```
#! /bin/bash
/*
배쉬를 통해 실행할 수 있도록 추가합니다
*/

#매개변수 개수 확인
if [ $# -ne 3 ]; then
  echo "사용법: ./test.sh u.item u.data u.user"
  exit 1
fi
/*
매개변수 개수를 확인해 3개가 제대로 들어왔는지 확인하고, 그렇지 않으면 사용방법을
출력합니다.
*/

item="./$1"
data="./$2"
user="./$3"
/*
쉘 파일과 같은 디렉토리 내에 있는 매개변수들의 경로를 쉘 변수에 저장합니다.
*/

func1() {
        echo ""
        read -p "Please enter 'movie id' (1~1682) :" mId
        echo ""
        cat $item | awk -v num=$mId 'NR==num {print $0}'
        echo ""
}
/*
movie id를 입력받고, cat $item으로 출력한 결과에 awk를 이용합니다.
awk 변수로 num에 쉘변수 mId를 받고, 몇번째 행인지를 나타내는 NR과 num을 비교해
같은 행은 전체 열을 출력합니다.
*/

func2(){
        echo ""
        read -p "Do you want to get the data of ‘action’ genre movies from 'u.item’?
(y/n): " YESNO
        echo ""
        if [ $YESNO = "y" ]; then
                cat $item | awk -F '|' '$7==1 {print $1, $2}' | head -n 10
        fi
```

```
        echo ""
}
/*
y 또는 n을 입력받고, cat $item으로 출력한 결과에 awk를 이용합니다.
구분을 '|'로 할 수 있도록 -F 옵션을 주고, '|'로 구분했을 때
u.item의 7번째 열에 action영화임을 나타내는 이진수가 존재하므로 7번째 열이 1이면 영화의 index와 이름을 출력합니다.
맨 끝의 | head -n 10을 통해 상위 10줄만을 출력합니다.
*/

func3(){
        echo ""
        read -p "Please enter 'movie id' (1~1682) :" mId

        local sum=$(cat $data | awk -v num=$mId '$2==num {sum+=$3} END {print sum}')
        local count=$(cat $data | awk -v num=$mId '$2==num {count++} END {print count}')



        echo ""
        echo "$sum $count" | awk -v num=$mId '{printf("average rating of %d: %.g₩n₩n", num, $1/$2)}'
}
/*
movie id를 입력받고, cat $data로 출력한 결과에 awk를 이용합니다.
mId를 awk 변수 num으로 받고, cat의 출력 결과의 2번째 열인 평가된 영화의 id와 num을 비교합니다.
같다면 sum에 3번째 열에 있는 평점을 더하는데, END {print sum}을 통해 끝날 때 sum을 출력한 후 local sum에 출력된 sum을 저장합니다.
count도 sum과 동일한 방식으로 구합니다.

echo를 통해 num과 count를 출력한 결과를 awk를 이용해 평점의 평균을 계산합니다.
printf로 포맷팅된 문장에 movie id인 num과 평균인 $1/$2를 넣어 문장을 완성합니다.
포맷팅 할 때 %d는 정수, %g는 실수지만 소숫점 뒤 쓸모없는 0을 지워 출력합니다.
*/

func4(){
        echo ""
        read -p "Do you want to delete the 'IMDb URL' from 'u.item'?(y/n):" YESNO
        echo""
        if [ $YESNO = "y" ]; then
                cat $item | sed -E
```

```
's/(.*)₩|(.*)₩|(.*)₩|₩|(.*)₩|([01₩|]{37})/₩1₩|₩2₩|₩3₩|₩|₩|₩5/' | head -n 10
        fi
        echo""
}
/*
y또는 n을 입력받고, cat $item의 출력 결과에 sed를 이용합니다.
Extended Regular Expression syntax를 이용하기 위해 -E 옵션을 주고, 's/패턴/대체할것/
액션'에서 (.*)로 단어를 묶고, |로 구분하고,
주소 부분인 $4를 뺀 후 대체문장을 입력했습니다. |는 이스케이프문자를 이용해 원래 기
능인 파이프 기능을 없애주어야 제대로 작동합니다.
마지막 | head -n 10로 상위 10줄을 출력합니다.
*/

func5(){
        echo ""
        read -p "Do you want to get the data about users from 'u.user'?(y/n):" YESNO
        echo""
        if [ $YESNO = "y" ]; then
                cat $user | sed -E 's/(.*)₩|(.*)₩|(.*)₩|(.*)₩|(.*)/user ₩1 is ₩2 years old
₩3 ₩4/' | sed -E 's/M/male/' | sed -E 's/F/female/' | head -n 10
        fi
        echo""
}
/*
y또는 n을 입력받고, cat $user의 출력 결과에 sed를 이용합니다.
 Extended Regular Expression syntax를 이용하기 위해 -E 옵션을 주고, 's/패턴/대체할것/
액션'에서 (.*)로 단어를 묶고, |로 구분하고,
묶어서 구분해놨던 단어들을 ₩1 ₩2 ₩3 ₩4을 이용해 새로운 문장에 집어 넣고 출력합니
다.
이 결과를 다시 sed로 M을 male로 대체하고, F를 female로 대체한 후 마지막 | head -n
10로 상위 10줄을 출력합니다.
*/

func6(){
        echo ""
        read -p "Do you want to Modify the format of ‘release data’ in ‘u.item’?(y/n):"
YESNO
        echo""
        if [ $YESNO = "y" ]; then
                cat $item | 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' | sed -E 's/([0-9]{2})-([0-9]{2})-([0-9]{4})/₩3₩2₩1/g'
```

```
| tail -n 10
        fi
        echo""
}
/*
y와 n을 입력받고, cat $item의 출력 결과에 sed를 이용합니다.
Extended Regular Expression syntax를 이용하기 위해 -E 옵션을 주고, -로 시작하면서 월에 관한 영어를 숫자로 대체하도록 했고,
찾은 모든 것을 대체하도록 's///g'의 g를 추가했습니다. 1월에서 12월 모두를 바꿔야 하므로 순차적으로 진행하도록 ' ' 안의 s///g 구문들을 ;로 구분했습니다.
이스케이프 문자를 이용해 -의 원래 기능인 범위 기능을 없앴습니다.
영어로 표현된 월을 모두 숫자로 바꾼 출력을 다시 sed에서 ([0-9]{2})-([0-9]{2})-([0-9]{4})패턴을 통해 00-11-2222의 패턴을 찾아 22221100으로 대체하였습니다.
마지막 | tail -n 10를 통해 최하위 10줄을 출력합니다.
*/

func7(){
        echo ""
        read -p "Please enter the 'user id'(1~943):" uId
        echo""

        cat $data | awk -v num=$uId '$1==num {print $2}' | sort -n | awk '{printf("%d | ", $1)}'
        mvs=$(cat $data | awk -v num=$uId '$1==num {print $2}' | sort -n | head -n 10)

        echo ""
        echo ""
        for i in $mvs
        do
                cat $item | awk -F '|' -v num=$i 'num==$1{printf("%d | %s", $1, $2)}'
                echo ""
        done
        echo ""
}
/*
user id를 입력받고, cat $data를 통한 출력결과에 awk를 이용합니다.
awk에서 쉘 변수 uId를 awk변수 num으로 받고, u.data의 첫번째 열인 user id와 비교하여 동일하다면 두번째 열인 movie id를 출력합니다.
이를 | sort -n를 통해 정렬을 한 후, | awk '{printf("%d | ", $1)}를 통해 user가 평가한 영화들의 movie id들을 '|'로 구분지어 모두 출력합니다.
cat $data를 awk를 활용해 영화의 id를 뽑고 정렬까지 한 후 상위 10 줄을 mvs 변수에 받습니다.
```

for문을 통해 mvs의 변수 한줄 한줄을 i에 놓고, cat $item의 출력결과를 -F옵션으로 '|'로 구분하도록 한 후 awk변수 num에 i를 받습니다.
num과 u.item의 첫번째 열인 movie id가 같다면, printf로 포맷팅된 문장에 movie id, 영화이름을 넣고 10개의 영화들을 출력합니다.
*/

```
func8(){
        echo ""
        read -p "Do you want to get the average 'rating' ofmovies rated by users with 'age' between 20 and 29 and 'occupation' as 'programmer'?(y/n):" YESNO
        echo""

        if [ $YESNO = "y" ]; then
                programmers20s=$(cat $user | awk -F '|' '$2>=20 && $2<30 && $4=="programmer" {print $1}' | sort -n | uniq)
        fi

        ratedMovie=""
        for programmersId in $programmers20s
        do
                ratedData+=$(cat $data | awk -v pId=$programmersId '$1==pId {printf("₩n%s"), $0}')
        done

        for i in $(seq 1 1682)
        do
                local sum=$(echo "$ratedData" | awk -v num=$i '$2==num {print $0}' | awk '{sum+=$3} END {print sum}')
                local count=$(echo "$ratedData" | awk -v num=$i '$2==num {print $0}' | wc -l)
                if [ "$count" = "0" ];
                then
                        continue
                else
                        echo "$i $sum $count" | awk '{printf("%d %g₩n", $1, $2/$3)}'
                fi
        done
        echo ""
}
/*
y또는 n을 입력받고, cat $user의 출력값을 awk를 이용해 -F 옵션을 주고, '|'로 구분하면서 2열의 나이와 4열의 직업을 비교해
```

20대 프로그래머들을 programmers20s에 받습니다. 받을 때, | sort -n | uniq를 통해 정렬과 중복을 제거합니다.

cat $data의 출력값을 awk를 이용해 awk변수인 pId에 programmersId를 받고, 1열인 user id와 pId가 같다면 모든 열을 출력하고 이를 ratedData에 받습니다.
for문을 통해 echo "$ratedData"의 결과에 awk를 이용해 awk 변수 num에 i를 받고, 2열인 movie id와 num이 동일하다면 모든 열을 출력합니다
출력된 것을 awk를 이용해 3열인 rating을 sum에 다 더한 후, 끝나면서 sum을 출력하고 그것을 지역변수 sum에 받습니다.
count는 대체로 동일한 방식이지만, 마지막 | wc -l를 통해 행의 수로 count를 구합니다.
만약 count 0이라면 건너뛰고, 0이 아니라면 echo "$i $sum $count"의 출력값을 awk를 이용해 ₩
포맷팅된 문장에 i와 sum/count를 집어넣어 문장을 완성해 출력합니다.
포맷팅 할 때 %d는 정수, %g는 실수지만 소숫점 뒤 쓸모없는 0을 지워 출력합니다.

u.user에서 20대 프로그래머들의 user id들을 구하고, u.data에서 20대 프로그래머들의 user id로 평가된 data들을 구하고,
for문을 통해 movie id가 1~1682인 것들로 각각 sum, count를 구하고, i,sum,count를 통해 movie id, 20대 프로그래머들이 평가한 평균을 출력하는 과정으로 코드를 짰습니다.
*/

```
func9(){
        echo "Bye!"
        exit 1
}
/*
echo를 통해 Bye!를 출력하고, exit에 1값을 넣어 종료합니다.
*/

echo "--------------------------"
echo "User Name: Yoonsuyeong"
echo "Student Number: 12223763"

echo "[ MENU ]"
echo "1. Get the data of the movie identified by a specific 'movie id' from 'u.item'"
echo "2. Get the data of action genre movies from 'u.item'"
echo "3. Get the average 'rating' of the movie identified by specific 'movie id' from 'u.data'"
echo "4. Delete the 'IMDb URL' from 'u.item"
echo "5. Get the data about users from 'u.user'"
echo "6. Modify the format of 'release date' in 'u.item'"
echo "7. Get the data of movies rated by a specific 'user id' from 'u.data'"
echo "8. Get the average 'rating' of movies rated by users with 'age' between 20 and
```

```
29 and 'occupation' as 'programmer'"
echo "9. Exit"
echo "-------------------------"

while true
do
        read -p "Enter your choice [ 1-9 ] " choice
        case $choice in
                1)
                func1
                ;;
                2)
                func2
                ;;
                3)
                func3
                ;;
                4)
                func4
                ;;
                5)
                func5
                ;;
                6)
                func6
                ;;
                7)
                func7
                ;;
                8)
                func8
                ;;
                9)
                func9
                ;;
        esac
done
/*
read에 -p옵션을 주어 choice를 입력받고, 그것을 토대로 case문을 반복합니다.
while true
do

done
```

을 통해 Ctrl+C를 입력받을 때 까지 안 쪽 내용인 case문을 반복합니다.

```
case 쉘변수 in
        case)
        ;;
        case)
        ;;
        ..
esac
```

을 통해 쉘 변수 안 내용에 따라 함수들을 실행합니다.
*)는 디폴트 상태이지만 선택적으로 입력하면 되고,
choice가 1~9가 아니라면 반복적으로 입력을 받기 때문에 디폴트 상태를 입력하지 않았습니다.
*/