlucky vampire who must eat the garlic.
- Rolling two sixes results in special activity: the vampires on the immediate left and right of the vampire who rolled the sixes *resurrect* or *sparkle*.
  - A vampire is resurrected if they were previously dead, meaning they had zero lives. Since each vampire rolls the dice following the sequence of the circle, it is possible for a vampire to be resurrected after their turn to roll the dice. **They do not roll the dice in this event.**
  - A vampire sparkles if they are still alive.

  In either case, the neighboring vampires will gain 1 life. Nothing of note occurs for the vampire who rolled the sixes.

The Garlic Game ends when there is only one vampire left with any lives. The remaining vampire is congratulated for being the ~~lone survivor~~ crowned victor of the Garlic Game.

3 Your Task

Your job will be to simulate the Garlic Game. We will give you a file called `names.h` that you *must* use. It contains the special dice roll names as well as the names of the vampires that are participating in the Garlic Game. The vampires are named as follows:

```
const char *names[10] = {
  "Alec",  "Bree",    "Carmen", "Demetri", "Edward",
  "Felix", "Garrett", "Heidi",  "Irina",   "Jane"
};
```

© 2021 Darrell Long

The sequence in which the vampires roll follows the order in which the names are presented: Alec rolls first, followed by Bree, followed by Carmen, and so on and so forth.

3.1 Rolling the Dice

You will need to use a *pseudorandom number generator* in order to simulate the dice rolls in your implementation of the Garlic Game. You will utilize the `random()` function included in the `<stdlib.h>` library. To simulate rolling a die, you should call `random()` and limit the returned value to the range 0–5 inclusive (0-indexing is used in Computer Science and by vampires). The modulo operator will be handy here.

In order to make your program *reproducible*, you must specify the starting point for the pseudorandom number sequence that will be generated. This is accomplished through *setting the random seed* using `srandom()`, also included in `<stdlib.h>`. The following is an example of setting the random seed and generating a pseudorandom number.

```
1 srandom(1337);     // Sets the random seed to 1337.
2 int r = random(); // The returned value is stored in r.
```

In an effort to make things a tad bit more interesting before most likely meeting their doom, the vampires have agreed to name their dice rolls. The names of the possible dice rolls are stored in a 6×6 matrix, a 2-D array of strings. Due to its size, the matrix of dice roll names is not shown in this document, but will be provided to you in `names.h`. The name of the matrix is `rolls`. To print out the name of each vampire's roll during the Garlic Game, simply index into the matrix using the first roll as the row index and the second roll as the column index (**C** uses *row-major* order).

```
1 int first = ... // First roll.
2 int second = ... // Second roll.
3 printf("Rolled %s\n", rolls[first][second]);
```

Again, the matrix of dice roll names, as well as the names of the vampires playing the Garlic Game, will be provided for you in the file `names.h`. Do not cut and paste from this document. You must include the file in your source code like so:

© 2021 Darrell Long

vampire.c

```c
#include "names.h"

int main(void) {
    // Your implementation goes here.
    // You may use additional functions if you so choose.
    return 0;
}
```

3.2 User Input

Your program, when run, should prompt the user for the number of vampires playing the Garlic Game and the random seed to set. You will want to use the `scanf()` function included in `<stdio.h>`. It is designed to read input according to a format specification. It is used much like `printf()` and its variants. Here is an example of scanning user input, commonly referred to as `stdin` (standard input), for an `int`.

```c
int input = 0;
scanf("%d", &input);
printf("User input: %d\n", input);
```

For additional details regarding `scanf()` usage, see the `man` page:

```
$ man scanf
```

Here is how your program should prompt for the number of vampires playing, which is followed by the prompt for the random seed (note that the user input is included in the game example):

```
$ ./vampire
Number of players: 3
Random seed: 2021
Round 1
 - Alec rolls Hard Six...
 - Bree rolls Seven Out...
 - Carmen rolls Yo-leven...
Alec is forced to eat garlic!
Alec has 2 lives remaining.
Round 2
 - Alec rolls Ace Deuce...
 - Bree rolls Hard Six...
 - Carmen rolls Easy Six...
Alec is forced to eat garlic!
Alec has 1 life remaining.
```

© 2021 Darrell Long

```
Round 3
 - Alec rolls Easy Eight...
 - Bree rolls Fever Five...
 - Carmen rolls Easy Eight...
Bree is forced to eat garlic!
Bree has 2 lives remaining.
Round 4
 - Alec rolls Fever Five...
 - Bree rolls Seven Out...
 - Carmen rolls Ace Deuce...
Carmen is forced to eat garlic!
Carmen has 2 lives remaining.
Round 5
 - Alec rolls Hard Eight...
 - Bree rolls Easy Four...
 - Carmen rolls Hard Eight...
Bree is forced to eat garlic!
Bree has 1 life remaining.
Round 6
 - Alec rolls Easy Six...
 - Bree rolls Seven Out...
 - Carmen rolls Nina...
Alec is forced to eat garlic!
Alec has died.
Round 7
 - Bree rolls Seven Out...
 - Carmen rolls Easy Ten...
Bree is forced to eat garlic!
Bree has died.
Carmen wins the Garlic Game!
```

It is imperative that your program handle invalid user inputs as they are one of the biggest sources of security holes in software. Your program should check if the user inputs an invalid number of players or random seed and print out an error message accordingly.

```
$ ./vampire
Number of players: -1
Invalid number of players.
```

```
$ ./vampire
Number of players: 11
Invalid number of players.
```

```
$ ./vampire
Number of players: asdf
Invalid number of players.
```

```
$ ./vampire
Number of players: 3
Random seed: -1
Invalid random seed.
```

```
$ ./vampire
Number of players: 3
Random seed: 1231920412948
Invalid random seed.
```

```
$ ./vampire
Number of players: 3
Random seed: asdf
Invalid random seed.
```

Error messages *must* be printed to `stderr` (*standard error*) instead of `stdout` (*standard output*). You must use `fprintf()` to print your error messages. An example of printing to `stderr`:

```c
fprintf(stderr, "Printing to stderr.\n");
```

3.3 Midnight

It is not commonly known that Samuel Hahnemann, the originator of *homeopathy*, did not actually die. In fact, his tomb in Père Lachaise is empty. After being bitten by a vampire during his return home from dining at *Au Clairon des Chasseurs*, Hahnemann turned into one as well. From that time on, he focused on curing the ills of his vampire kin. The fruits of his labors? A *homeopathic concoction* capable of even raising vampires from the dead.

When a vampire rolls two sixes, a *Midnight*, it is overcome with compassion for its comrades participating in the Garlic Game. The vampire administers Hahnemann's homeopathic concoction to the vampires immediately to the *left* and *right* of the vampire who rolled it. When quaffed, the concoction *resurrects* vampires who are dead and *sparkles* vampires who are still alive, strengthening them. The vampires are standing in a *circle* during the Garlic Game and are numbered from 0 to $n-1$. Vampire 4 has vampire 3 on its left and vampire 5 on its right. What about vampire $n-1$? It has vampire $n-2$ on its left and vampire $0 = n$ on its right. Here is some code to help calculate the indices of the

vampires to the left and right:

```c
//
// Returns the position of the player to the left.
//
// pos: The position of the current player.
// players: The number of players in the game.
//
uint32_t left(uint32_t pos, uint32_t players) {
  return (pos + players - 1) % players;
}

//
// Returns the position of the player to the right.
//
// pos: The position of the current player.
// players: The number of players in the game.
//
uint32_t right(uint32_t pos, uint32_t players) {
  return (pos + 1) % players;
}
```

Be careful when writing your code to check if a vampire rolled a Midnight. If rolling a die returns the values 0–5 inclusive, then the maximal sum when rolling two dice (a Midnight) would be 10. Why not 12? Vampires, like computer scientists, like to count from *zero*.

4 Deliverables

> *Thinking doesn't guarantee that we won't make mistakes. But not thinking guarantees that we will.*
>
> —Leslie Lamport

You will need to turn in:

1. `vampire.c`: This file will contain your implementation of the Garlic Game. The output of your program must match that of the reference program's in order to receive full credit.
2. `names.h`: This file will be supplied for you and contains the naming of the vampires and dice rolls. Do not change this file.
3. `Makefile`: This is a file that will allow the grader to type `make` to compile your program. Running `make` or `make all` must build your program. Running `make clean` must remove any compiler-generated files. As this is your first assignment, a working `Makefile` will be supplied for you.
4. `README.md`: This must be in *Markdown*. This must describe how to build and run your program.

© 2021 Darrell Long

5. `DESIGN.pdf`: This *must* be a PDF. The design document should describe the purpose of your program and communicate its overall design 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. **C** code is **not** considered pseudocode. You *must* push `DESIGN.pdf` before you push *any* code.

5 Submission

> *Calvin: Hocus-pocus abracadabra! I command my homework to do itself! Homework, be done! Rats.*
>
> Bill Watterson, *Calvin and Hobbes*

To submit your assignment through `git`, refer to the steps shown in `asgn0` Remember: *add, commit,* and *push*! Your assignment is turned in *only* after you have pushed. If you forget to push, you have not turned in your assignment and you will get a *zero*. "I forgot to push" is not a valid excuse. It is *highly* recommended to commit and push your changes *often*.

We will provide you with an *Ubuntu 20.04* binary of our implementation. Your code should produce *exactly* the same output for *all* inputs in order to receive full credit.

6 Supplemental Readings

> *The more that you read, the more things you will know. The more that you learn, the more places you'll go.*
>
> —Dr. Seuss

- *The C Programming Language* by Kernighan & Ritchie
  - Chapter 3 §3.4-3.7
  - Chapter 4 §4.1 & 4.2
  - Chapter 7 §7.2
  - Appendix B §B4
- *The Kollected Kode Vicious* by George Neville-Neil
  - §1.16
  - §2.6

© 2021 Darrell Long