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

int* arrNum;        // array of the random numbers of the threads
int* arrWins = NULL;       // array of wins/losses
int ready = 0;      // next round indicator

void *runner(void *param);

int main(int argc, char* argv[])    // first arg is num of threads, second arg is
num of rounds
{
    // variables
    int size = atoi(argv[1]);
    int gameOver = atoi(argv[2]);
    time_t t;
    pthread_t *tid;
    pthread_attr_t attr;

    pthread_attr_init(&attr);

    // initialize srand
    srand((unsigned) time(&t));

    // allocate space
    tid = malloc(sizeof(pthread_t)*size);
    arrNum = (int*)malloc(size * sizeof(int));
    arrWins = (int*)malloc(size * sizeof(int));

    for(int i = 0; i < size; i++)
    {
        // initialize arrays
        arrNum[i] = -1;
        arrWins[i] = 0;
    }

    for(int i = 0; i < size; i++)
    {
        // create thread, pass num to each thread to know where to put generated
numbers
        int *index = (int *)malloc(sizeof(int));
        *index = i;
        pthread_create(&tid[i], &attr, runner, (void*)index);

    }


    // loop for each round
    int round = 0;
    while(round != gameOver){
        // change ready to have threads send in numbers
        ready = 1;
        round++;
        printf(" ------- \n Round %d \n ------- \n", round);

        // wait for threads to send in numbers - they will because they will see
ready has changed
```

```
    for(int i = 0; i < size; i++)
    {
        while(arrNum[i] == -1)
        {
            if(arrNum[size-1] != -1)
                break;
        }
    }

    // calculate which threads score
    int max = 0;
    for(int i = 0; i < size; i++)
    {
        if(arrNum[i] > max)
        {
            max = arrNum[i];
        }
        printf("arrNum[%d]: %d \n", i, arrNum[i]);
    }
    for(int i = 0; i < size; i++)
    {
        // update score & print which threads score
        if(arrNum[i] == max)
        {
            arrWins[i] = arrWins[i] + 1;
            printf("Thread #%d Scored \n", i);
        }
    }
    ready = 0;
}

// print total score for each thread
printf("\n");
for(int i = 0; i < size; i++)
{
    printf("Thread #%d Total Score: %d\n", i, arrWins[i]);
}

printf("\nOverall Winner(s): \n");

// update arrWins which threads win or lose
int winner = 0;
int winIndex = 0;
for(int i = 0; i < size; i++)
{
    if(arrWins[i] > winner)
    {
        winner = arrWins[i];
        winIndex = i;
    }
}

for(int i = 0; i < size; i++)
{
    if(arrWins[i] == winner)
    {
        printf(" - Thread #%d \n", i);
    }
}
```

```
    // update ready for game over
    ready = 2;

    // wait for all threads to finish (pthread_join())
    for(int i = 0; i < size; i++)
        pthread_join(tid[i], NULL);

    // print program complete
    printf("Game over. Program finished. \n");

    // free the memory after program has ended
    free(arrNum);
    free(arrWins);
}

void *runner(void *param)
{
    // convert param to int
    int x = *((int*) param);
    int random = rand() % 100;

    // into loop and check ready variable
    if (ready == 1)
    {
        // if ready = next round, select random number between 0-99 and put in
array
        // print thread # and the number generated
        arrNum[x] = random;
        printf("Thread #%d --> Random Number: %d \n", x, random);
    }
    else if (ready == 2){
        // else if game over, check win or loss
            // print thread # and win/loss
        printf("Thread #%d Wins: %d \n \t  Losses: %d \n", x, arrWins[x], ready -
arrWins[x]);
    }

    // exit thread
    pthread_exit(0);
}
```