**Introduction to Open Source SW**

**Project 1**

**(readme)**

**(모든 코드는 Github 참고)**

**12191629**

**유재웅**

**(Yu Jaewoong)**

목차

**1.이번 sh파일에 필요한 조건 3**

**2.각 화면에 대한 설명3**

1) 기본 화면 구성3

2) 조건 1 부분 구성 7

3) 조건 2 부분 구성 7

4) 조건 3 부분 구성 8

5) 조건 4 부분 구성 9

6) 조건 5 부분 구성 10

7) 조건 6 부분 구성 12

**3.각 화면에 대한 실행 결과 화면15**

1. **이번 sh파일에 필요한 조건**

|  |
| --- |
| 1. **Players.csv를 통해 손흥민이 현재 클럽에서 얻은 data를 출력할 것** 2. **Teams.csv를 통해 현재 리그의 팀의 data를 출력할 것** 3. **Matches.csv를 통해 관중이 많았던 top-3의 경기를 출력할 것** 4. **Teams.csv와 players.csv를 통해 각 팀에서 최다 득점 선수와 득점을 출력할 것** 5. **Matches.csv를 통해 약 10개의 GMT data를 변경할 것** 6. **Teams.csv와 matches.csv를 통해 각 팀이 home 경기에서 최대 격차로 이긴 경기를 출력 할 것** |

1. **각 화면에 대한 설명**
2. **기본 화면 구성**

* **기본 화면 구성 조건은 다음과 같다.**

|  |
| --- |
| 1. **화면을 출력할 때 SID, Name을 출력할 것.** 2. **3개의 file을 입력해야 실행하도록 진행할 것** |

* **따라서 3개의 파일을 입력해야만 SID와 이름 및 메뉴를 출력하도록 구성하였으며, 그 코드는 다음과 같다.**

|  |
| --- |
| * function checkCsv { * for i in "$@"; * do * if [[ "$i" == "players.csv" ]] || [[ "$i" == "matches.csv" ]] || [[ "$i" == "teams.csv" ]]; then * continue; * else * echo "$i is not csv that this program need" * return 1; * fi * done * return 0; * } * if [ $# -ne 3 ]; then * echo "this program need 3 csv files" >&2 * else * checkCsv $1 $2 $3 * if [ $? -eq 0 ]; * then * echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OSS1 Project1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"; * echo "\*          Student Id : 12191629          \*"; * echo "\*          Name : Yu Jae Woong            \*"; * echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"; * menu * fi; * fi |

* **추가적으로 3개의 파일을 읽으면서, 그 파일의 이름이 players.csv , matches.csv, teams.csv인지 확인하였다.**
* **checkCsv 함수에서는 $@를 통해 전달받은 모든 인자에 대해 측정하도록 하였는데, 이미 앞에서 if [ $# -ne 3 ] 구문을 통해 입력 받은 파일이 3개일 때에만 진행할 수 있으므로 인자를 넘겨줄 때 $1 $2 $3로 넘겨주었다. 사용자가 어떤 순서로 players.csv matches.csv teams.csv를 입력할지 모르기 때문이다.**
* **checkCsv를 통해 제대로 진행되었으면 return 0을 받고 아닌 경우에는 return 1을 받으므로 if [ $? -eq 0 ] 을 통해 앞선 함수가 성공적으로 마쳐야만 프로그램을 실행할 수 있는 메뉴를 보여주도록 하였다.**
* **Menu 함수는 다음과 같이 구성하였다.**

|  |
| --- |
| * function menu { * while true * do * echo "-------------------------------------------------------------------"; * echo "[MENU]"; * echo "1.Get the data of Heung-Min-Son's Current Club, Apperarances, 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 matches.csv"; * echo "4.Get the team's league position and team's top soccer in teams.csv & players.csv"; * echo "5.Get the modified format of date\_GMT in matches.csv"; * echo "6.Get the data of winning team by the largest difference home stadium in teams.csv & matches.csv"; * echo "7.Exit"; * read -p "Enter your CHOICE(1~7) : " c * case $c in * 1) read -p "Do you want to get the Heung-Min Son's Data? (y/n) :" tmp * execute $tmp SonData ;; * 2) teamData ;; * 3) read -p "Do you want to know Top-3 attendance data and average attendance?(y/n) :" tmp * execute $tmp top3 ;; * 4) read -p "Do you want to get each team's ranking and the highest-scoring player?(y/n) :" tmp * execute $tmp positionData ;; * 5) read -p "Do you want to modify the format of date?(y/n) :" tmp * execute $tmp GMTData ;; * 6) winningData ;; * 7) echo "Bye!"; break ;; * \*) echo "Please Enter 1~7" ;; * esac; * done; * } |

* **1부터 7이라는 수를 입력 받아야 하며, 7을 입력 받았을 때에는 종료해야 하므로, 전체적인 코드는 while true를 통해 무한 loop를 실행하고 7을 입력 받으면 break를 통해 탈출하도록 구성하였다.**
* **또한 1부터 7 사이의 수가 아닌 경우는 case 문에서 \*를 통해 모두 예외처리 하도록 하였다.**
* **또한 많은 실행에서 정말로 진행할 지에 대한 질문을 하므로, execute라는 함수를 만들었다. 이에 대한 코드는 다음과 같다.**

|  |
| --- |
| function execute {      if [ $1 ==  "y" ] || [ $1 == "Y" ];      then          $2      elif [ $1 == "n" ] || [ $1 == "N" ];      then          echo "skip";      else          echo "Please Enter Y or N";      fi  } |

* **이미 menu에서 물어볼 때 read를 통해 입력 받았고, 수행할 함수도 알기 때문에 이 2개를 넘겨주어 진행할 것인지 확인하도록 하였다.**

1. **조건 1 부분 구성**

* **조건 1에서는 손흥민의 현재 구단에서의 data를 읽어야 한다. 이 때 데이터를 정말 불러올 것인지 확인하는 구문을 추가해야 한다.**
* **전반적인 코드는 다음과 같다.**

|  |
| --- |
| * function SonData { * cat players.csv | awk -F,  '$1=="Heung-Min Son" {print "Team : " $4 " Apperance : " $6 " Goal : " $7 " Assist : " $8}'; * } |

* **Awk를 통해 players.csv에서 이름이 손흥민인 데이터를 찾을 수 있다. 이 때, 파일은 csv이므로 구분점은 반점( ‘ , ‘ )이므로 -F을 통해 구분자가 반점임을 알려주었다.**

1. **조건 2 부분 구성**

* **조건 2의 실행 순서는 menu에서 2를 입력 , 보고 싶은 팀의 순위를 입력 , 팀 순위 팀 이름 승률을 출력 이다.**
* **팀 순위를 알아야 하고 승률은 wins/(wins+draws+losses)로 출력해야한다.**
* **전반적인 코드는 다음과 같다.**

|  |
| --- |
| function teamData {      read -p "What do you want to get the team data of league\_position[1~20] : " rank      if (( $rank >= 1 && $rank <=20 ))      then            cat teams.csv | awk -F, -v a=$rank '$6==a {print $1, $2/($2+$3+$4)}';      else          echo "Wrong Input"      fi  } |

* **데이터를 출력하기에 앞서, 20개의 팀이 존재하므로 1부터 20 사이의 수를 입력 받았는 지부터 확인해야 한다.**
* **1부터 20사이의 수가 아니라면 Wrong Input만 출력하고 넘어간다.**
* **제대로 된 값을 입력 받았으면, teams.csv의 순위가 6번째 열에 존재하므로 awk를 통해 입력 받은 값과 일치하는지 확인하는 과정을 통해 찾을 수 있다. 이 때 입력 받은 변수를 사용하기 위해 -v 함수를 통해 외부 변수를 내부에 선언할 수 있도록 해주었다. 또한 csv파일이므로 -F를 통해 구분자가 반점임을 알려주었다. 이후 과정은 csv에 맞게 인자를 조정해주면 된다.**

1. **조건 3 부분 구성**

* **가장 관중이 많은 top-3 경기를 출력해야 한다.**
* **전반적인 코드는 다음과 같다.**

|  |
| --- |
| * function top3 { * cat matches.csv | sort -t, -nk2 -r | head -n 3 | awk -F, '{print $3 "  vs  " $4 " (" $1 ")"" \n" $2 " " $7}' * } |

* **먼저 matches.csv를 읽어오고, 관중수에 맞게 정렬해주었다.**
* **Csv 파일이므로 sort할 때에 구분자가 반점임을 알려주기 위해 -t를 통해 알려주었으며, 관중의 수가 2번째 열에 존재하므로 -nk2를 통해 2번째 열을 기준으로 정렬하라고 알려주었다. 기본적인 정렬은 오름차순 정렬이므로 가장 많은 관중 수를 보기 위해 -r 을 넣어 내림차순으로 정렬하도록 하였다.**
* **Top-3의 경기를 보고 싶으므로 head -n 3를 통해 앞서 관중수를 기준으로 내림차순 정렬된 csv에서 3개의 열만 가져오도록 하였다.**
* **이후 과정은 조건 1,2와 비슷하게 awk를 통해 가져오고 싶은 인자를 읽어오도록 하였다.**

1. **조건 4 부분 구성**

* **각 팀의 최다 득점자와 득점 수를 출력해야 한다.**
* **이 때 출력하는 순서는 팀의 순위에 따라 출력해야 한다.**
* **전반적인 코드는 다음과 같다.**

|  |
| --- |
| * function positionData { * cat teams.csv | awk -F, 'NR>1 {print $1 "," $6}' | sort -t,  -k2,2n > temp\_team.csv * declare -a team\_names * while IFS=, read -r name rank; * do * team\_names+=("$name") * done < <(awk -F, '{print $1}' temp\_team.csv) * i=1 * for team in "${team\_names[@]}"; * do * echo "$i : $team" * awk -F, -v team="$team" '$4 == team {print $1 "," $7}' players.csv > temp\_player.csv * cat temp\_player.csv | sort -t, -nk2 -r | head -n 1 * i=$(($i+1)) * done * $(rm temp\_team.csv) * $(rm temp\_player.csv) * } |

* **필요한 data를 따로 뽑아왔다. 우리가 teams.csv에서 가장 궁금한 부분은 팀의 이름과 순위 이므로 awk를 통해 $1(이름) $6(순위)를 반점을 통해 나누어 cat 해주는데, 이걸 스크립트에 출력하지 않고 temp\_team.csv를 만들어 입력해주었다.**
* **이후 이 팀 소속인 모든 선수를 뽑아볼 수 있도록 배열을 선언하였다. Declare -a를 통해 배열을 선언할 수 있고 여기에는 앞서 뽑은 이름과 순위에 대해 while문을 통해 넣도록 한다.**
* **저장한 배열을 모두 읽기 위해 for문에서 @를 사용해서 저장된 요소들을 불러오고, 이를 temp\_player.csv라는 일시적인 파일에 저장한다. 여기서 소속된 선수를 모두 출력한 다음, sort 를 사용하여 득점 순으로 정렬하고 가장 많이 넣은 선수 한 명을 보기 위해 head를 통해 한 줄 만 출력하도록 한다.**
* **이후 임시 파일인 temp\_team.csv와 temp\_players.csv 파일을 삭제하여 프로그램이 종료되면 시작 전과 동일하게 파일이 유지되도록 한다.**

1. **조건 5 부분 구성**

* **Matches.csv에 있는 GMT 날짜를 10개를 읽어온 다음, 특정 포멧에 맞게 변경한다.**
* **이 때 포멧은 yyyy/mm/dd h:mm(am/pm) 이며 sed를 사용하여야 한다.**
* **전반적은 코드 구성은 다음과 같다.**

|  |
| --- |
| function changeMonth {      case "$1" in          Jan) echo "01" ;;          Feb) echo "02" ;;          Mar) echo "03" ;;          Apr) echo "04" ;;          May) echo "05" ;;          Jun) echo "06" ;;          Jul) echo "07" ;;          Aug) echo "08" ;;          Sep) echo "09" ;;          Oct) echo "10" ;;          Nov) echo "11" ;;          Dec) echo "12" ;;          \*) echo "??" ;;      esac  }  function GMTData {      GMT=$(awk -F, 'NR > 1 { print $1 }' matches.csv | head -n 10)      echo "$GMT" | while read -r date;      do          formatted=$(echo "$date" | sed -E 's/^([A-Za-z]{3}) ([0-9]{1,2}) ([0-9]{4}) - ([0-9]{1,2}:[0-9]{2}[ap]m)$/\1,\2,\3,\4/')          IFS=',' read -r month day year time <<< "$formatted"          month\_num=$(changeMonth "$month")          echo "$year/$month\_num/$day $time"      done  } |

* **Matches.csv에 저장되어 있는 데이터의 경우, month 데이터가 문자로 적혀있으므로 이를 숫자로 바꾸어줄 함수 changeMonth를 만들었다.**
* **궁금한 10개의 날짜 데이터를 읽어오기 위해 GMT라는 변수에 awk를 통해 10개의 data를 넣어주었다.**
* **이후 저장된 10개를 while문을 통해 하나 씩 읽도록 하였으며, 조건에 맞게 나누어 주었다.**
* **아직 month 데이터를 changeMonth에 넣지 않았으므로 문자인 상태이다. 따라서 [A-Za-z]{3}을 통해 문자 3개 , 나머지는 숫자로 저장되어 있으므로 [0-9]로 받았으며, 연도를 받을 때는 {4}를 통해 4개임을 알려주고 시간은 각각 2개의 숫자로 표현되므로 {2}로 알려주었다.**
* **현재 Formatted라는 변수는 Sed를 통해 나눈 문자는 반점을 통해 정의되어있으므로 읽어올 때 IFS=,를 통해 나누어 줄 수 있고 이 다음 month를 changeMonth를 통해 숫자로 바꾸어주었다.**

1. **조건 6 부분 구성**

* **20개의 팀을 출력하되, 한 줄에 10개의 팀이 나오도록, 두 줄에 걸쳐 출력해야 한다.**
* **이후 출력된 팀의 번호를 입력하면, 그 팀이 home 경기에서 가장 큰 격차로 이긴 경기를 출력해야 한다.**
* **만약 큰 격차로 이긴 경기가 여러 개라면, 모두 출력할 수 있도록 한다.**
* **전반적인 코드 구성은 다음과 같다.**

|  |
| --- |
| function winningData {      first=$(awk -F, 'NR>1 {print $1}' teams.csv | head -n 10)      second=$(awk -F, 'NR>1 {print $1}' teams.csv | tail -n 10)      IFS=$'\n' read -rd '' -a first\_array <<<"$first"      IFS=$'\n' read -rd '' -a second\_array <<<"$second"      for i in {0..9};      do          first\_num=$((i + 1))      first\_team="${first\_array[$i]}"      second\_num=$((11 + i))          second\_team="${second\_array[$i]}"      printf "%2d) %-30s %2d) %-30s\n" "$first\_num" "$first\_team" "$second\_num" "$second\_team"      done      read -p "Enter your team number : " num      if (( $num >= 1 && $num <=20 ))      then          team=$(awk -F, -v r="$num" 'NR-1== r {print $1}' teams.csv)          awk -F, -v team="$team" '$3 == team { point = $5 - $6; print $1,",",$3,",",$4,",",$5,","$6,",",point}' matches.csv > temp.csv          cat temp.csv | sort -t, -r -nk6 > sortedTemp.csv;          $(rm temp.csv)          max=$(awk -F, '{print $6}' sortedTemp.csv | head -n 1);          awk -F, -v point="$max" '$6==point {              print $1 "\n" $2 $4 " vs " $5 $3}' sortedTemp.csv;          $(rm sortedTemp.csv);      else          echo "Wrong Input"      fi    } |

* **이 코드의 실행 방식은 다음 순서로 정리할 수 있다.**

|  |
| --- |
| 1. **20개의 팀을 출력하는 방법**  * **1~10번째의 팀은 head를 통해 10개를 불러온다** * **11~20번째의 팀은 tail을 통해 10개를 불러온다** * **현재 저장된 변수는 구분자가 공백이므로 IFS=’ ‘를 통해 나누어주고 이를 각자의 변수에 저장해준다.** * **for문을 통해 변수에 존재하는 값을 불러와서 1번째 줄에 1번째 팀과 11번째 팀은, 2번째 줄에 2번째 팀과 12번째 팀을 i번째 줄에 i번째 팀과 10+i번째 팀을 불러오는 방식으로 구성해준다.** * **각 팀의 이름의 길이가 모두 다르므로 printf문에서 %-30s와 같이 공백을 채우면서 문자열을 넣는 방식으로 출력해준다.**  1. **입력 받은 팀이 Home 경기에서 가장 큰 격차로 이긴 경기를 출력하는 방법**  * **먼저 입력받은 변수는 숫자이므로, 이 숫자에 해당하는 팀의 이름을 awk를 통해 받아온다.** * **이름을 알았으면, 이 팀의 모든 경기를 temp.csv라는 변수에 저장한다. Temp.csv에는 팀 이름, 상대팀 이름, 경기 날짜, 팀 득점 수 , 상대 팀 득점 수가 포함되어야 하며, 이후 가장 큰 격차를 보아야 하므로 (팀 득점 수 – 상대 팀 득점 수)로 정의된 point를 만들어 같이 저장해주었다.** * **이 temp.csv를 point를 기준으로 정렬하여 sortedTemp.csv라는 임시파일에 저장해준다.** * **가장 큰 격차로 이겼을 때의 격차를 알기 위해 head를 통해 하나만 뽑아 point를 출력하여 max라는 변수에 저장해준다.** * **이후 sortedTemp.csv에는 입력 받은 팀의 모든 경기가 있으므로, 각 경기의 득점 격차가 max와 동일하면 모두 출력하도록 awk문을 작성해주었다.** * **이후 생성한 모든 임시 파일을 제거해주었다.** |

1. **각 화면에 대한 실행 결과 화면**

* **모든 경우에 대한 실행 결과는 아니며, 전반적인 코드 및 프로그램은 github를 통해 확인할 수 있다.**
* **기본 화면 출력**

|  |
| --- |
|  |

* **조건 1 출력**

|  |
| --- |
|  |

**-조건 2 출력**

|  |
| --- |
|  |

* **조건 3 출력**

|  |
| --- |
|  |

**-조건 4 출력**

|  |
| --- |
|  |

* **조건 5 출력**

|  |
| --- |
|  |

* **조건 6 출력**

|  |
| --- |
|  |

* **탈출 조건 출력**

|  |
| --- |
|  |