Tommy Hollenberg
CS161 Assign5
Design Document
02/24/2017

Understanding the Problem

The purpose of this program is to simulate a game of bowling. The program will "bowl" for the players by randomly generating numbers to determine how many pins they knock down. In every other aspect, however, this program will be identical to the programs used at bowling alleys to create games, customize the number of players and player names, track results, and keep score.

The program must accept user input for how many players are participating, followed by the name of each player. For each player, the result of every bowl must be stored, as well as the number of points scored in each frame, and a running total must be kept updated throughout the game. This information will be used to create and maintain a scoreboard, which must be updated and printed to the console after every bowl. When the game is finished, a winner must be declared and the user must be asked whether they would like to play again.

One point is awarded for every pin knocked down in each frame. Additionally, if a strike is bowled (all 10 pins knocked down with one ball, denoted by 'X' on the scoreboard), the number of pins knocked down with the next two balls is added to the score of the frame in which the strike was bowled. If a spare is bowled (all 10 pins knocked down with two balls, denoted by '/' on the scoreboard), the number of pins knocked down the next ball is added to the score of the frame in which the spare was bowled. If a strike or spare is bowled on the final (10th) frame, the additional balls needed to settle the score are bowled, meaning a total of three balls are bowled in the 10th frame if a strike or a spare is bowled initially. Knocking down 0 pins with a roll is called a gutter ball, denoted by '-' on the scoreboard.

Devising a Plan

A user-defined **Player** object will be used to hold player information such as player name, bowling outcomes, score by frame, and total score. Member variables of the Player object will include:

- A C-style string (character array) of length 64 to hold the player's name (people with longer names will have to use a nickname).
- A character array of length 21 to hold the outcomes of the up to 21 balls bowled by each player. Every element of this array will be initialized to 0, and only non-zero elements will be printed on the scoreboard. This will provide an easy way to keep track of any balls that weren't bowled (i.e. the second ball of the frame when a strike is bowled with the first ball).

- An integer array of length 10 to hold the scores of each frame. This will facilitate updating scores of frames in which a strike or spare is bowled once additional balls have been rolled.
- An integer variable to hold the player's total score. The maximum score possible in bowling is 300, so 32 bits will be more than enough.

In the **main** function, the program will declare variables, then call a series of functions within a while loop to allow the user to play multiple games of bowling. A Player pointer will be created in main and passed by address to a **game_setup** function, where the user will enter the number of players. A Player array of length equal to the number of players entered by the user will be dynamically allocated on the heap and assigned to the Player pointer created in main (by dereferencing the pointer to a Player pointer parameter). Then, the user will be asked to enter names for all of the players, which will be assigned to the name strings of the Player objects in the array using the cin.getline() function. Finally, game_setup will return an integer value equal to the number of players. Next, the **lets_go_bowling** function will be called from main and passed the Player pointer to the dynamically allocated Player array and the integer number of players. The lets_go_bowling function will use a for-loop to conduct each of the 10 bowling frames. In each frame, a for-loop will be used to cycle through all of the players, calling the **bowl_frame** for each. After all 10 frames have been bowled by all players, the lets_go_bowling function will call the **declare_winner** function before returning to main. Back in main, the memory on the heap occupied by the Player array will be deallocated using the delete[] operator and the Player pointer will be set to 0. Then the user will be asked if they would like to play again, in which case the function will loop back and call game_setup again. Otherwise, the program ends.

The bowl_frame function calls the **bowl**, **update_score**, and **print_scoreboard** functions once, and if the player did not get a strike it calls them all a second time. Finally, if the it is the FINAL_FRAME (a preprocessor #define macro equal to 9, because the 10 frames span from 0 to 9) and the player bowls a strike or a spare, the **fill_balls** function is called to handle the additional bowls in the 10th frame.

The bowl function is passed the bowler's name, the memory address where the result character should be stored, the number of pins standing before the player bowls, and a Boolean value signifying whether it is a new frame or not. First the **prompt_bowler** function is called, which simply prompts the bowler to press enter to bowl, and eats the newline character with cin.getline(). Then an integer variable is declared and assigned a randomly generated number from 0 to the number of pins standing, which represents how many pins they knock down. If all of the pins are knocked down, the player gets a strike if this is the first bowl of the frame, and a spare otherwise. If the player knocks down 0 pins, they get a gutter ball. The number of pins knocked down is returned to the caller.

The update_score function is passed the frame number, a pointer to the Player object of the current bowler, the number of pins knocked down, and how many balls have been bowled in the current frame. The number of pins knocked down are added to the frame score of the current frame as well as the total score. If this is the first frame, the function returns to the caller. Otherwise, if this is the first ball of the current frame, if the frame score of the previous frame is 10 (indicating that a strike or a spare was bowled), the number of pins knocked down is added to the frame score of the previous frame and the total score (again). If there are at least two previous frames, and the last two frames before this one were strikes, then the pins knocked down are added to the frame score of the frame before the previous frame and the total score (again). If instead this is the second ball of the current frame, if the previous frame was a strike, then the number of pins knocked down is added to the frame score of the previous frame and the total score (again).

The print_scoreboard function cycles through all of the players, print their name and calls the **print_bowls** and **print_scores** functions for each. The print_bowls function cycles through all of the player's bowl results, printing the result character or, if the character is equal to 0, a space (this ensures that results for bowls that have not occurred yet are not printed). The print_scores function cycles through all of the player's frame scores and, if the frame score and bowl result character for the first bowl of that frame are not both zero, prints the frame score (a space is printed otherwise, because this means that the frame has not been bowled yet). Finally it also prints the player's total score.

The fill_balls function is passed a pointer to the Player object of the bowler, a Boolean value indicating whether the player got a strike or a spare with their initial 10th frame. If the player got a strike, the bowl, update_score, and print_scoreboard functions are each called twice. In between, if the player got another strike with their first bowl, the number of pins will be reset to 10 for the second bowl. If the player got a spare initially, the bowl, update_score, and print_scoreboard functions are called only once.

In the declare_winner function, all of the players are cycled through with a for-loop, and the highest score encountered as well as the name of the player who achieved the score are kept track of with an integer and C-style string. A Boolean variable indicating whether there is a tie will also be created. When a new score is equal to the previous seen max score, this variable will be set to true. When a new score is greater than the previous max score and becomes the new max score, this variable is reset to false. Afterwards, if there is a tie, this is stated. Otherwise, the winner is congratulated.

Pseudocode:

```
int main() {
    Declare integer to hold number of players and Player pointer to
      point to the dynamic Player array allocated in game_setup
    Seed the random number generator with the system clock
    Do:
        Call game_setup, passing the address of the Player pointer and
          assigning the return value to the player number integer
        Call lets_go_bowling, passing the number of players and the
          Player pointer
        Free the memory on the heap occupied by the Player array and
          set the Player pointer to 0
    Loop while the user affirmatively responds to the prompt, "Would
      you like to play again?"
}

int game_setup(pointer to the Player pointer in main) {
    Declare integer to hold number of players and assign it a user
      input positive integer
    Dynamically allocate a Player object array on the heap of length
      equal to the number of players and assign the returned address
      to the Player pointer in main
    For each player
        Prompt user for the player's name, accept input for the name
          member variable of the Player object using cin.getline()
    Return the number of players
}

void lets_go_bowling(number of players, pointer to Player array) {
    For frames 1 to 10
        For each player
            Call bowl_frame, passing the frame number and the address
              of the Player object associated with the current player
    Call declare_winner, passing the number of players and the pointer
      to the Player array
}
```

```
void bowl_frame(frame number, pointer to Player object of bowler,
  number of players, pointer to Player array) {
    Declare integer for number of pins left and initialize to 10
    Declare integer for pins knocked down and assign it the return
      value of a call to the bowl function)
    Call update_score
    Call print_scoreboard
    Subtract the pins knocked down from the pins left
    If the number of pins left is not zero
        Call bowl again and assign the return value to the number of
          pins knocked down
        Call update_score
        Call print_scoreboard
    If this is the final frame and the current frame score is 10 (thus
      a strike or a spare was bowled)
        Call fill_balls, passing true if the first bowl result of the
          frame is ‘X’, false otherwise
}

int bowl(bowler’s name, memory address to store bowl result, pins
  left, Boolean value indicating whether this is a new frame) {
    Call prompt_bowler
    Declare an integer for the pins knocked down and assign the result
      of a call to rand() % (pins left + 1)
    If pins knocked down is equal to pins left
        If it is a new frame
            Bowl result is ‘X’
            Output “You got a strike!”
        Otherwise
            Bowl result is ‘/’
            Output “You got a spare!”
    Else if pins knocked down is zero
        Bowl result is ‘-’
        Output “You got a gutter ball...”
    Else
        Bowl result is pins knocked down + ‘0’
        Output “You knocked down ” number of pins knocked down
    Return number of pins knocked down
}
```

```
void update_score(frame number, pointer to Player object of bowler,
  pins knocked down, ball number in current frame) {
    Add pins knocked down to frame score and total score
    If this is the first frame, return to caller
    If this is the first ball of the frame
        If the previous frame score is 10
            Add pins knocked down to previous frame score and total
              score
        If the frame number is greater than 1 (zero-based) and the
          first bowl results of the previous two frames are both 'X'
            Add pins knocked down to the frame score before the
              previous frame and total score
    If this is the second ball of the frame and the first bowl result
      of the previous frame is 'X'
        Add pins knocked down to the previous frame score and total
          score
}

void print_scoreboard(number of players, pointer to Player array) {
    Output scoreboard column headings
    For each player
        Print player's name
        Call print_bowls
        Call print_scores
}

void print_bowls(pointer to Player object of current player) {
    For each frame
        Print first bowl result of frame or a space if it is 0
        Print second bowl result of frame or a space if it is 0
        If this is the final frame
            Print the third bowl result or a space if it is 0
}

void print_scores(pointer to Player object of current player) {
    Print total score, then print newline character
    For each frame
        If the frame score and the first bowl result are both 0
            Print ' '
        Otherwise print frame score
}
```

```
void fill_balls(pointer to Player object of bowler, Boolean variable
  (true if strike, false if spare), number of players, pointer to
  Player array) {
    Declare integer for pins left and initialize to 10, declare
      integer for pins knocked down
    If strike
        Assign the return value of bowl function call to pins knocked
          down
        Call update_score, passing 2 to ball number
        Call print_scoreboard
        Subtract pins knocked down from pins left
        If pins left isn't 0
            Call bowl again, passing false to the new frame Boolean
        Otherwise
            Reset pins left to 10
            Call bowl again, passing true to the new frame Boolean
        Call update_score, passing 3 to ball number
    Otherwise (if spare)
        Call bowl function
        Call update_score, passing 3 to ball number
    Call print_scoreboard
}

void declare_winner(number of players, pointer to Player array) {
    Declare 256 character C-style string to hold winner's name
    Declare integer to hold max score and initialize to -1
    Declare Boolean to indicate whether there is a tie and initialize
      to false
    For each player
        If total score is equal to max score
            Set tie to true
        Otherwise if total score is greater than max score
            Set winner name to the player's name
            Set max score to player's total score
            Set tie to false
    If tie is true
        Output "It was a tie!"
    Otherwise
        Output winner name " won the game!"
}
```

Testing Plan

Testing scoring:

Bowl results of first 3 frames:	**Frame scores (and total score):**
1 2 3 4 7 -	3 7 7 (17)
X 1 3 4 1	14 4 5 (23)
3 / 4 / X	14 20 10 (44)

Other things to test for:

The program should ask the players to bowl in the correct order and assign the bowl result to the correct player in the correct frame. When printing the scoreboard, the program should print spaces rather than scores for bowl results and frame scores of frames that have not been bowled yet, as well as for the second bowl result of a frame in which the first bowl was a strike. The additional balls in the 10th frame, if necessary, should be printed and scored correctly. The winner should be correctly identified and congratulated, unless there is a tie, in which case the tie should be correctly identified.