OSS Project1

학과 컴퓨터공학과

학번 12181670

이름 이한진

1. 구현 결과

과제에서 요구하는 사항을 모두 구현하였습니다.

1. 실행시 출력

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

1. Get the data of Heung-Min Son's Current Club, Appearances, Goals, Assists in players.csv

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

1. Get the team data to enter a league position in teams.csv

텍스트, 대수학, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Get the Top-3 Attendance matches in matches.csv

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Get the team's league position and team's top scorer in

teams.csv & players.csv

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Get the modified format of date\_GMT in matches.csv

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

1. Get the data of the winning team by the largest difference

on home stadium in teams.csv & matches.csv

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Exit

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

1. 구현방법

각 기능을 함수 단위로 나눠서 구현하였습니다. 원본 코드와 함께 설명 첨부합니다.

#! /bin/bash

check\_arguments() {

if [ $# -ne 3 ]; then

echo “Three arguments are required”

exit 1

fi

for arg in “$@”

do

if [[ ! “$arg” =~ \.csv$ ]]; then

echo “Each argument must be a CSV file name.”

exit 1

fi

TEAMS=$1

PLAYERS=$2

MATCHES=$3

done

}

프로그램 호출과 함께 전달된 파라미터의 유효성을 확인하는 부분입니다. 파라미터 갯수와 파일 형식이 csv인지 확인합니다.

Get\_user\_input() {

echo “”

echo “[MENU]”

echo “1. Get the data of Heung-Min Son’s Current Club, Appearances, Goals, Assists in players.csv”

echo “2. Get the team data to enter a league position in teams.csv”

echo “3. Get the Top-3 Attendance matches in mateches.csv”

echo “4. Get the team’s league position and team’s top scorer in teams.csv & players.csv”

echo “5. Get the modified format of date\_GMT in matches.csv”

echo “6. Get the data of the winning team by the largest difference on home stadium in teams.csv & matches.csv”

echo “7. Exit”

read -p “Enter your CHOICE (1~7) : “ reply

}

프로그램을 실행하고 반복해서 사용자에게 어떤 명령을 호출할지 선택지를 보여주고 입력을 받는 부분입니다.

Query\_son\_data() {

read -p “Do you want to get the Heung-min Son’s data? (y/n): “ answer

if [ “$answer” = “y” ]; then

cat $PLAYERS | awk -F, ‘$1 == “Heung-Min Son” {printf “Team: %s, Appearance: %s, Goal: %s, Assist: %s\n”, $4, $6, $7, $8}’

fi

}

손흥민의 데이터를 보여주는 함수입니다. Awk를 사용하여 구현하였습니다.

Query\_team\_of\_ranking(){

read -p “What do you want to get the team data of league\_position[1~20]: “ answer

cat $TEAMS | awk -F, -v answer=”$answer” ‘$6==answer {printf “%s %s %f\n\n”,$6, $1, $2/($2+$3+$4)}’

}

입력받은 순위의 팀 데이터를 보여주는 함수입니다. Awk를 사용하여 구현하였습니다.

Query\_top\_attendance\_matches(){

read -p “Do you want to know Top-3 attendance data and average attendance? (y/n): “ answer

if [ “$answer” = “y” ]; then

echo “\*\*\*Top-3 Attendance Match\*\*\*”

echo “”

cat $MATCHES| tail -n +2 | sort -t ‘,’ -k 2 -n -r| head -n 3 | awk -F, ‘{printf “%s vs %s (%s)\n%s %s\n\n”, $3, $4, $1, $2, $7}’

fi

}

관중이 많은 경기를 보여주는 함수입니다. 데이터가 아닌 행을 제외하고 Sort를 통해 정렬한 뒤 awk를 사용하여 구현하였습니다.

Query\_team\_info(){

read -p “Do you want to get each team’s ranking and the highest-scoring player? (y/n): “ answer

if [ “$answer” = “y” ]; then

sorted\_teams=$(cat $TEAMS | tail -n +2 | sort -t ‘,’ -k 6 -n)

IFS=’,’

echo “$sorted\_teams” | while read -r team\_name wins draws losses points ranking other\_fields

do

echo “”

echo “$ranking $team\_name”

highest\_goals=0

top\_scorer=$(awk -F’,’ -v team=”$team\_name” ‘

$4 == team {

if ($7 > highest\_goals) {

highest\_goals=$7;

result=$1” “$7;

}

}

END { print result }

‘ $PLAYERS)

echo “$top\_scorer”

done

fi

}

각 팀을 랭킹순으로 최다 득점 선수와 함께 보여주는 함수입니다.

먼저 팀 파일에서 가져온 데이터를 랭킹 순서대로 정렬하였습니다. 그리고 각 팀 데이터에 대해 선수 파일을 조회하며, 최다 득점자 이름과 최다 득점자의 골 수를 변수로 저장하여 능가하는 선수가 있다면 갱신하도록 구현하였습니다.

이렇게 찾은 최다 득점자를 팀 데이터와 함께 출력하였습니다.

Query\_match\_date(){

read -p “Do you want to modify the format of date? (y/n): “ answer

if [ “$answer” = “y” ]; then

cat $MATCHES | tail -n +2 | head -n 10 | sed -E ‘

s/Jan/01/g;

s/Feb/02/g;

s/Mar/03/g;

s/Apr/04/g;

s/May/05/g;

s/Jun/06/g;

s/Jul/07/g;

s/Aug/08/g;

s/Sep/09/g;

s/Oct/10/g;

s/Nov/11/g;

s/Dec/12/g;

s/^(([^,]\*),.\*)$/\1/;

s/([0-9]{2}) ([0-9]{1,2}) ([0-9]{4}) – ([0-9]{1,2}:[0-9]{2}(am|pm))/\3\/\1\/\2 \4/’

fi

}

경기 일자의 형식을 바꿔서 출력해주는 함수입니다. Sed를 이용하여서 구현하였습니다. 날짜 중 월 형식의 경우 각 경우에 대해 하나하나 지정해서 바꿔주는 식으로, 나머지는 포멧을 지정해주는 식으로 구현하였습니다.

Query\_largest\_diff\_matches(){

teams=(

“Arsenal”

“Tottenham Hotspur”

“Manchester City”

“Leicester City”

“Crystal Palace”

“Everton”

“Burnley”

“Southampton”

“AFC Bournemouth”

“Manchester United”

“Liverpool”

“Chelsea”

“West Ham United”

“Watford”

“Newcastle United”

“Cardiff City”

“Fulham”

“Brighton & Hove Albion”

“Huddersfield Town”

“Wolverhampton Wanderers”

)

for (( i=0; i<10; i+=1 )); do

printf “%-2d) %-20s %-2d) %-20s\n” $((i+1)) “${teams[i]}” $((i+11)) “${teams[i+10]}”

done

read -p “Enter your team number: “ answer

cat $MATCHES | awk -v ans=”${teams[answer-1]}” -F, ‘

BEGIN {

max\_diff = -1;

}

$3 == ans {

diff = $5 - $6;

if (diff > max\_diff) {

max\_diff = diff;

delete lines;

}

if (diff == max\_diff) {

lines[NR] = $1 “\n” $3 “ “ $5 “ vs “ $6 “ “ $4 “\n”;

}

}

END {

print(“\n”);

for (I in lines) {

print lines[i];

}

}’

}

홈 경기에서 가장 큰 점수차로 이긴 경기를 출력해주는 함수입니다. 각 팀 이름을 배열 형식으로 저장하고 팀 목록 출력과 데이터 조회에 활용하였습니다. 데이터를 조회할 때는 매치 파일에서 팀 이름과 일치하면서 점수차가 큰 경기 정보와 점수차를 변수로 저장하였습니다. 만약 점수차가 현재까지의 최다점수차와 같다면 경기 정보를 추가해주고, 최다점수차보다 높다면 지금까지 저장한 경기 정보를 지우고 해당 경기만 저장하는 식으로 구현하였습니다.

check\_arguments $@

echo "\*\*\*\*\*\*\*\*\*\*\*\*OSS1 - Project1\*\*\*\*\*\*\*\*\*\*\*\*"

echo "\* StudentID : 12181670 \*"

echo "\* Name : Hanjin Lee \*"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

while true

do

get\_user\_input

case "$reply" in

1)

query\_son\_data;;

2)

query\_team\_of\_ranking;;

3)

query\_top\_attendance\_matches;;

4)

query\_team\_info;;

5)

query\_match\_date;;

6)

query\_largest\_diff\_matches;;

7)

break;;

esac

done

각 함수를 호출하며 프로그램을 동작하도록 하는 부분입니다. 가장 먼저 파라미터 유효성 검사 함수를 호출하고 프로그램 정보를 출력합니다. 반복하여 어떤 동작을 할 것인지 유저에게 입력을 받고 입력에 맞는 동작의 함수를 호출합니다. 입력으로 7번이 들어오면 반복에서 벗어나며 프로그램이 종료됩니다.