YILDIZ TECHNICAL UNIVERSITY

ELECTRICAL- ELECTRONICS

FACULTY

COMPUTER ENGINEERING

DEPARTMENT

KAMRAN BALAYEV 17011904

BFS on Graph Application

In this assignment I will explain the code of Bread First Search algorithm on Graphs. Breadth First Search is an algorithm used to search the Tree or Graph. BFS search starts from root node then traversal into next level of graph or tree and continues, if item found it stops otherwise it continues. In this application I have used 8 functions excluded main function and 2 structures.

First structure has used for creating actor with 5 parameters:

- ❖ Name of actor
- ❖ Movies of actor
- ❖ Number of movies of actor
- ❖ Visit flag for unique control
- ❖ Tail path variable for printing

Second structure has used for creating movies with 5 parameters:

- ❖ Name of movie
- ❖ Actors of movie
- ❖ Visit flag for unique control
- ❖ Tail path variable for printing

Actor queue and Film queue both have enqueue and dequeue functions and start and finish indexs. Default value of start index value is –1 and default value of finish index value is 0. Enqueue function adds new element to the queue and Dequeue function removes element from queue.

In order to show the path of the actor, I need a function. This function is named as show finded path, with 2 parameters:

- ❖ Actor queue
- ❖ Integer value which represents the path

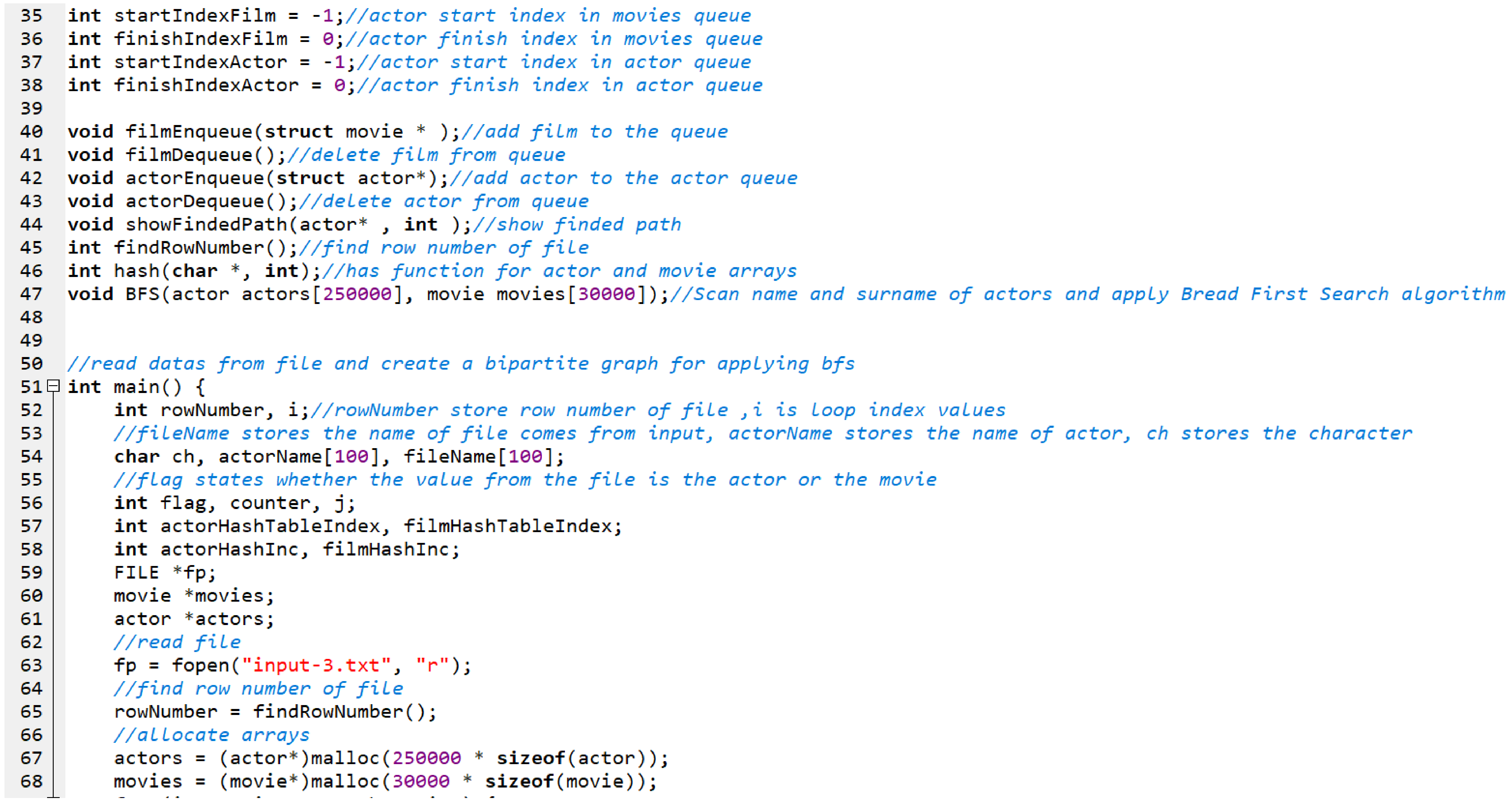
In order to search actor names, I need to calculate the row number of file and loop. Find row number function reads file and calculate the row numbers of file.

I have used hash table for storing the Kevin Bacon information of actors and the function which named as hash do these operations.

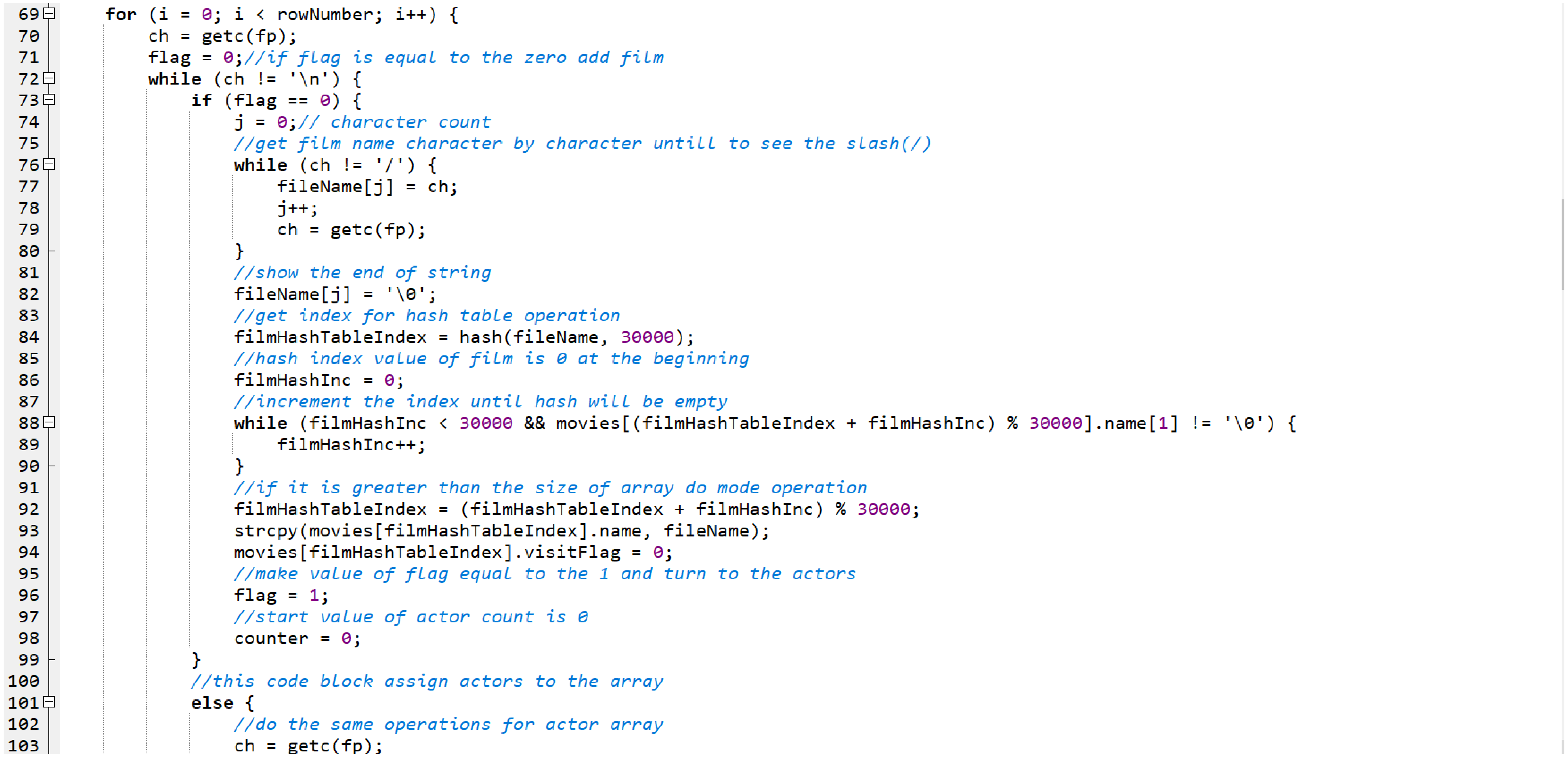
Finally, for doing BFS operation I have created BFS function with actor array and movie array parameters. This function contains user input also. User input should be in format of Surname, Name(attention to the empty space character).

Detailed explanations and screenshots are available below:

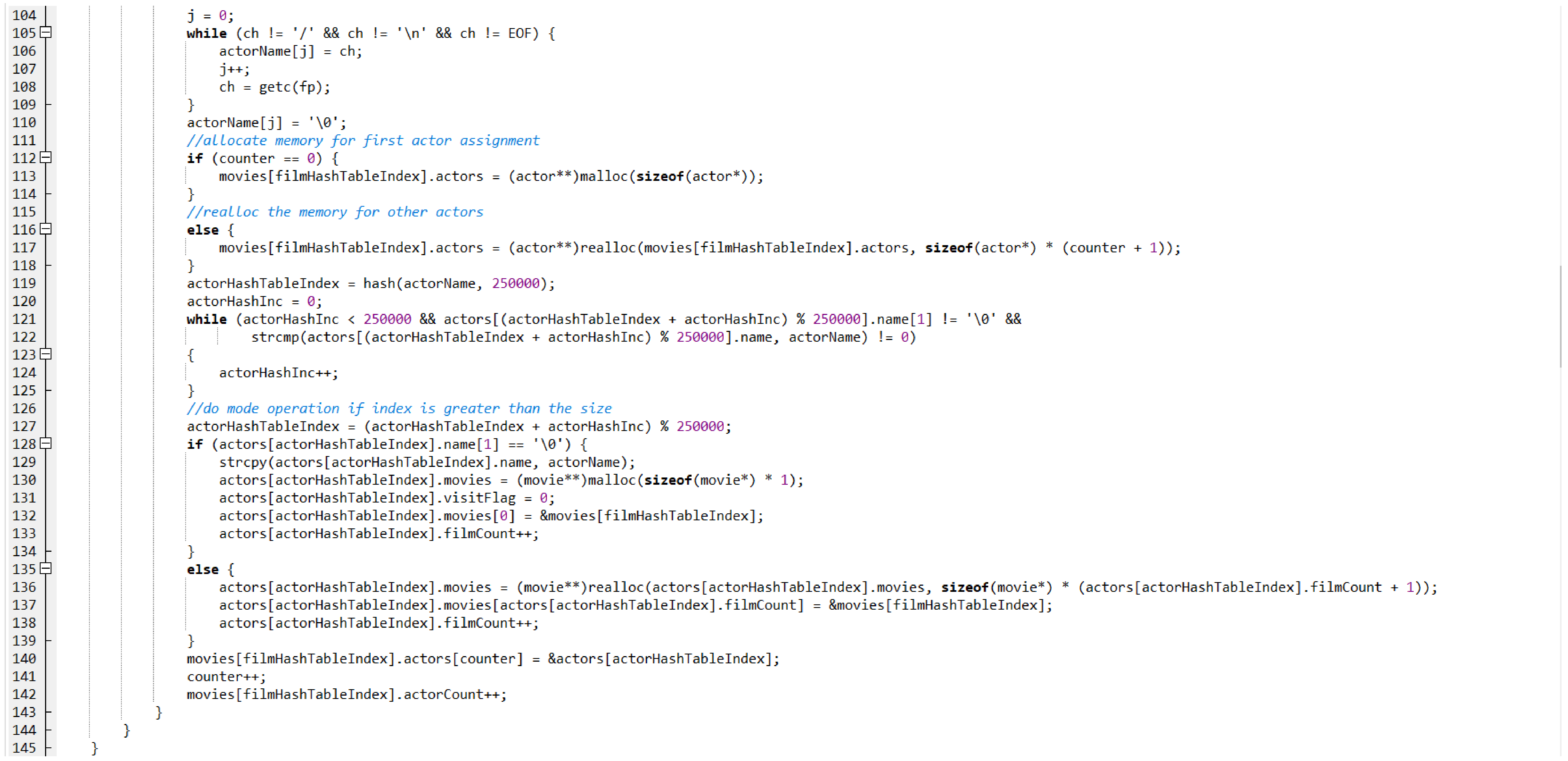
```c
/*
    KAMRAN BALAYEV
    17011904
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<math.h>
#define MAX 300

//struct for actors
typedef struct actor {
    char name[MAX];//store the name of actor
    struct movie **movies;//movies of actor
    int filmCount;//number of movies played by actor
    short visitFlag;//control for unique adding to the queue
    struct movie *tailPath;//will be used in printing the tail path
}actor;

//struct for movies
typedef struct movie {
    char name[MAX];//store the name of movies
    struct actor **actors;//actors which played in movie
    short visitFlag;//control for unique adding to the queue
    struct actor *tailPath;//will be used in printing the tail path
    int actorCount;//number of actors played in movie
}movie;

/*  BFS operation was done by using 2 queue
    one of them is actor queue and
    another one is film queue
*/
struct actor **actorQueue;
struct movie **filmQueue;
```



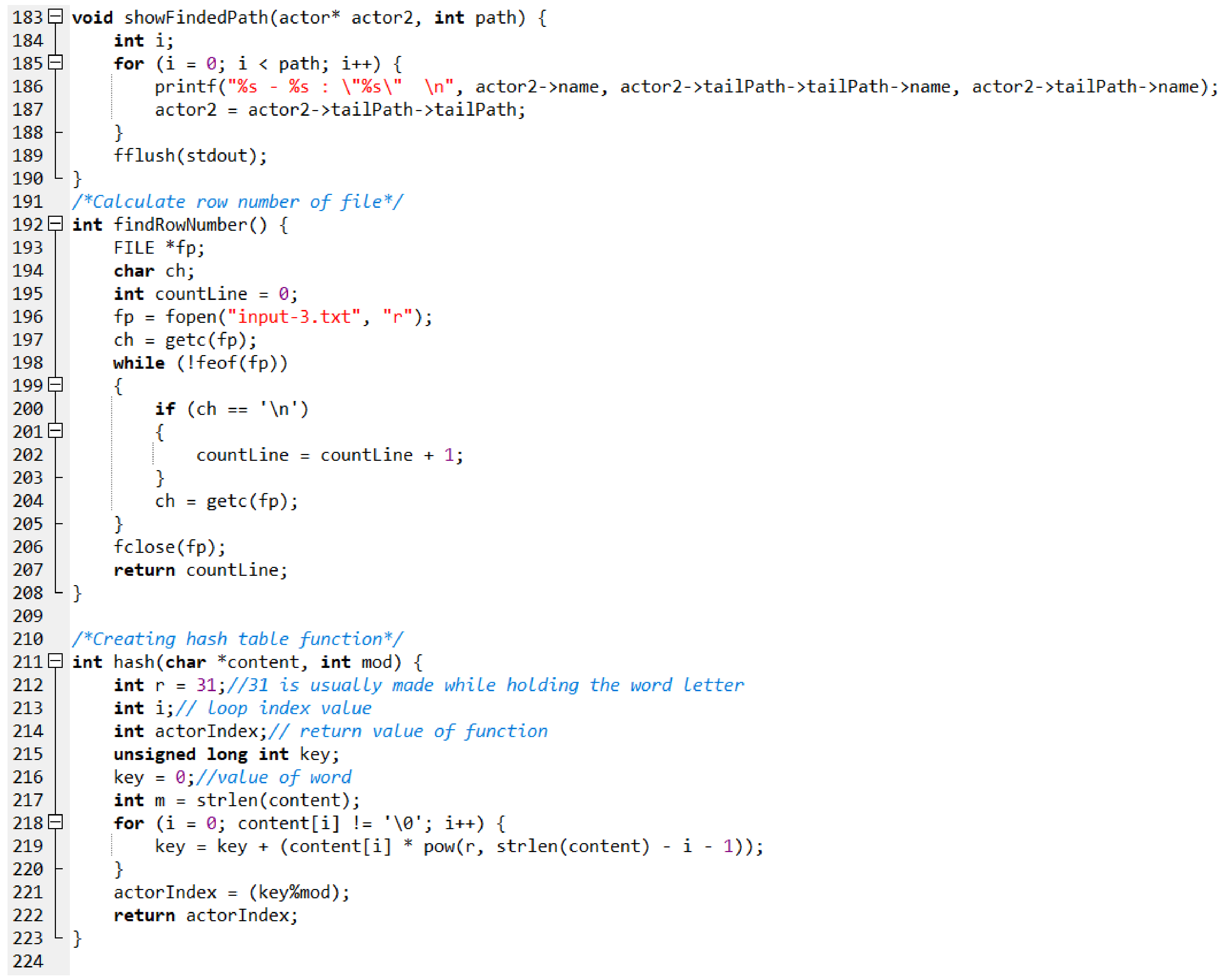
```c
int startIndexFilm = -1;//actor start index in movies queue
int finishIndexFilm = 0;//actor finish index in movies queue
int startIndexActor = -1;//actor start index in actor queue
int finishIndexActor = 0;//actor finish index in actor queue

void filmEnqueue(struct movie * );//add film to the queue
void filmDequeue();//delete film from queue
void actorEnqueue(struct actor*);//add actor to the actor queue
void actorDequeue();//delete actor from queue
void showFindedPath(actor* , int );//show finded path
int findRowNumber();//find row number of file
int hash(char *, int);//has function for actor and movie arrays
void BFS(actor actors[250000], movie movies[30000]);//Scan name and surname of actors and apply Bread First Search algorithm


//read datas from file and create a bipartite graph for applying bfs
int main() {
    int rowNumber, i;//rowNumber store row number of file ,i is loop index values
    //fileName stores the name of file comes from input, actorName stores the name of actor, ch stores the character
    char ch, actorName[100], fileName[100];
    //flag states whether the value from the file is the actor or the movie
    int flag, counter, j;
    int actorHashTableIndex, filmHashTableIndex;
    int actorHashInc, filmHashInc;
    FILE *fp;
    movie *movies;
    actor *actors;
    //read file
    fp = fopen("input-3.txt", "r");
    //find row number of file
    rowNumber = findRowNumber();
    //allocate arrays
    actors = (actor*)malloc(250000 * sizeof(actor));
    movies = (movie*)malloc(30000 * sizeof(movie));
```



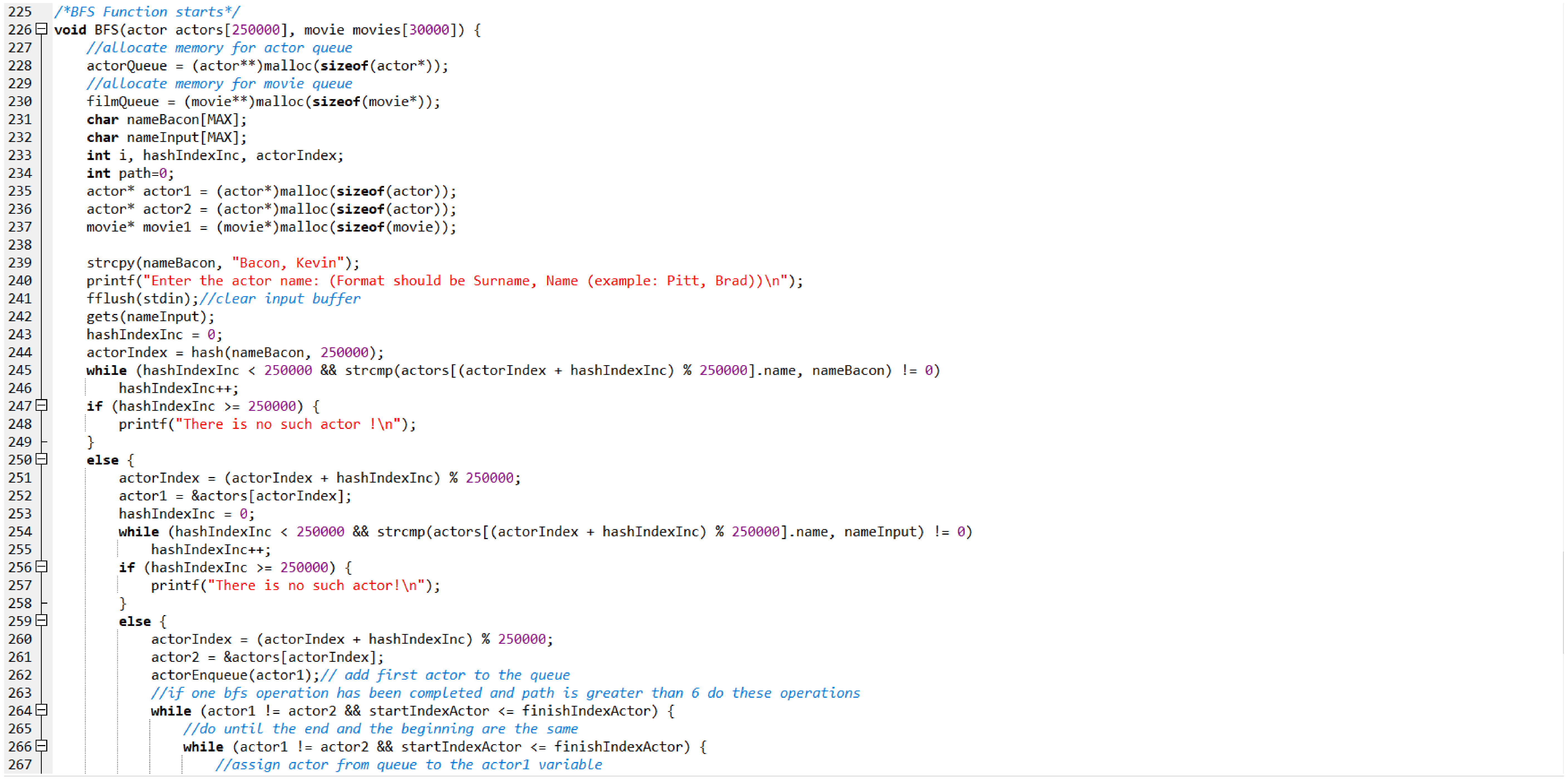
```c
    for (i = 0; i < rowNumber; i++) {
        ch = getc(fp);
        flag = 0;//if flag is equal to the zero add film
        while (ch != '\n') {
            if (flag == 0) {
                j = 0;// character count
                //get film name character by character untill to see the slash(/)
                while (ch != '/') {
                    fileName[j] = ch;
                    j++;
                    ch = getc(fp);
                }
                //show the end of string
                fileName[j] = '\0';
                //get index for hash table operation
                filmHashTableIndex = hash(fileName, 30000);
                //hash index value of film is 0 at the beginning
                filmHashInc = 0;
                //increment the index until hash will be empty
                while (filmHashInc < 30000 && movies[(filmHashTableIndex + filmHashInc) % 30000].name[1] != '\0') {
                    filmHashInc++;
                }
                //if it is greater than the size of array do mode operation
                filmHashTableIndex = (filmHashTableIndex + filmHashInc) % 30000;
                strcpy(movies[filmHashTableIndex].name, fileName);
                movies[filmHashTableIndex].visitFlag = 0;
                //make value of flag equal to the 1 and turn to the actors
                flag = 1;
                //start value of actor count is 0
                counter = 0;
            }
            //this code block assign actors to the array
            else {
                //do the same operations for actor array
                ch = getc(fp);
```



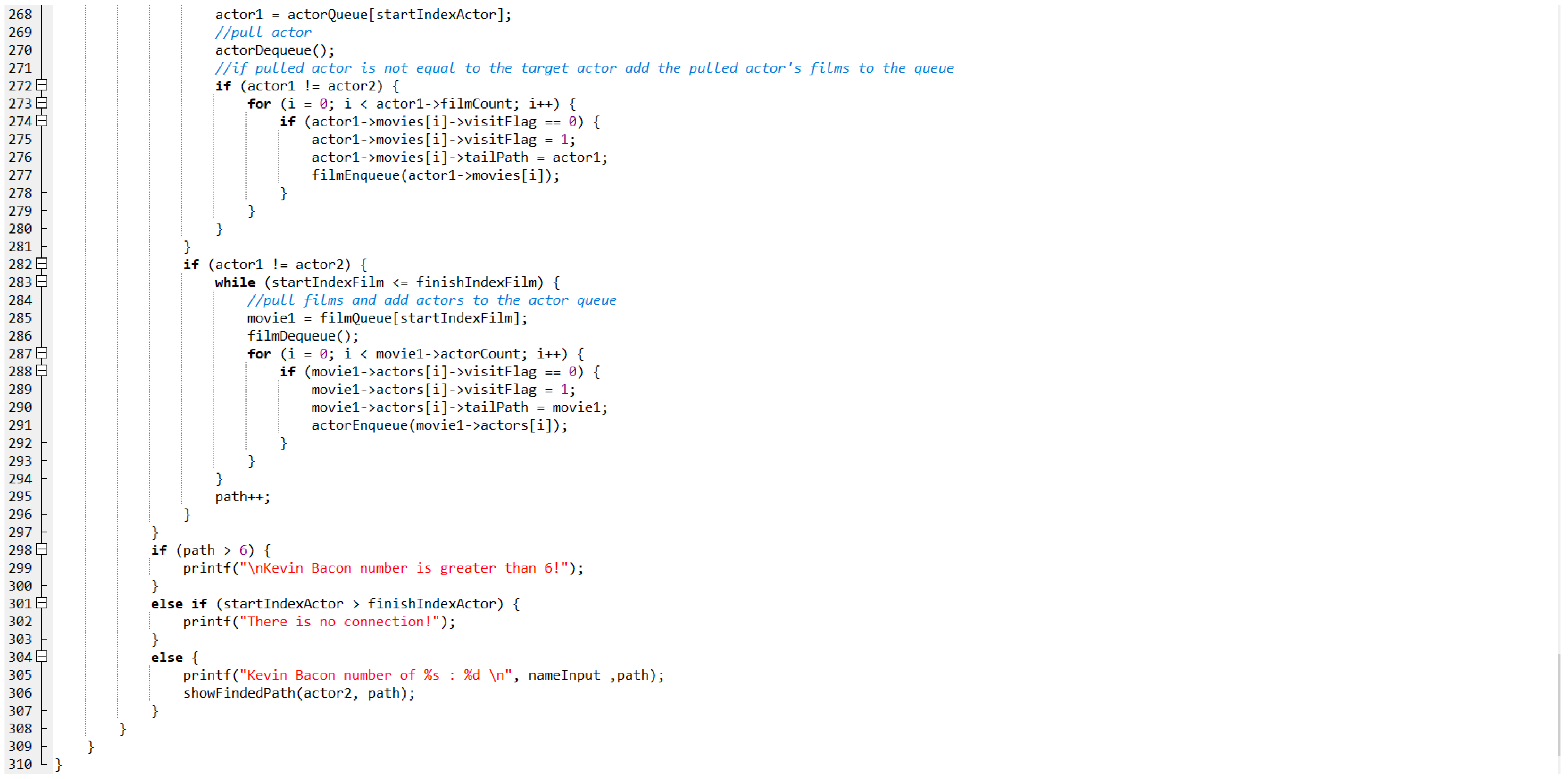
```c
                j = 0;
                while (ch != '/' && ch != '\n' && ch != EOF) {
                    actorName[j] = ch;
                    j++;
                    ch = getc(fp);
                }
                actorName[j] = '\0';
                //allocate memory for first actor assignment
                if (counter == 0) {
                    movies[filmHashTableIndex].actors = (actor**)malloc(sizeof(actor*));
                }
                //realloc the memory for other actors
                else {
                    movies[filmHashTableIndex].actors = (actor**)realloc(movies[filmHashTableIndex].actors, sizeof(actor*) * (counter + 1));
                }
                actorHashTableIndex = hash(actorName, 250000);
                actorHashInc = 0;
                while (actorHashInc < 250000 && actors[(actorHashTableIndex + actorHashInc) % 250000].name[1] != '\0' &&
                        strcmp(actors[(actorHashTableIndex + actorHashInc) % 250000].name, actorName) != 0)
                {
                    actorHashInc++;
                }
                //do mode operation if index is greater than the size
                actorHashTableIndex = (actorHashTableIndex + actorHashInc) % 250000;
                if (actors[actorHashTableIndex].name[1] == '\0') {
                    strcpy(actors[actorHashTableIndex].name, actorName);
                    actors[actorHashTableIndex].movies = (movie**)malloc(sizeof(movie*) * 1);
                    actors[actorHashTableIndex].visitFlag = 0;
                    actors[actorHashTableIndex].movies[0] = &movies[filmHashTableIndex];
                    actors[actorHashTableIndex].filmCount++;
                }
                else {
                    actors[actorHashTableIndex].movies = (movie**)realloc(actors[actorHashTableIndex].movies, sizeof(movie*) * (actors[actorHashTableIndex].filmCount + 1));
                    actors[actorHashTableIndex].movies[actors[actorHashTableIndex].filmCount] = &movies[filmHashTableIndex];
                    actors[actorHashTableIndex].filmCount++;
                }
                movies[filmHashTableIndex].actors[counter] = &actors[actorHashTableIndex];
                counter++;
                movies[filmHashTableIndex].actorCount++;
            }
        }
    }
    BFS(actors, movies);
    return 0;
}

/*realloc memory and add film to the end of queue*/
void filmEnqueue(struct movie *film) {
    if (startIndexFilm == -1) {
        startIndexFilm++;
        filmQueue[startIndexFilm] = film;
    }
    else {
        finishIndexFilm++;
        filmQueue = (struct movie**)realloc(filmQueue, sizeof(movie*)*(finishIndexFilm + 1));
        filmQueue[finishIndexFilm] = film;
    }
}

void filmDequeue() {
    startIndexFilm++;
}
/*realloc memory and add actor to the end of queue*/
void actorEnqueue(struct actor *Actor) {
    if (startIndexActor == -1) {
        startIndexActor++;
        actorQueue[startIndexActor] = Actor;
    }
    else {
        finishIndexActor++;
        actorQueue = (struct actor**)realloc(actorQueue, sizeof(actor*)*(finishIndexActor + 1));
        actorQueue[finishIndexActor] = Actor;
    }
}

void actorDequeue() {
    startIndexActor++;
}

```

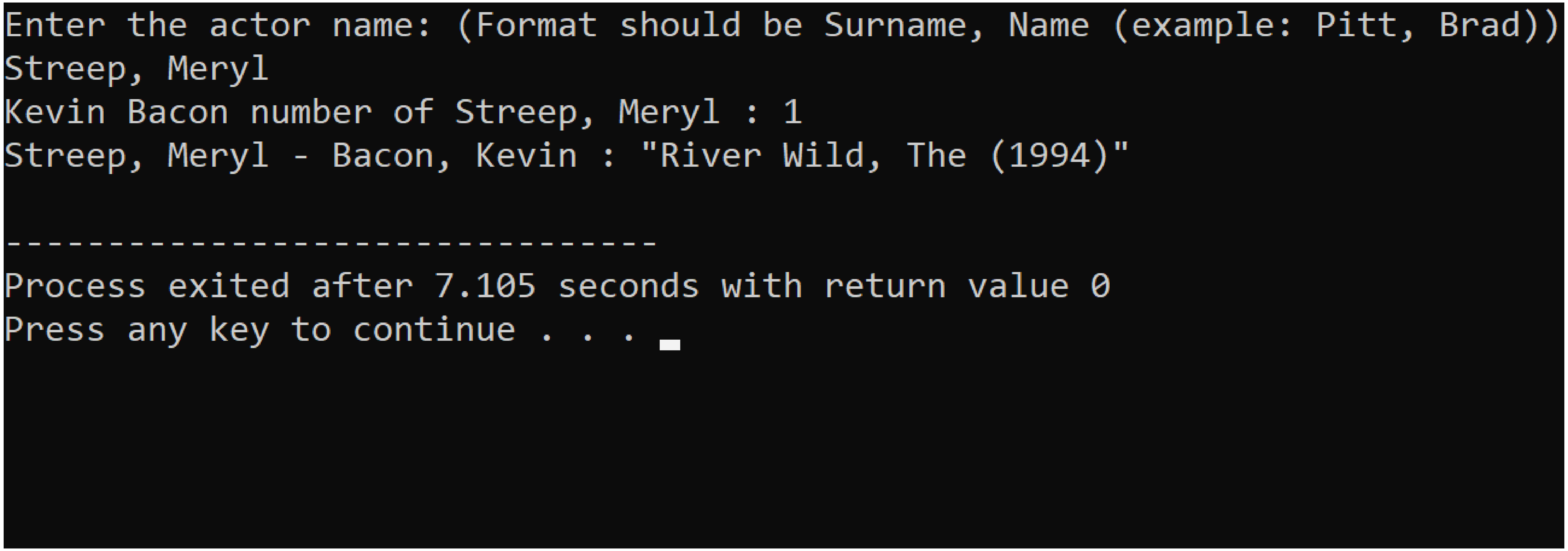


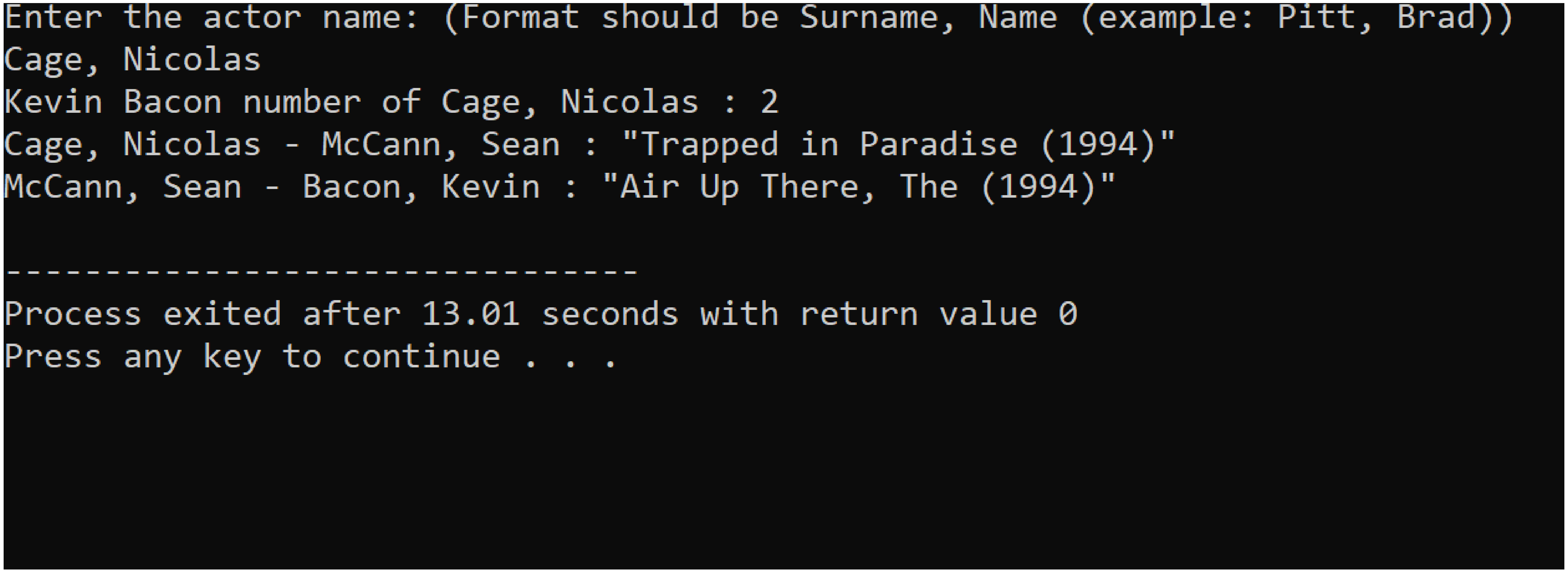
```c
void showFindedPath(actor* actor2, int path) {
    int i;
    for (i = 0; i < path; i++) {
        printf("%s - %s : \"%s\"  \n", actor2->name, actor2->tailPath->tailPath->name, actor2->tailPath->name);
        actor2 = actor2->tailPath->tailPath;
    }
    fflush(stdout);
}
/*Calculate row number of file*/
int findRowNumber() {
    FILE *fp;
    char ch;
    int countLine = 0;
    fp = fopen("input-3.txt", "r");
    ch = getc(fp);
    while (!feof(fp))
    {
        if (ch == '\n')
        {
            countLine = countLine + 1;
        }
        ch = getc(fp);
    }
    fclose(fp);
    return countLine;
}

/*Creating hash table function*/
int hash(char *content, int mod) {
    int r = 31;//31 is usually made while holding the word letter
    int i;// loop index value
    int actorIndex;// return value of function
    unsigned long int key;
    key = 0;//value of word
    int m = strlen(content);
    for (i = 0; content[i] != '\0'; i++) {
        key = key + (content[i] * pow(r, strlen(content) - i - 1));
    }
    actorIndex = (key%mod);
    return actorIndex;
}
```



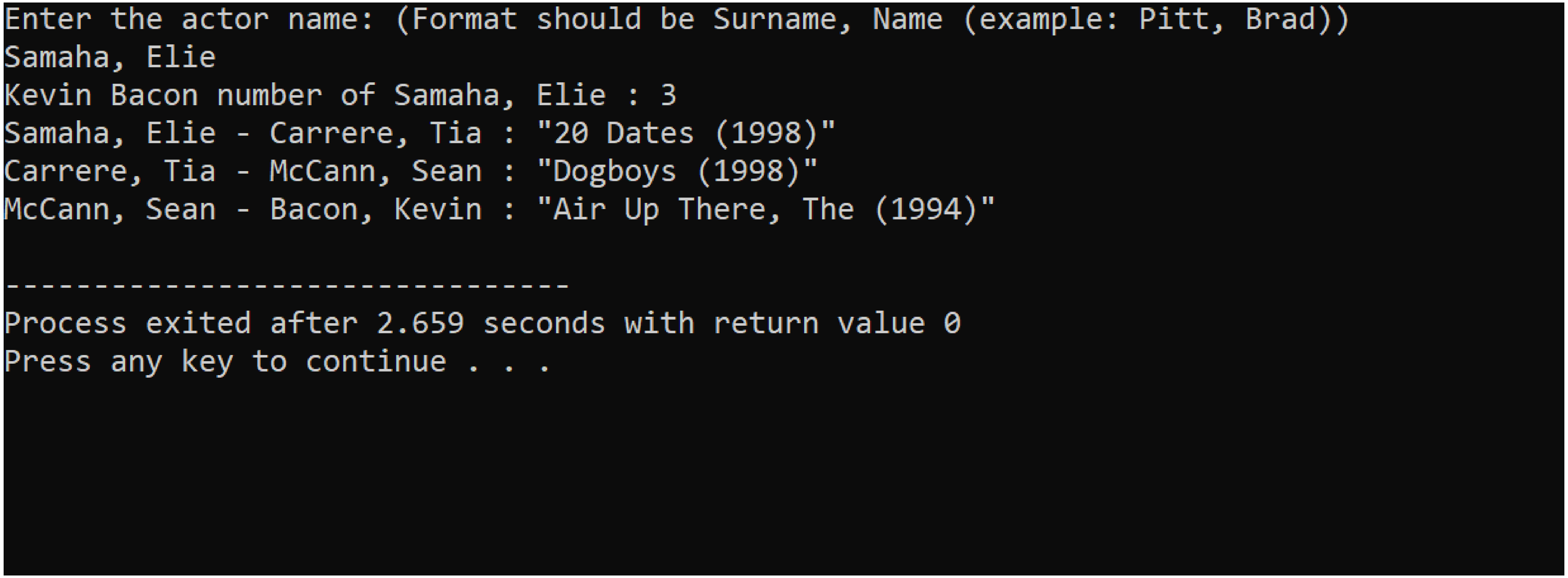
```c
/*BFS Function starts*/
void BFS(actor actors[250000], movie movies[30000]) {
    //allocate memory for actor queue
    actorQueue = (actor**)malloc(sizeof(actor*));
    //allocate memory for movie queue
    filmQueue = (movie**)malloc(sizeof(movie*));
    char nameBacon[MAX];
    char nameInput[MAX];
    int i, hashIndexInc, actorIndex;
    int path=0;
    actor* actor1 = (actor*)malloc(sizeof(actor));
    actor* actor2 = (actor*)malloc(sizeof(actor));
    movie* movie1 = (movie*)malloc(sizeof(movie));

    strcpy(nameBacon, "Bacon, Kevin");
    printf("Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))\n");
    fflush(stdin);//clear input buffer
    gets(nameInput);
    hashIndexInc = 0;
    actorIndex = hash(nameBacon, 250000);
    while (hashIndexInc < 250000 && strcmp(actors[(actorIndex + hashIndexInc) % 250000].name, nameBacon) != 0)
        hashIndexInc++;
    if (hashIndexInc >= 250000) {
        printf("There is no such actor !\n");
    }
    else {
        actorIndex = (actorIndex + hashIndexInc) % 250000;
        actor1 = &actors[actorIndex];
        hashIndexInc = 0;
        while (hashIndexInc < 250000 && strcmp(actors[(actorIndex + hashIndexInc) % 250000].name, nameInput) != 0)
            hashIndexInc++;
        if (hashIndexInc >= 250000) {
            printf("There is no such actor!\n");
        }
        else {
            actorIndex = (actorIndex + hashIndexInc) % 250000;
            actor2 = &actors[actorIndex];
            actorEnqueue(actor1);// add first actor to the queue
            //if one bfs operation has been completed and path is greater than 6 do these operations
            while (actor1 != actor2 && startIndexActor <= finishIndexActor) {
                //do until the end and the beginning are the same
                while (actor1 != actor2 && startIndexActor <= finishIndexActor) {
                    //assign actor from queue to the actor1 variable
```


```
                    actor1 = actorQueue[startIndexActor];
                    //pull actor
                    actorDequeue();
                    //if pulled actor is not equal to the target actor add the pulled actor's films to the queue
                    if (actor1 != actor2) {
                        for (i = 0; i < actor1->filmCount; i++) {
                            if (actor1->movies[i]->visitFlag == 0) {
                                actor1->movies[i]->visitFlag = 1;
                                actor1->movies[i]->tailPath = actor1;
                                filmEnqueue(actor1->movies[i]);
                            }
                        }
                    }
                }
                if (actor1 != actor2) {
                    while (startIndexFilm <= finishIndexFilm) {
                        //pull films and add actors to the actor queue
                        movie1 = filmQueue[startIndexFilm];
                        filmDequeue();
                        for (i = 0; i < movie1->actorCount; i++) {
                            if (movie1->actors[i]->visitFlag == 0) {
                                movie1->actors[i]->visitFlag = 1;
                                movie1->actors[i]->tailPath = movie1;
                                actorEnqueue(movie1->actors[i]);
                            }
                        }
                    }
                    path++;
                }
            }
            if (path > 6) {
                printf("\nKevin Bacon number is greater than 6!");
            }
            else if (startIndexActor > finishIndexActor) {
                printf("There is no connection!");
            }
            else {
                printf("Kevin Bacon number of %s : %d \n", nameInput ,path);
                showFindedPath(actor2, path);
            }
        }
    }
}
```

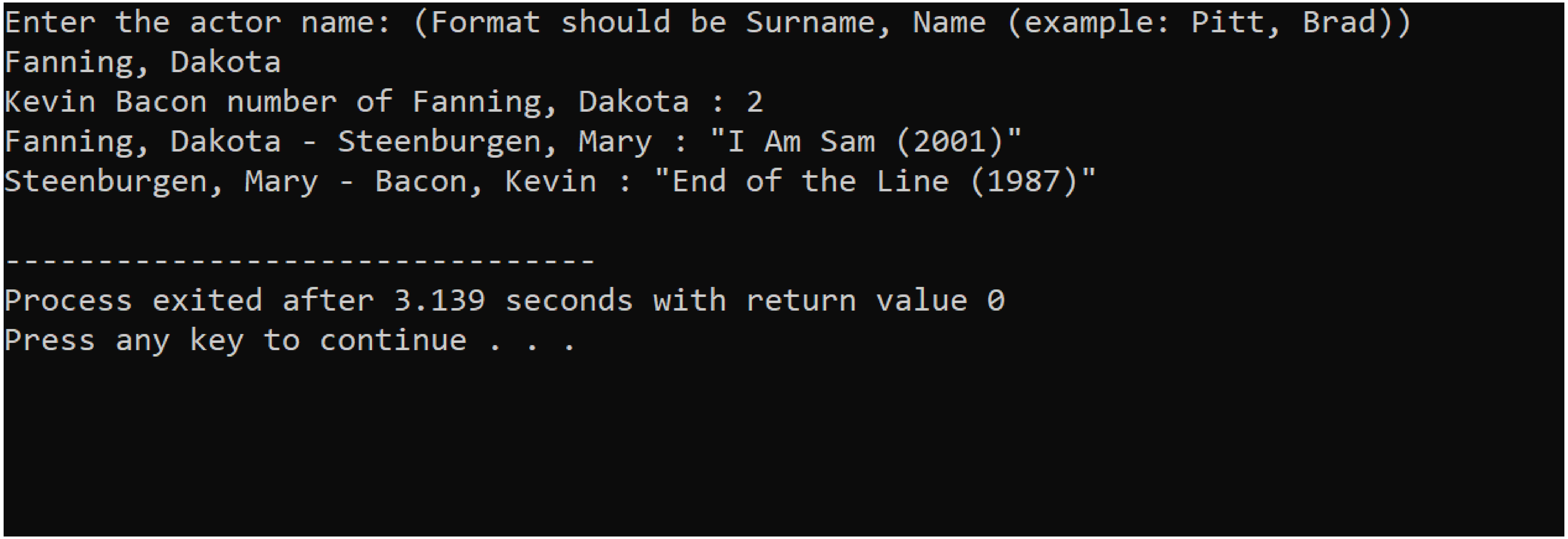

```
Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))
Streep, Meryl
Kevin Bacon number of Streep, Meryl : 1
Streep, Meryl - Bacon, Kevin : "River Wild, The (1994)"

--------------------------------
Process exited after 7.105 seconds with return value 0
Press any key to continue . . .
```



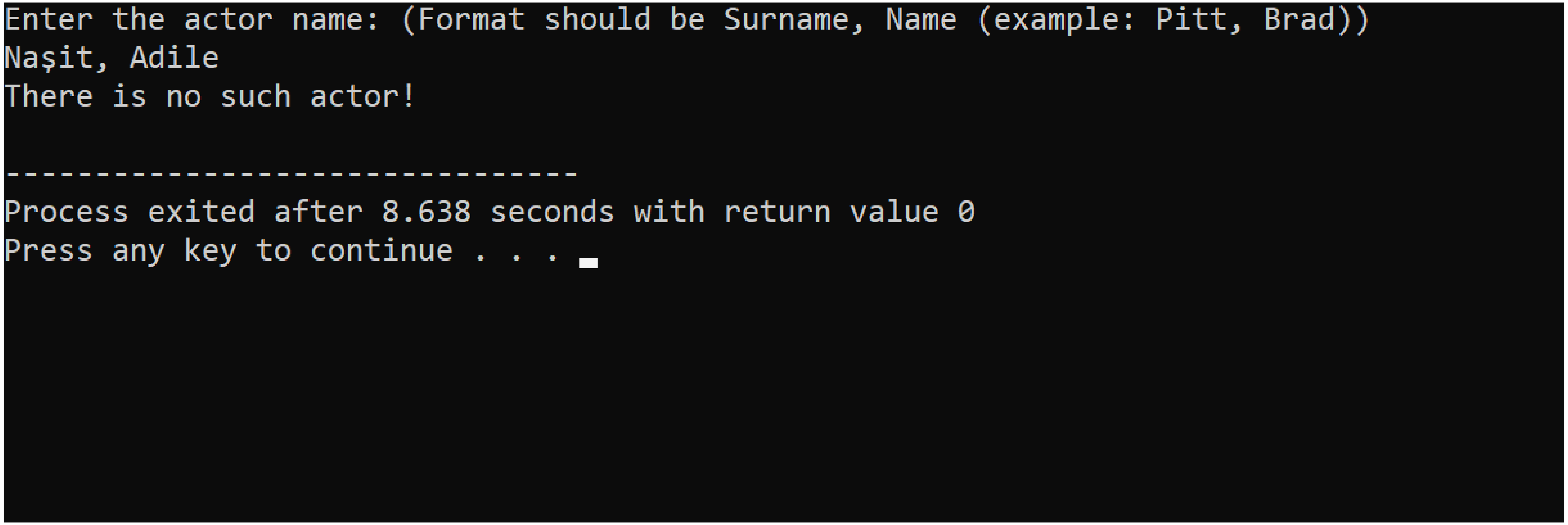
```
Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))
Cage, Nicolas
Kevin Bacon number of Cage, Nicolas : 2
Cage, Nicolas - McCann, Sean : "Trapped in Paradise (1994)"
McCann, Sean - Bacon, Kevin : "Air Up There, The (1994)"

--------------------------------
Process exited after 13.01 seconds with return value 0
Press any key to continue . . .
```



```
Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))
Samaha, Elie
Kevin Bacon number of Samaha, Elie : 3
Samaha, Elie - Carrere, Tia : "20 Dates (1998)"
Carrere, Tia - McCann, Sean : "Dogboys (1998)"
McCann, Sean - Bacon, Kevin : "Air Up There, The (1994)"

--------------------------------
Process exited after 2.659 seconds with return value 0
Press any key to continue . . .
```



```
Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))
Fanning, Dakota
Kevin Bacon number of Fanning, Dakota : 2
Fanning, Dakota - Steenburgen, Mary : "I Am Sam (2001)"
Steenburgen, Mary - Bacon, Kevin : "End of the Line (1987)"

--------------------------------
Process exited after 3.139 seconds with return value 0
Press any key to continue . . .
```



```
Enter the actor name: (Format should be Surname, Name (example: Pitt, Brad))
Naşit, Adile
There is no such actor!

--------------------------------
Process exited after 8.638 seconds with return value 0
Press any key to continue . . .
```