```perl
#!/usr/bin/perl
# Nate McCain
# CS 390
# Program 6
# 11/20/2017

######################################################################
# A subroutine called by sort.
# Automatically gets two parameters,
# $a and $b. The comparison should
# return 0, 1, -1.

sub by_last_then_first
{
	my @personA;
	my @personB;

	# Parse each name into its two fields.
	@personA = split ':', $a;
	@personB = split ':', $b;

	# Perform the comparison of the last names.
	if (($personA[0] cmp $personB[0]) < 0)
	{
		return -1;
	}

	elsif (($personA[0] cmp $personB[0]) > 0)
	{
		return 1;
	}

	# If the last names are the same.
	else
	{
		# Compare the first names.
		if (($personA[1] cmp $personB[1]) < 0)
		{
			return -1;
		}
		elsif (($personA[1] cmp $personB[1]) > 0)
		{
			return 1;
		}
	}

	# The names are the same, return 0.
	return 0;
}
######################################################################
# A subroutine called by sort. It automatically gets
# two parameters, $a and $b. The comparison should
# return -1, 0, or 1.

sub by_avg
{
	my @personA;
	my @personB;

	# Parse each person into their respective arrays.
	@personA = split ':', $a;
	@personB = split ':', $b;

	# Return the comparison between batting averages.
	return ($personB[2] <=> $personA[2]);
}
######################################################################
```

```
my @allPlayers;
my @validvals;
my @finalAllPlayers;
my @finalValidVals;

# Try to open the file specified by the user's input. If no file exists,
# or if no input was given, exit the program.
open FILE1, $ARGV[0] or die "Could not open file, exiting!\n";

# While reading to the end of the file.
while(<FILE1>)
{
        # Split the current line using whitespace.
        @values = split(/ /,$_);

        # Count the number of arguments in the current line.
        $numOfArguments = @values;

        # Too many arguments are given on the line.
        if ($numOfArguments > 9)
        {
                # Create an error statement.
                $errorStatement = join(":",$values[1],$values[2],"Error, Too many
arguments.");

                # Add the error statement to the array of all players.
                push @allPlayers, $errorStatement;
        }

        # Not enough arguments are given on the line.
        elsif ($numOfArguments < 9)
        {
                # Create an error statement.
                $errorStatement = join(":",$values[1],$values[2],"Error, Not enough
arguments.");

                # Add the error statement to the array of all players.
                push @allPlayers, $errorStatement;
        }

        # Correct number of arguments given.
        else
        {
                # Check that the name given is valid.
                if (($values[0] =~ /^[a-zA-Z'.\-]+$/) && ($values[1] =~ /^[a-zA-Z'.\-]+$/))
                {
                        # Check that only numbers are given for the remaining arguments.
                        if (($values[2] =~ /^[0-9]+$/) && ($values[3] =~ /^[0-9]+$/) &&
($values[4] =~ /^[0-9]+$/) &&
                            ($values[5] =~ /^[0-9]+$/) && ($values[6] =~ /^[0-9]+$/) &&
($values[7] =~ /^[0-9]+$/) &&
                            ($values[8] =~ /^[0-9]+$/))
                        {
                                # Check to see if the values are possible, and that Plate
Appearances and At Bats don't start with a zero.
                                if (($values[2] =~ /^[1-9][0-9]*$/) && ($values[3] =~ /^[1-9]
[0-9]*$/) &&
                                    ($values[2] >= ($values[4] + $values[5] + $values[6] +
$values[7] + $values[8])) &&
                                    ($values[3] >= ($values[4] + $values[5] + $values[6] +
$values[7])))
                                {
                                        # Check to see if Plate Appearances are greater than
At Bats.

                                        if ($values[2] >= $values[3])
                                        {
                                                # The given line is valid. Calculate the
player's stats and print them.
```

```
                                                  $avg = ($values[4] + $values[5] + $values[6]
+ $values[7]) / $values[3];
                                                  $slg = ((1 * $values[4]) + (2 * $values[5])
+ (3 * $values[6]) + (4 * $values[7])) / $values[3];
                                                  $obp = ($values[4] + $values[5] + $values[6]
+ $values[7] + $values[8]) / $values[2];

                                                  # Output the stat line.
                                                  #printf ("%-12s, %-12s : %1.3f %1.3f %1.3f
\n",$values[1],$values[0],$avg,$slg,$obp);

                                                  # Put the values together.
                                                  $glue = join(":",$values[1],$values[0],$avg,
$slg,$obp);

                                                  # Put the new value into array for valid
values.
                                                  push @validvals, $glue;

                                                  # Put the new value into the array that
holds all players.
                                                  push @allPlayers, $glue;
                                          }

                                          # Plate Appearances must be greater than At Bats.
                                          else
                                          {
                                                  # Create an error statement.
                                                  $errorStatement = join(":",$values[1],
$values[2],"Error, Plate Appearances must be greater than or equal to At Bats.");

                                                  # Add the error statement to the array of
all players.
                                                  push @allPlayers, $errorStatement;
                                          }
                                  }

                                  # Either Plate Appearances and/or At Bats start with a zero,
or the stat line might be impossible.
                                  else
                                  {
                                          # Check to see if all the other stats are zero.
                                          if (($values[2] == 0) && ($values[3] == 0) &&
($values[4] == 0) && ($values[5] == 0) &&
                                              ($values[6] == 0) && ($values[7] == 0) &&
($values[8] == 0))
                                          {
                                                  # The given line is valid. Print out the
player's stats.
                                                  #printf ("%-12s, %-12s : %1.3f %1.3f %1.3f
\n",$values[1],$values[0],$values[2],$values[3],$values[4]);

                                                  # Put the values together.
                                                  $glue = join(":",$values[1],$values[0],$avg,
$slg,$obp);

                                                  # Put the new value into array for valid
values.
                                                  push @validvals, $glue;

                                                  # Put the new value into the array that
holds all players.
                                                  push @allPlayers, $glue;

                                          }

                                          # The stat line given is impossible.
                                          else
                                          {
                                                  # Create an error statement.
                                                  $errorStatement = join(":",$values[1],
$values[2],"Error, Too given stat line is impossible.");

                                                  # Add the error statement to the array of
all players.
                                                  push @allPlayers, $errorStatement;
                                          }
                                  }
                          }

                          # Valid numbers were not given for the remaining arguments.
                          else
                          {
                                  # Create an error statement.
                                  $errorStatement = join(":",$values[1],$values[2],"Error,
Valid numbers were not given.");

                                  # Add the error statement to the array that holds all
players.
                                  push @allPlayers, $errorStatement;
                          }
                  }

                  # Name given is not valid.
                  else
                  {
                          # Create an error statement.
                          $errorStatement = join(":",$values[1],$values[2],"Error, Not a valid
name.");

                          # Add the error statement to the array that holds all players.
                          push @allPlayers, $errorStatement;
                  }

        }
}
# Finished reading through the players list.

# Array that holds all of the player records (including those with errors)
# after sorting by last name and then first name.
@finalAllPlayers = sort by_last_then_first @allPlayers;

# Array that holds only player records without errors (for the extra credit)
# after sorting by batting average.
@finalValidVals = sort by_avg @validvals;


printf "---------------------- BEGIN STATISTICS REPORT ----------------------\n\n\n";

printf "LASTNAME    , FIRSTNAME       AVG   SLG   OBP \n\n";

# Go through each record in the array.
foreach my $lineAllPlayers(@finalAllPlayers)
{
        # Split the array up into parts.
        @vals = split(":",$lineAllPlayers);

        # Find the number of items in the current array.
        $numOfVals = @vals;

        # If there are 5 elements, then there are no errors with the current array.
        if ($numOfVals == 5)
        {
                # Output the stat line.
                printf ("%-12s, %-12s : %1.3f %1.3f %1.3f \n",$vals[0],$vals[1],$vals[2],
$vals[3],$vals[4]);
```

```
        }

        # There is an error with the current array.
        else
        {
                # Output the error message.
                printf ("%-12s, %-12s : %-20s\n",$vals[0],$vals[1],$vals[2]);
        }
}

printf "\n----------------------- END STATISTICS REPORT -----------------------\n\n";

# Extra credit code!!!!
printf "-------- BATTING AVERAGES REPORT --------\n\n\n";

printf "LASTNAME    , FIRSTNAME       AVG\n\n";

# Output the stats for the records that don't have any errors.
foreach my $lineValidVals(@finalValidVals)
{
        # Split the array up into parts.
        @goodvals = split(":",$lineValidVals);

        # Output the stat line.
        printf ("%-12s, %-12s : %1.3f \n",$goodvals[0],$goodvals[1],$goodvals[2]);
}

printf "\n--------- END STATISTICS REPORT ---------\n\n";
# End of extra credit code.

# Close the file.
close (FILE1);
```