```
///-------------------------------------------------------------------------
/// Lotto Longshot - a Lotto 6/49 simulator showing the futility of lotteries
/// Created By: David Young, December 2022
/// This requires crates:
///    scan-rules="^0.1"
///    rand = "0.8.5"
///    chrono = "0.4"
///    thousands = "0.2.0"
///-------------------------------------------------------------------------

// Modules used
#[macro_use] extern crate scan_rules;
use rand::Rng;
use std::time::Instant;
use chrono::prelude::*;
use thousands::Separable;

// global constants
const MIN_BALL: usize = 1;  // lowest number you can pick
const MAX_BALL: usize = 49;  // highest number you can pick
const MAX_PICKS: usize = 6;  // maximum number of numbers you can pick
const RESULT_SCENARIOS: usize = 7;  // can score from 0 - 6 right = 7 scenarios
const PAYOFF_RATES: [u64; 7] = [0, 0, 3, 10, 80, 2_500,
                                9_000_000];  // payoff per balls
const COST_PER_TICKET: u64 = 3;  // like it says

struct Simulation {  // Structure to hold all variables for a simulation run
    quick_picks_choice: char,  // y or other
    start_instant: Instant,  // for calculating runtime
    finish_instant: Instant, // as above
    runtime_seconds: f64,  // as above
    my_picks_idx: [usize; MAX_PICKS],  // which numbers user picked, 0-index
    num_games_to_run: u64,  // how many lottery games to simulate
    count_results: [u64; RESULT_SCENARIOS]  // tally each result, e.g. guessed 0..6
}

```

```
37  fn main() {
38      // Main program...duh.
39      // initialize global variables for a simulation run
40      // note that arrays are zero-indexed, so ball label would be index + 1
41      let now = Instant::now();
42      let mut sim_run = Simulation {
43          quick_picks_choice: 'y',
44          start_instant: now,
45          finish_instant: now,
46          runtime_seconds: 0.0,
47          my_picks_idx: [0; MAX_PICKS],
48          num_games_to_run: 0,
49          count_results: [0; RESULT_SCENARIOS]
50      };
51
52      // run the program
53      get_user_input(&mut sim_run);  // get simulation paramters from user via standard
    input
54      run_simulation(&mut sim_run);  // run all the games
55      report_results(&mut sim_run);  // report on the total wins by type and earnings
56
57  }  // main
58
59  fn get_user_input(this_run: &mut Simulation) {
60      // Get user's paramters for the simulation from standard input
61      println!("\nWelcome to Lotto Longshot - a lesson in futility!");
62      println!("--------------------------------------------------");
63      println!("\nThis simulates a Lotto 6/49 lottery to see how lucky you are (not).");
64      println!("\nType 'y' + Enter for a random quick pick, else any other letter +
    Enter");
65      let user_choice: char;
66      let mut picks: [usize; MAX_PICKS];
67      let mut valid: bool;
68      loop {
69          let result = try_readln! {
70              (let c: char) => (c)
```

```
71          };
72          match result {
73              Ok(c) => {
74                  user_choice = c;
75                  break;
76              },
77              Err(_) => {
78                  println!("Type a single character and press enter ");
79                  continue;
80              }
81          }
82      } // loop
83      if user_choice == 'y' {
84          picks = do_quick_pick();
85      }
86      else {
87          loop {
88              (picks, valid) = get_user_picks();
89              if valid {
90                  break;
91              }
92          }
93      }
94      picks.sort();  // sort the chosen balls
95      this_run.quick_picks_choice = user_choice;
96      this_run.my_picks_idx = picks;
97
98      println!("Game details will be shown for up to 100 simulations.");
99      println!("How many games do you want to simulate?");
100     loop {
101         let result = try_readln! {
102             (let n: u64) => (n)
103         };
104         match result {
105             Ok(n) => {
106                 this_run.num_games_to_run = n;
```

```
107                 break;
108             },
109             Err(_) => {
110                 println!("Type a positive integer number and press enter, butthead");
111                 continue;
112             }
113         }
114     }  // loop
115 }  // get_user_input
116 
117 fn do_quick_pick() -> [usize; MAX_PICKS] {
118     // Get and return random ball picks rather than letting user choose
119     let mut picks: [usize; MAX_PICKS] = [0; MAX_PICKS];
120     // use the game's ball-drawing to get a random set
121     let balls_array: [bool; MAX_BALL] = draw_balls();
122     let mut num_picked: usize = 0;
123     let mut n: usize = 0;
124     while num_picked < MAX_PICKS {
125         if balls_array [n] {
126             picks [num_picked] = n;
127             num_picked += 1;
128         }
129         n += 1;
130     }
131     return picks;
132 }  // do_quick_pick
133 
134 fn draw_balls() -> [bool; MAX_BALL] {
135     // Simulate a lottery game draw; return array of booleans
136     // representing which balls were pulled (true's) from all possible values
137     let mut balls_array: [bool; MAX_BALL] = [false; MAX_BALL];
138     let mut nballs_picked: usize = 0;
139     let mut test_ball: usize;
140     while nballs_picked < MAX_PICKS {  //i.e, from index 0 to MAX_PCKS - 1
141         test_ball = rand::thread_rng().gen_range(0..MAX_BALL);  // to max_ball - 1
142         if !balls_array [test_ball] {  // if this ball hasn't already been generated
```

```
143             balls_array [test_ball] = true;
144             nballs_picked += 1;
145         }
146     }
147     return balls_array;
148 }  // draw_balls
149 
150 fn get_user_picks() -> ([usize; MAX_PICKS], bool) {
151     // Get user's choice of balls, check validity, and return
152     // an array of the balls picked (if valid) plus a 'valid' boolean.
153     let mut picks: [usize; MAX_PICKS] = [0; MAX_PICKS];
154     println!("Enter {} numbers from {} to {}", MAX_PICKS, MIN_BALL, MAX_BALL);
155     loop {
156         let result = try_readln! {  // ugly hardcoding but readln! doesn't do arrays
157             (let n0: usize, let n1: usize, let n2: usize, let n3: usize,
158             let n4: usize, let n5: usize) => (n0, n1, n2, n3, n4, n5)
159         };
160         match result {
161             Ok((n0, n1, n2, n3, n4, n5)) => {
162                 picks[0] = n0;  // will later change pick to index of picks
163                 picks[1] = n1;
164                 picks[2] = n2;
165                 picks[3] = n3;
166                 picks[4] = n4;
167                 picks[5] = n5;
168                 break;
169             },
170             Err(_) => {
171                 println!("Enter {} numbers from {} to {}", MAX_PICKS, MIN_BALL,
    MAX_BALL);
172                 continue;
173             }
174         }  // match
175     }  // loop
176     // check for errors
177     for n in 0..MAX_PICKS {  // loop from 0 to (MAX_PICKS - 1)
```

```
178         if (picks[n] < MIN_BALL) | (picks[n] > MAX_BALL) {
179             println!("You chose {} but numbers must be from {} to {}",
180                 picks[n], MIN_BALL, MAX_BALL);
181             return (picks, false);
182         };  // if
183         for m in 0..MAX_PICKS {  // loop from 0 to MAX_PICKS - 1
184             if (n != m) & (picks[n] == picks[m]) {
185                 println!("Duplicate numbers: {}", picks[n]);
186                 return (picks, false);
187             }  // if
188         }  // for
189     }  // for
190     // valid, so change the chosen numbers to index values
191     for n in 0..MAX_PICKS {  // change from ball label to index (ie, - 1)
192         picks[n] -= 1;
193     }
194     return (picks, true);
195 }  // get_user_picks
196
197 fn run_simulation(this_run: &mut Simulation) {
198     // Simulate all the lottery games, and accumulate statistics
199     let nowx = Local::now();
200     let show_date_time = nowx.format("%Y-%m-%d %H:%M:%S");  // Printable date / time
201     let mut big_number_str : String = this_run.num_games_to_run.separate_with_commas();
202     println!("Running simulation for {} games at {}...",
203         big_number_str, show_date_time);
204     // Create a displayable set of picked balls
205     let mut picks_display = this_run.my_picks_idx;
206     for n in 0..MAX_PICKS {
207         picks_display[n] += 1;
208     }
209     println!("Numbers chosen : {:?}", picks_display);
210     this_run.start_instant = Instant::now();
211     this_run.count_results = [0; RESULT_SCENARIOS];
212     let mut num_right: usize;
213     let mut balls_array: [bool; MAX_BALL];
```

```
214      // Run the simulation x times
215      for g in 1..=this_run.num_games_to_run {
216          balls_array = draw_balls();
217          num_right = 0;
218          for n in 0..MAX_PICKS {  // loop from 0 to (MAX_PICKS - 1)
219              if balls_array[this_run.my_picks_idx[n]] {
220                  num_right += 1;
221              }  // if
222          } // for
223          this_run.count_results[num_right] += 1;
224          if this_run.num_games_to_run <= 100 {  // show details for small runs
225              print!("Game # {:3} : ", g);
226              for i in (MIN_BALL - 1)..MAX_BALL {
227                  if balls_array[i] {
228                      print!(" {} ", (i + 1));
229                  }  // if
230              }  // for
231              println!("  You got {} right", num_right);
232          }  // if
233          if g % 1_000_000 == 0 {  // print every xxx games as progress indicator
234              big_number_str = g.separate_with_commas();
235              println!("Running Game {}...", big_number_str);
236          } // if
237      } // for
238  }  // run_simulation
239
240  fn report_results(this_run: &mut Simulation) {
241      // Print a summary of the overall simulation results
242      let nowx = Local::now();
243      let show_date_time = nowx.format("%Y-%m-%d %H:%M:%S");  // Printable date / time
244      let mut big_number_str : String = this_run.num_games_to_run.separate_with_commas();
245      println!("Finished simulation for {} games at {}...",
246          big_number_str, show_date_time);
247      let now2 = Instant::now();
248      let run_time = now2.duration_since(this_run.start_instant);
249      let run_seconds: f64 = (run_time.as_micros() as f64) / 1000000.0f64;
```

```
250      let runs_per_second : f64 = this_run.num_games_to_run as f64 / run_seconds;
251      big_number_str = (runs_per_second as u64).separate_with_commas();
252      println!("Run time = {} seconds", run_seconds);
253      println!("Runs per second = {}\n", big_number_str);
254      this_run.finish_instant = now2;
255      this_run.runtime_seconds = run_seconds;
256      // Print how many games resulted in which outcomes, and accumulate totals
257      let mut total_payoff : u64 = 0;
258      let mut this_payoff : u64;
259      for n in 0..RESULT_SCENARIOS { // from 0 to scenarios - 1
260          this_payoff = this_run.count_results[n] * PAYOFF_RATES[n];
261          total_payoff += this_payoff;
262          big_number_str = this_run.count_results[n].separate_with_commas();
263          print!("You picked {} correct {} times", n, big_number_str);
264          big_number_str = this_payoff.separate_with_commas();
265          println!("  --> Payoff = ${}", big_number_str);
266      } // for
267      // Print the overall totals
268      let total_cost : u64 = this_run.num_games_to_run * COST_PER_TICKET;
269      let total_profit : i64 = (total_payoff as i64) - (total_cost as i64);
270      big_number_str = total_cost.separate_with_commas();
271      println!("\nTotal cost of tickets : ${}", big_number_str);
272      big_number_str = total_payoff.separate_with_commas();
273      println!("Total money won : ${}", big_number_str);
274      big_number_str = total_profit.separate_with_commas();
275      println!("Total profit / loss : ${}", big_number_str);
276      let profit_pct : f64 = ((total_profit as f64) / (total_cost as f64)) * 100.0f64;
277      println!("Percent profit / loss : {:.2} %", profit_pct);
278      if profit_pct < 0.0 {
279          println!("*** Loser!!! I hope you learned something from this! ***");
280      }
281      else {
282          println!("*** Winner!!! Pure fluke though, don't make this a habit ***");
283      }
284      println!("\n*************** END SIMULATION ***************\n");
285  }  // report_results
```