The connected world of Frank Ramsey

Group Project
Final Report
May 17th, 1999

Project supervisor: Professor D. F. Brailsford

Group members:

Steven Boxhall	**(skb97c)**
Alan Griffiths	**(apg97c)**
Manek Hakimjee	**(mhh97n)**
Thomas Muskett	**(tam97p)**
Anthony Truhlar	**(ajt97c)**

Final Group Report

Introduction

This project is entitled "The Connected World of Frank Ramsey". This gives only a small indication of what the finished program performs, and the design and implementation processes required in completing the project.

The project specification comes from the fusion of two diverse influences: the infamous "Oracle of Kevin Bacon" site at the University of Virginia, which has taken the Internet by storm; and Ramsey's theory of graphs. The former was a game written by two undergraduates, that has proved so successful that it has spawned numerous copycat sites and a considerable number of column inches. This original Kevin Bacon game takes inspiration from one of the ideas of Paul Erdos, who was one of the most prolific writers of mathematical papers this century. Erdos considered his published papers in a problem that he set some years ago: so-called "Erdos numbers". Erdos assigned himself an Erdos number of zero. He then considered each of the people whom he had co-authored papers with, and they were assigned an Erdos number of one. Those who had only co-authored a paper with a mathematician with an Erdos number of one were given an Erdos number of two. And so on.

The "Oracle of Kevin Bacon" updated and expanded the idea of Erdos numbers. The game originally came from the theory that the moderately successful, ageing Hollywood actor Kevin Bacon was the most connected star in Hollywood, meaning that he had starred with a wider range of individuals than anyone else. Consequently, the Erdos number became the "Bacon number". Actors who had starred with Kevin himself were assigned a Bacon number of one. Actors who had starred with individuals with a Bacon number of one were given a Bacon number of two, and so on. It was first proposed that any actor who didn't have a Bacon number of infinity (meaning that they cannot be linked to Kevin) would have a Bacon number of six or less. Since then, it has emerged that there are a handful of people have a Bacon number of seven, and two privileged individuals even have Bacon numbers of eight[1].

This project took inspiration from the "Oracle of Kevin Bacon", but threw a new spin onto the game, injecting the most famous theory published by mathematician Frank Ramsey. Ramsey's theory comes from a branch of combinatorial mathematics, and considers that in any set of disordered objects, any pattern of order can be found if the set is large enough. Paul Erdos interpreted this in the context of a dinner party. For example, how many guests would you have to invite to ensure that either three guests are strangers or three guests already know one another? It must be possible to make sure that the guests fit in with this rule as long as you invite enough people.

This example is actually quite simple to work out. However, first we must assume that knowing someone is mutual. Taking this as being true, imagine a person goes to a party

[1] Masud Zandbegleh and Leia Zanganeh

with five other people, and that they already know three of them. There's a chance that none of their friends have ever met each other before. If this is true though, there will be at least three mutual strangers, which solves Ramsey's problem. Maybe all three all know each other already. In this case, there will be at least three people who already know each other. Finally, there could be two of their friends that have met. This would still satisfy the rule, as if there are two who have met then the three people who know each other will be the original person and those two. From this, it is apparent that to have a dinner party with at least three mutual friends or strangers, six people need to be invited. Increasing the number of mutual friends or strangers to four would result in at least seventeen or eighteen guests being required at the party.

Our project specification was to produce a game based around generating "dinner parties" containing Kevin Bacon and various other actors. In this version of Ramsey's parties, "knowing someone" is interpreted as having directly co-starred with them. The data for this is taken from the Internet Movie Database, which is a global resource containing information about hundreds of thousands of actors, films and filmmakers. Bacon numbers crop up in this game, as they are required for ascertaining the success of a given party.

The rules governing our new "parties" are simple. Kevin Bacon must be one of the guests. He, and each of the other, user-specified guests, must have starred with only two of the actors present at the party. Also, each actor must be "seated" in such a way at Kevin's "table" that they are in-between their co-stars. Hence, the co-starring relationships can be thought to go around the outside of the table. Such a configuration actually forms an unsuccessful Ramsey party, so it was important to carefully choose the number of "guests" present so no cliques can form. The number chosen was five, as an analysis of the party shows that it is never guaranteed exactly what the co-starring relationship will be. The rest of the project specification was left open ended, allowing the group to develop their own ideas.

Our finished product contains the following components on the "Kevin Bacon's Dinner Parties" web site:

- Two different versions of the five-guest game: a Java version which represents parties graphically; and an HTML version, which gives text feedback;
- A "Hall Of Fame" page which securely allows the user to enter a successful party, which is then displayed on another page. Previous results are archived;
- A full explanation of the mathematics behind the game, complete instructions, and other background information;
- A "Bacon number" generator, where a user can enter an actor's name to query their specific Bacon number, and a page of statistics concerning this;
- Numerous links to the IMDb, through specific links, an input box at the base of each page, and ensuring that every actor name that appears on any page is linked the specific IMDb biography.

Behind the scenes, the following systems and programs were also developed:

- A C-program which searches through the entire IMDb and calculates every actor's Bacon number;
- Several shell-script tools to test co-starring relationships, etc;
- Several Perl scripts, which are used by the web pages, to check whether a given dinner party is successful or not, and give the appropriate feedback and error handling;
- Several algorithms for automated party searches;
- A Perl script to generate successful dinner parties automatically;
- Scripts to download the complete, updated IMDb, and update the Bacon number files from this.

The rest of this paper will deal with each aspect of the completed project in turn, focusing on design and implementation issues, as well as reflective thoughts as to how our system could be improved.

User Interface Implementation (front-end)

The user interface for “Kevin Bacon’s Dinner Parties” comprises of several sub-areas:

- Reading the user’s input for the game
- Outputting the results of the user’s attempt to enter a successful dinner party
- Providing instructions and information about the game
- Containing any other aspects of the project
- Dealing with links to other sites.

The final user interface mainly comprises of HTML, with one notable use of Java.

HTML Development The HTML interface is the part of the project that the user will have to use directly, consequently this makes it very important. Taking this into account we chose to develop this part of the project first. We knew that the final product needed to have a high level of usability and user satisfaction.

To make the implementation of the site as easy as possible we chose to create a blank template page, which could be used by all members of the group for page creation. To ensure a high level of usability we used Nielsen’s web-standard heuristic evaluation guidelines [1]. It was necessary to include a lot of links to IMDb in the design (see later in the report) and we were also keen to have a set of pages that would run on Netscape 1. For the second reason we omitted the use of frames and used tables instead, but in the end it became apparent that it would still be too difficult to make the template compatible with Netscape 1. In terms of the layout of the page we were forced to use absolute positioning, as the Java applet needed a certain amount of space. This sadly means that the page can become a bit difficult to read at very high resolutions.

The actual pages on the site are as follows:

- A title page, which only appears once
- Two versions of the game, one in HTML and one in Java
- A “Hall Of Fame” entry page, as well as another two pages that display previous entries
- Several pages (nine in total) describing the mathematics behind the game These are clearly linked to one another in “chapters”, and protect the novice user from too much detail by becoming progressively more complex
- Two pages of instructions, linked in the same way as the mathematics pages
- A page of general information about the game itself, and the development methods behind it
- A “Bacon number generator” page
- A page of statistics concerned with Bacon numbers.

Every page (although only the first mathematics and instruction pages) is linked to every other page, to ensure that the user is able to navigate the site as easily as possible. There are also some external links on the site: one to the IMDb; one to the “Oracle Of Kevin

Bacon" at the University of Virginia; and one to Kevin Bacon's specific biography page at the IMDb. At a late stage, it was decided to put a standard IMDb search box at the base of every page. This allows the user to search the database for any actor or film name from our site, whilst giving sufficient credit to IMDb.

Graphics have been kept to a minimum on the site to reduce download times, but without making the site look too boring. We also chose to specify all links from the root directory and not absolutely, this means that if the system is moved to another machine it should require less effort to get it up and working.

Overall we as a group are very happy with the design of the interface. The maths pages in particular required a large amount of work in order to find the right level of complexity. Sadly we didn't quite manage to make the template fully cross-browser compatible, but taking into account the technical limitations we got about as close as we could.

Play the game: HTML version The HTML version of the game input page was designed before its Java counterpart. The lack of flexibility of HTML means that the options available for this page were somewhat limited. It was therefore decided to have a diagrammatic representation of the table, displaying the positions where each actor inputted by the user are placed by labelling each apex on the table with a single letter. The actual input is taken by four uniform text-fields each labelled with a letter corresponding to the diagram. The input boxes are aligned in a two by two table. Initially, labels were horizontal to these text-fields, but due to problems with this layout in lower resolutions, they were moved to above their respective fields. There are also the standard "Submit" and "Clear" buttons, situated below the input fields. The whole input section is below the diagram. Above the diagram are some simple instructions about the required form of input, as well as a link to further instructions, and to the Java page. The feedback given to the user about the success of their party, although written in HTML, is outputted by a Perl script, and therefore will be covered later.

This HTML page can either be chosen to be used, or is forced if the browser being used is detected as being Java incompatible.

Play the game: Java version The Java version of the game input obviously had more flexibility than its HTML counterpart. The group felt that is was important to use the applet in a constructive rather than merely decorative manner. Concerns had already been raised over the form of output given by the game, which had to be constructive and increase playability. Therefore, it was decided that the applet should in some way improve the feedback given to the user about the success of their party.

The form of output required was decided to be graphical. It was planned that, after the user entered their party, the applet would send the query to a Perl script, the output of which would be parsed by the applet and then represented diagrammatically as a "table", with co-starring relationships shown as a line between two points. Obviously, a successful party would have every point connected to each of the two points on either

side of it, with no lines across the table. Somewhere in this design, links to each actor's biography page on the IMDb would be required.

The HTML page that activates the applet is extremely similar in design to the HTML version. At the top of the screen, there are some simple instructions, a link to the instruction page, and a link to the HTML version of the game. This maintains a consistency in design and layout, and allows easy navigation of the pages without the site becoming intimidating to the novice user.

Firstly, it was decided to tackle this multi-part problem with a single applet. The first, and perhaps the most awkward issue, was the layout. Initial attempts to align the text input fields with the points of a bitmap table were unsuccessful due to the somewhat quirky behaviour of Java's layout managers. Eventually, it was decided to group the input fields above the said bitmap (which, similar to the HTML version, comprised of a labelled table image), with buttons to submit and clear positioned below.

Two panels were used, one aligned at the top of the applet and one at the bottom, courtesy of BorderLayout. Each of these panels then used GridLayout to position the components. The upper components was a GridLayout(2,4): four TextFields and four Labels, positioned in such a way that the labels were horizontal to their associated text fields. The lower panel comprised of two Buttons of changing action (initially used to submit and clear), and a Choice box (used for actor links) which is initially hidden. In Internet Explorer 4, where there is a bug in the Java interpreter that means that Choice box cannot be hidden. Therefore, it is also disabled. These three components are positioned in a horizontal line, with a GridLayout(1,3).

To afford a level of pre-submission error trapping, only inputs where each one of the four fields has been filled in can be submitted. This is implemented by disabling the "Submit" button until all four TextFields have a length greater than zero. To avoid getting trapped in a loop, this is only checked when the mouse is moved. However, the average user will not even notice this.

When "Submit" is clicked on, the applet uses the input in each of the fields to generate a string to query an applet-specific Perl script. The applet then takes the output from this script, which takes the form of a binary string, into an array. This string of digits represents all the co-starring relationships between every actor in the party, and also tells the applet whether the party is successful, unsuccessful or has errors in its input. In the last case, the applet merely calls the HTML script that gives a text output of any errors in the user's input (for example, misspelling an actor's name, or attempting to enter the same actor's name twice).

If the user has found a successful party, a standard congratulatory bitmap is displayed, and the action and labelling of the two buttons are changed. "Submit" becomes "Enter Hall Of Fame", which displays the HTML page that allows a user to enter a successful party in the hall of fame, and "Clear" becomes "Read Instructions", in case the user has

no idea what is going on. In the case of an unsuccessful party, the applet comes into its own by providing a graphical input.

The graphics implementation is contained in the Paint() method, which behaves differently depending on the status of a global variable, AppletState. When the applet is in the "input" state, or when the user has found a successful party, this method merely displays a bitmap. However, when an unsuccessful party has been entered, a number of things happen. Firstly, a bitmap of a "blank" table is displayed. This image also contains a standard message telling the user that their party is unsuccessful. Secondly, lines are drawn to represent the co-starring relationships between each actor, as given by the aforementioned Perl script. The co-ordinates of these lines are specified absolutely and placed in an array. Then, where necessary, two lines one pixel away from one another are drawn onto the blank table. This is to make thicker, more easily identifiable lines. The applet has no concept of co-starring relationships; it merely makes a graphical representation of the output of the script. As well as this, the buttons are relabelled and their actions are changed, by using more global variables. "Submit" becomes "Try Again", and if this is selected, AppletState is reset to putting the applet into the "input" state, and the Repaint() method is called, resulting in the current bitmap being cleared and the original image being redisplayed. "Clear" becomes "Read Instructions", which simply displays the appropriate HTML pages.

An important aspect of this design is the Choice box, which allows links to be made between each of the four actors and their relevant IMDb pages. This box only becomes visible when either a successful or unsuccessful – but correctly entered – party has been entered. It uses the content of the TextFields to generate the IMDb URL, and for this reason, the TextFields have their isEditable property set to "false" during feedback. Initially, it was hoped that the actor's names could actually appear in the Choices box, but this proved unworkable, so instead the selectable items are merely "Actor A", "Actor B" etc. Using a StringBuffer, a short routine checks through the selected actor string using a loop, and replaces all non-alphanumeric characters with their ASCII code, preceded by a "%". A URL is then generated using this, and displayed accordingly.

Error trapping is included fully in the applet code. Hopefully, there is no instance where a user could generate an error, although if there is an exception, the applet's background colour is set to red. If the database or Perl script is down, there is simply no response from the applet, which is unfortunately not user friendly, but cannot really be solved from the applet's end.

There have been some strange effects with the interface between the applet and the corresponding Perl script. At times, it has almost appeared that the results are in some way being cached, as at one point in development, the applet would draw the same result regardless of the output of the Perl script. It was unclear where this problem was occurring, but considerable simultaneous work on the Perl script and the Java has resulted in the two interfacing correctly, seemingly all of the time.

In retrospect, the applet appears to have been designed and developed successfully and fully encompasses all of the design specifications that were applied to it. There are a few irritating quirks caused by different versions of Java. For example, the Choices box is positioned slightly differently under X-Windows than it is under Netscape, which appears to have no solution, although it is merely an aesthetic issue. The bug in Internet Explorer is also slightly unseemly, as the Choices box doesn't always hide and show when told to. This problem is improved by disabling it when it should be hidden, and enabling it when it should be shown. This does not affect the performance under Netscape.

On some systems, the applet takes a considerable amount of time to initialise. However, despite attempts to optimise the number of classes included in the code, this appears to have no straightforward solution.

Querying the Alternative Movies Database (interface link)

To implement the online Kevin Bacon game it was necessary to have a level to link the front-end (web interface) to the backend (movie database etc). It was the job of this level to take the four actor's names, as supplied by the front-end, and to execute the necessary queries on the backend to decide whether they constituted a successful Kevin Bacon dinner party. As this link was going to be triggered by web browsers it would consist of CGI scripts. When it came to deciding which language we would use for this, the choice was pretty easy, Perl. This language was designed to mimic and improve the standard UNIX tools such as awk and sed, but it has grown into the most widely used language for Internet programming. Its advanced string handling features and libraries specifically geared towards programming on the web, make it the easiest CGI scripting language by a long way.

In the project there were three areas where Perl scripts where needed:

- HTML version of the game
- Java applet version of the game
- Hall of Fame maintenance.

HTML version of the game This was the first script that we wrote to query the database, consequently it was where we made all the initial mistakes and found the best ways to do things.

In our naivety we originally tried to deal with the decoding of the data passed from the web browser ourselves. With all the special characters that are put into hex format it gets very difficult. Luckily we were not the first people to come across this difficulty and we were able to make use of a library file that had been written just for the purpose (`cgi-lib.pl`). Once the data had been received and decoded it was then a matter of processing it. This fell into two parts, firstly checks on the validity and integrity of the supplied data and secondly the actual tests to decide whether the party was a successful one.

As with all CGI programmers we were aware of the problems that can arise from remote users being able to execute processes on our server. The biggest problem is people who try to get your CGI script to do something malicious, by entering rogue data. The biggest security hole we found was the issue of back primes, as they have a traditional use in UNIX. The shell will attempt to execute any string passed to a command that is framed between back primes. Even with the reduced permissions given to CGI scripts it is still possible for files in temporary directories to be altered by CGI scripts. We had to implement a check on each actor's name to check for back primes before the data could be safely passed on to query the database.

To make the game slightly easier we also chose to give the player extra information on why unsuccessful parties failed. The checks we implemented were:

- Check that Kevin Bacon is not a specified actor
- Check that no actor is used more than once
- Check that all names supplied exist within the movie database.

When it comes to actually checking if a party is successful, the tests fall into two categories: the queries executed on the AMD (6) and searching Bacon number files for actors (4). Both these tests can be completed easily in Perl, thanks to its almost seamless integration with the shell. To query the movie database you simply execute the necessary command and the resulting data (if any) is piped back into the script. To search a file in Perl you open a filehandle to it and then iterate through it like you would an array. By this method the script was able to decide whether each actor had the appropriate Bacon number.

Having decided on the party's validity then it was just a matter of passing the information back to the player in HTML format. We achieved this through a library file of our own creation. It contained the necessary functions to allow us to output data using the same HTML template that all the webpages on the site are constructed with.

Java applet version This version of the game uses a separate CGI script to the HTML game, but a lot of the code is borrowed from it. All the checks on the party data are identical to that of the HTML version, but the data returned is slightly different. Instead of the heavily formatted HTML code it was just necessary to provide eleven binary bits to enable the applet to draw the appropriate graph to represent the party. However, it took a while to modify the code due to the applet needing the information in a different form to that which it is natively stored in the CGI script.

After the Bacon number generator this was the second hardest bit of the project to debug. As with any two components that link together to form a hopefully seamless link it is hard to work out which bit is going wrong if there is an error. The web browser we were using was occasionally caching results, especially after the problem was amplified when the CGI script had been edited. The problems were eventually solved and this version of the game became the showpiece of the website.

Hall of Fame This was the last bit of the site to be implemented and so we were able to use our experience gained from implementing the rest of the project. However, this part of the project was still had some problems. The first implementation of the Hall was, we discovered, impractical. It used a single CGI script carry out all required maintenance, such as adding new actors and checking that there were no duplications. We found though that CGI scripts just do not have the permissions to write permanent data. It was necessary to rethink out strategy, thanks to some helpful advice we implemented a new strategy.

The CGI script was just used for the temporary storage of the parties. Another script was written that was to be executed from the crontab. This used one of the group member's permissions and so was allowed to write to all group areas of the disk. It was this script's job to check that the party had not already been found and if not to add it to the Hall

along with its finder and the date. By running this script every hour the Hall is kept very up to date.

The third part of the Hall concerns its long-term maintenance, as the movie database is updated, parties that were valid will no longer be so and need removing from the Hall. We chose to have a second Hall of Fame for no longer valid parties. We implemented a third script, which is triggered after each update of the movie database. This script checks to see if any parties are no longer valid and switches them to the secondary Hall of Fame, then both Halls are regenerated from their new log files.

As the Hall of Fame grows it will become quite processor intensive to check all the parties in it, so it was necessary to streamline this script as much as possible. We deduced that it would not be necessary to apply all ten checks to each party, as the party had been successful certain properties must be true:

- Actor A will have BN1
- Actor B will have BN1
- A and C will have co-starred
- B and D will have co-starred
- C and D will have co-starred.

This cut the number of tests down to five, but the script was still quite slow. We eventually found that the operation that takes all the time is searching through the Bacon number two file (BN2), it is about 91,000 lines long. It occurred to us that in this instance it is not actually necessary to search through the BN2 file. We know that actors C and D can only have Bacon numbers of one or two. So, it is possible to search through the BN1 instead (only about 1,300 lines), and check for the actors in there. If they are not in that file then it is safe to assume that they are in BN2. Removing this time consuming operation greatly increased the speed of the script.

Movie Database System (backend)

During the analysis stage of the project, the group investigated what queries were necessary in order determine whether a given party was successful or not (see appendix B) and also how these queries would be carried out. The first option available to us was to use remotely located data, with a local cache. The Oracle of Kevin Bacon at the University of Virginia [2] contained the required Bacon number information and the International Movies Database (IMDb) [3] contained the required co-starring information. Faust was set up to query these sites on our behalf and return the results. However there were major problems with this system:

- The average time taken to return a query was unacceptably slow, especially considering the number of queries required for each party.
- There continually exist inconstancies between the data of the two sites. While IMDb is updated weekly, Virginia has remained static for some considerable time.

The second and by far the most attractive option open to us was to use a local database. This would mean that all information could be stored on Crilly and would give almost instantaneous access. The database in question was the Alternative Movies Database (AMD) [4]. This is maintained by the IMDb people and contains identical information. It consists of a series of text files that contain the data, and a collection of tools to access the data. The database can also be updated weekly from FTP sites around the world. There were two problems that needing solving though before we could use AMD:

- We would be implementing the system on a publicly accessible site, so we would need explicit permission from IMDb to use it.
- The Unix version of AMD would not compile on Crilly.

Thankfully both problems were solved in a reasonably short amount of time. We sent an email to IMDb asking their permission to use the database. Their Managing Director replied to our email (see appendix A), saying that we had their permission as long as we put a link to IMDb on every web page and every occurrence of an actor's name was linked to the appropriate filmography page at IMDb. The second problem was simply due to the fact that the AMD makefile was using the wrong compiler[2] and was easily fixed, once it had been discovered.

Implementing the System In order to make full use of the AMD two supporting systems had to be implemented. The first was a system to generate Bacon numbers for every actor in the database, and the second a maintenance system to look after all the movie data. With these systems in place we would have all the information we needed in one self-maintaining unit.

[2] It was trying to use the cc compiler rather than gcc.

Bacon numbers Rather than building a full database system from scratch for the Bacon numbers; we opted to develop a much simpler system consisting of text files. We planned eight files in total, `BN1.txt` to `BN8.txt`. Each one would contain the names of all the actors, in the AMD, with that bacon number. The actor's names would be in the same format as IMDb:

```
Surname, Forename(s)
```

The actor names would be placed in the file one per line – this has the major benefit of allowing exact line matching. Also, by using a simple file structure like this, it would be possible for other programs (such as the CGI scripts that form the interface between the front and back ends of the system) to query the files directly. This is more favourable than invoking the AMD query tools, which would incur an extra overhead.

We used the following procedure for calculating Bacon numbers using the AMD. First find all Kevin Bacon's co-stars (these people have a Bacon number of one), then find all of these actors' co-stars (these people have Bacon numbers of two). The method then propagates through the database until all possible actors have been linked to Kevin Bacon. To find an actor's co-stars it is necessary to generate a list of all the films the actor has starred in, then to generate a list of all the people who starred in those films.

The only problem with the aforementioned procedure is that most actors appear in more than one file. This is solved by searching through the lists and marking the first occurrence of each actor, all other occurrences of that actor are removed.

The Bourne shell prototype Initially a prototype of the Bacon number generator was constructed using the Bourne shell. This method was quick and dirty and was mainly done to generate some data, hence allowing other areas of the project to progress. This prototype only generated Bacon numbers up to and including two, because that is all the game needs, but the plan was to eventually extend this up to eight. The generator was completed at the end of December, and took an hour and a half to run.

The prototype consisted of two scripts, one called `generate`[3] and the other called `co-stars`. The co-stars script worked by taking an actor's name as an argument and returning the actor's entire co-star list. The main script was generate, this called the co-stars script on Kevin Bacon and then on subsequent actors to generate the Bacon number lists. Two further scripts, `bacon` and `mbacon`, were written. These combined to return the Bacon number of any specified actor by searching the lists. Their manual pages can be found in appendix D.

While providing some much-needed data fairly quickly and painlessly the system was by no means a long-term solution. The scripts were not very efficient, and despite being trimmed to run in only half an hour, were still too slow. To generate all the way to Bacon numbers of eight would have taken about two days. This would be unacceptable when the

[3] The source code for this and other sections of the report is available in appendix C.

AMD is being updated every seven days. Navigating through the complex tagging system employed by the AMD was also proving very difficult in the shell[4]. So it was decided that the proper version of the Bacon number generator should be implemented in C.

The Final Generate System The final solution implemented in C is by far the most complex code of the project, and warrants a report all to its self, but sadly we do not have the space. So a brief outline of the system will have to suffice.

Initially it was decided that the system, `fastgen` (source: `fastgen.c`), should implement the same basic algorithm as used by the prototype. As all the data manipulation was done in memory many of the standard Unix shell commands like grep, diff and sort had to be rewritten. This meant that new data structures had to be created and the whole process got very messy and complicated. The overheads incurred by doing everything in memory were also very large[5]. After a month of development it became clear that a drastic re-think was in order. Instead of using memory, files were used wherever possible. This meant that the standard Unix tools could used again. The change of plan meant that the algorithm had to be adjusted slightly.

Thankfully the huge effort required (two months of near solid work) to implement the Bacon number generator in C paid off. The final version was a vast improvement on the shell prototype. It can generate Bacon numbers up to two in just under five minutes, and go all the way up to eight in less than three hours! The drain on CPU and memory resources are also dramatically reduced. This means that it is possible to quickly and safely regenerate the Bacon number lists after each update of the AMD.

The maintenance system This currently consists of a script called `weekly_upd`, which can be run to update the local copy of the `AMD` and also keep the actor lists up to date. The script does everything from fetching the latest plain text data files from an IMDb mirror ftp site, regenerating the `AMD`, then generating the actor lists for the week using the new database, and archiving the previous week's lists. More details on `weekly_upd` can be found in the manual page (see appendix D).

[4] A list of all the problems encountered with using the AMD can be found in appendix F.
[5] At times blocks of memory in excess of 100Mb were required.

Searching for Successful Kevin Bacon Dinner parties

This is an area of the project that was unfortunately neglected due to time constraints, in fact the search was still running at the time of writing. We have had some interesting initial results though.

Our first attempts at writing search algorithms were based around, very basic, brute force techniques. In essence we had a script which went through the list of Bacon numbers and tried all the combinations. It was only when we realised how long each query would take, that it became apparent that the number of combinations was just too vast[6]. On Crilly it would have taken several years to come up with all the successful parties, by which time of course a lot of the data would be obsolete.

So, another method was needed to produce results in a reasonable amount of time. The method we eventually used came about by accident. I reasoned that most of the time is taken up by finding actors who have co-starred. So A-C and B-D are the difficult relationships to find. I was thinking that we needed a quicker way of finding these relationships. It just happened that Anthony was working on a program, for his Bacon number generator, which when given an actor returns all the actor's co-stars. Being written in C and searching the database files directly made it far quicker that anything we'd written in PERL. With this program it was possible to create an algorithm that would complete in a sensible amount of time (about two weeks).

The algorithm took the following form:

- Iterate through the BN1 file (A)
- For each actor, list all co-stars.
- Compile a list of all co-stars with BN2 (C).
- Iterate through this list
- For each actor, list all co-stars.
- Compile a list of all co-stars with BN2 (D).
- Iterate through this list
- For each actor, list all co-stars with BN1 (B).
- Check that A has not co-starred with B
- Check that B has not co-starred with C
- Check that A has not co-starred with D.

Any configuration of actors that successfully passes through this algorithm, is a successful Kevin Bacon dinner party.

The algorithm has currently been running for five days, and has iterated through about a quarter of the BN1 file. It has currently found fifty successful parties. We are however, not confident that it is catching all successful parties. The C program that we are using was never used for its intended purpose and was consequently never fully tested,

[6] The number of combinations is approximately 8×10^{15}

combined with errors in the database it is possible that we could have missed a lot of parties. It is also possible that parties could be found it clusters around well-known actors. In the parties we've found by accident during the process of the project, Tom Cruise appears quite often. It is possible that a search around him and other similar actors could reveal a large number of parties.

Reflective comments

In summary, we have achieved the aim of creating a Kevin Bacon dinner parties game. Admittedly, the game is not quite as specified in the initial proposal; i.e. that the player enters the name of one actor and the system then picks three other actors to try and form a successful dinner party of five including Kevin Bacon. This idea was put on hold at the initial formal meetings due to its complexity. At that stage we had decided (with the backing of Professor Brailsford) to develop a variant that would be more playable and also easier to implement, then if this could be implemented, we could move onto progressively more difficult versions of the game until we had developed the one specified. As it happens, despite the perhaps naïve optimism of the group, the initial case proved challenging enough to implement, and as a result we were unable to progress beyond this. Having said that however, we have laid the foundations for variants on this theme to be implemented, with the successful development of tools like `fastgen` to allow one to generate all the lists of actors connected to Kevin Bacon. Combine that with the initial work on automated searches for successful dinner parties, and you have a firm basis on which a game similar to that initially specified could be implemented.

The initial specifications of the game – i.e. that it should be playable in virtually any browser, and that there should be a Java version for Java-enabled browsers to enhance the game by giving graphical feedback of the various co-starring relationships in a given set of actors; have both been met. Also, the initial desire that it should be possible to put the game on the School's web server (Pat) has also come to fruition, since all of the code is portable and can be dropped onto another system and run with only a small number of modifications (no need to change every script or web page, etc).

The group believes that the project has been a success. We have worked well as a team and have developed code that interfaces very well indeed. There were of course features that we were not able to implement, one of the most lamented being a 'Star Links' system similar to that at the Oracle. Such a system would have allowed a player to view the exact connectivity (in terms of common films) from a given actor to Kevin Bacon. At the end of the day, we had simply run out of time and were unable to implement this feature. It would of course been a nice touch to our site.

We have been fortunate to have such a modular project. As one will have noticed, the project was broken down into three key stages, each of which was allocated to a different member. Having just one person in charge of the development of a given system and with a clearly defined interface between each of the parts of the system, has helped us to develop code reasonably quickly and well, with far fewer problems that we had expected (there were still a few problems when it came to interfacing the main sections).

We would not want to give the impression of the project being completely smooth running, as this was not always the case. The project didn't make any real progress until around December, which has put extra pressure on members of the group throughout the rest of the project. This has probably contributed to our current situation where we have had ideas but been unable to implement them. Also documentation proved to be a handful

at times. Groups that are at the start of their project would be well advised to document everything as they go along, since not keeping on top of this can result in a nasty backlog of documentation. This may not be looked at until the preparation of the reports, by which time it may be too late to do sufficient justice to all the hard work that group members have put in to the project.

Thankfully, as a group we were reasonably well organised and had minutes taken at each formal meeting which enabled group members to easily keep track of what was going on within the project, and also what their responsibilities were. We are also a reasonably well skilled group in terms of programming ability with each of us having little trouble specialising in the language in which out area of the project was being developed in. The combination of both of these factors has lead to us being a fairly independent group that has been able to produce a good product with relatively little assistance required.

Hyperlinks

[1] - http://www.useit.com

[2] - http://www.virginia.edu/oracle

[3] - http://www.imdb.com

[4] - http://us.imdb.com/interfaces

Appendix A

Date: Sat, 2 Jan 1999 17:20:47 +0000 (GMT)
From: Col Needham <cn@imdb.com>
To: Thomas Andrew Muskett <tam97p@Cs.Nott.AC.UK>
Subject: Re: use of downloadable interface
In-Reply-To: <36798886.8A42AC7F@imdb.com>
Message-ID: <Pine.BSF.4.02.9901021716101.28575-100000@cn.imdb.com>
Organization: The Internet Movie Database Ltd.
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Sender: Col Needham <cn@cn.imdb.com>

On Thu, 17 Dec 1998, Thomas Andrew Muskett wrote:

> I am part of a team of second year undergraduates at The University of
> Nottingham in the UK. We are working on a project similar to Virginia's
> "Oracle of Bacon", based on creating Kevin Bacon "dinner parties" based
> on his co-starring relationships.
>
> I am writing to enquire whether we would be able to use the downloadable
> version of IMDB on our server. We would also be interested in using
> your provided search algorithms. The database would not be accessible
> to the user, but instead would be used to verify co-starring relationships
> and "Bacon numbers".
>
> Although we are planning to put our project on the web and make it
> accessible to the public, it will not be for any commercial gain whatsoever.

Okay, sounds like a good project. No problems using the data and software
as long as we are properly credited on your pages. This means an acknowledgment
either on or linked directly off the main page plus also all names/titles
must be linked to their IMDb details pages (as at the Oracle of Bacon site).

Let me know if any of the above needs to be clarified. Also please contact us
when the project is up and running so we can take a look.

Good luck!

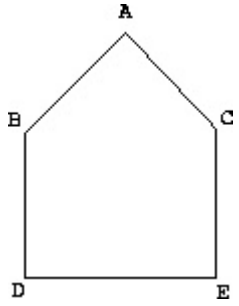
Col Needham
Managing director,
The Internet Movie Database Ltd.

Winner of 1997 & 1998 best film site Webby awards http://www.imdb.com/
160,000+ movies 1892-2000 The site that means movies

Appendix B

The Method

Assuming that the relationships are represented by a pentagon thus:



Each of the nodes represents actors, where node A is of course Kevin Bacon. The system currently proposed requires a minimum of ten checks to ensure that the given dinner party is either successful or unsuccessful. These checks are:

1. B must have a Bacon number of 1.
2. C must have a Bacon number of 1.
3. D must have a Bacon number of 2.
4. E must have a Bacon number of 2.
5. B must have co-starred with D.
6. C must have co-starred with E.
7. D must have co-starred with E.
8. B must NOT have co-starred with C.
9. B must NOT have co-starred with E.
10. D must NOT have co-starred with C.

Appendix C

The following pages contain the source code for the project.

```
#! /usr/bin/perl

# CGI script to redirect a browser to the IMDb page for a
# specified actor, by Alan Griffiths

# Get libraries
require "cgi-lib.pl";
require "output-lib.pl";

# Get input
&ReadParse;

$actor = $in{'actor'};

# Test actor vailidity
open (TEST, "/proj/gp-dfb2/moviedb-3.5/bin/list -cast \"$actor\" | ") || die;
$line = <TEST>;
close (TEST);

# Return appropriate result
if (! $line) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i><strong>$actor</strong> is not a valid actor name.</i><br><br>";
    print "<a href=\"/bacon/link.html\">Please try again.</a>\n";
    &printfoot (STDOUT);
}

else {
    open (URL, "./name2url \"$actor\" | ") || die;
    $line = <URL>;
    close (URL);
    print "Location: $line\n\n";
}

exit;
```

```
#! /usr/bin/perl

# Kevin Bacon Dinner Party Program by Alan Griffiths
# CGI script to decide if four actors make a successful party
# Input in the form a=Smith,+Will&b=...

# Actors names are stored in $namex, where x is their letter in the table
# The results of the tests are stored in $result[num]
# where num is the number of the test
# All results are initially set to zero

# N.B. all \‘ references are actually just ‘ - this change is necessary to
# ensure correct syntax highlighting

# Get libraries for parsing and output
require "cgi-lib.pl";
require "output-lib.pl";

# Set constants
$BN_PATH = "/proj/gp-dfb2/numbers/data";     # Path to Bacon no files
$AMD_PATH = "/proj/gp-dfb2/moviedb-3.5/bin"; # Path to AMD binaries

# Parse input
&ReadParse;

# Assign eash actor to a scaler
$namea = $in{’a’};
$nameb = $in{’b’};
$namec = $in{’c’};
$named = $in{’d’};

# Get link back address
$link = $ENV{’HTTP_REFERER’};

# Derive link back
if (! $link) {
    $link = "/bacon/game.html";
}

else {
    $link = "/bacon/html.html";
}


# Check for illegal characters
if ($namea =~ /\‘/ || $nameb =~ /\‘/ || $namec =~ /\‘/ || $named =~ /\‘/) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i>Illegal character(s) detected</i><br><br>\n";
    print "<a href=\"$link\">Please try again.</a>\n";
    &printfoot (STDOUT);
    exit;
}

# Replace %2C with a comma
#$namea =~ s/%2C/,/;
#$nameb =~ s/%2C/,/;
#$namec =~ s/%2C/,/;
#$named =~ s/%2C/,/;


# Set all results to zero
for ($i = 1; $i < 11; $i++) {
    $result[$i] = 0;
}

# Check that AMD is online
open (TEST, "$AMD_PATH/list -cast \"Sting\" | ") || die;
```

```
$line = <TEST>;
close (TEST);
if (! $line) {
    &printhead (STDOUT, "Error");
    print "<h1>Internal Error</h1>\n";
    print "<i>The movie database is currently off-line.";
    print " This is probably because it is being updated.";
    print " This does not usually take very long so please try again";
    print " in a bit. Sorry.</i>\n";
    &printfoot (STDOUT);
    exit;
}

# Check with AMD that actor names are valid
$chk1 = 1;
$chk2 = 1;
$chk3 = 1;
$chk4 = 1;

# Check A
open (CHK1, "$AMD_PATH/list -cast \"$namea\" | ") || die;
$line = <CHK1>;
close (CHK1);
if (! $line) {
    $chk1 = 0;
}

# Check B
open (CHK2, "$AMD_PATH/list -cast \"$nameb\" | ") || die;
$line = <CHK2>;
close (CHK2);
if (! $line) {
    $chk2 = 0;
}

# Check C
open (CHK3, "$AMD_PATH/list -cast \"$namec\" | ") || die;
$line = <CHK3>;
close (CHK3);
if (! $line) {
    $chk3 = 0;
}

# Check D
open (CHK4, "$AMD_PATH/list -cast \"$named\" | ") || die;
$line = <CHK4>;
close (CHK4);
if (! $line) {
    $chk4 = 0;
}

if ($chk1 + $chk2 + $chk3 + $chk4 != 4) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</H1>\n";
    if ($chk1 == 0) {
        print "<i><strong>$namea</strong> is not a vaild actor name</i><BR><BR>\n";
    }
    if ($chk2 == 0) {
        print "<i><strong>$nameb</strong> is not a valid actor name</i><BR><BR>\n";
    }
    if ($chk3 == 0) {
        print "<i><strong>$namec</strong> is not a valid actor name</i><BR><BR>\n";
    }
    if ($chk4 == 0) {
        print "<i><strong>$named</strong> is not a valid actor name</i><BR>\n";
    }
    print "<BR><a href=\"$link\">Please try again.</a>\n";
    &printfoot (STDOUT);
```

```
    exit;
}


# Check that Kevin Bacon is not one of the actors
if ($namea =~ m"Bacon, Kevin" || $nameb =~ m"Bacon, Kevin" || $namec =~ m"Bacon, Kevin" |
| $named =~ m"Bacon, Kevin") {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i><strong>Kevin Bacon</strong> cannot be a guest at his own party!</i><br><br
>\n";
    print "<a href=\"$link\">Please try again.</a>\n";
    &printfoot(STDOUT);
    exit;
}

# Check that duplicate actor names have not been provided
if ($namea =~ /^$nameb$/ || $namea =~ /^$namec$/ || $namea =~ /^$named$/) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i>You have entered <strong>$namea</strong> more than once.</i>\n";
    print "<a href=\"$link\">Please try again.</a>\n";
    &printfoot (STDOUT);
    exit;
}

if ($nameb =~ /^$namec$/ || $nameb =~ /^$named$/) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i>You have entered <strong>$nameb</strong> more than once.</i>\n";
    print "<a href=\"$link\">Please try again.</a>\n";
    &printfoot (STDOUT);
    exit;
}

if ($namec =~ /^$named$/) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i>You have entered <strong>$namec</strong> more than once.</i>\n";
    print "<a href=\"$link\">Please try again.</a>\n";
    &printfoot (STDOUT);
    exit;
}

# Tests
# (1)&(2) Do A and B have Bacon numbers of one?
open (FD, "$BN_PATH/BN1.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namea$/) {
        $result[1] = 1;
    }
    if ($line =~ /^$nameb$/) {
        $result[2] = 1;
    }
}

close (FD);

# Do C and D have Bacon numbers of two?
open (FD, "$BN_PATH/BN2.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namec$/) {
        $result[3] = 1;
    }
    if ($line =~ /^$named$/) {
        $result[4] = 1;
```

```
    }
}

close (FD);

# (5) Has A starred with C, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;

$line = <FD>;

if ($line) {
    $result[5] = 1;
}


# (6) Has A starred with B, Right = No
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$nameb\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if (! $line) {
    $result[6] = 1;
}

close (FD);

# (7) Has A starred with D, Right = No
open (FD, "/proj/gp-dfb2/moviedb-3.5/bin/list -cast \"$namea\" -cast \"$named\" | grep -v
 \'(T*VG*)\' | ") || die;
$line = <FD>;

if (! $line) {
    $result[7] = 1;
}

close (FD);

# (8) Has B starred with C, Right = No
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if (! $line) {
    $result[8] = 1;
}

close (FD);

# (9) Has B starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[9] = 1;
}

close (FD);

# (10) Has C starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namec\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[10] = 1;
}
```

```
close (FD);


# Check to see if we have a successful dinner party
$outcome = 1;

for ($i = 1; $i < 11; $i++) {
    if ($result[$i] == 0) {
        $outcome = 0;
    }
}


# Output results as HTML
&printhead (STDOUT, "Results");
print "<H1>Results</H1><P>\n";

if ($outcome == 1) {
    print "<i>Congratulations, you have found a successful Kevin Bacon dinner party!</i><
br><br>\n";

    print "Why not enter your name in the <a href=\"/bacon/enter.html\">Hall of Fame</a>?
\n";
}

else {
    print "<i>Sorry, you have not found a successful Kevin Bacon dinner party. Due to the
 following reasons.</i><br><br>\n";
    if ($result[1] == 0) {
        print "<strong>$namea</strong> does not have a Bacon number of one.<br><br>\n";
    }
    if ($result[2] == 0) {
        print "<strong>$nameb</strong> does not have a Bacon number of one.<br><br>\n";
    }
    if ($result[3] == 0) {
        print "<strong>$namec</strong> does not have a Bacon number of two.<br><br>\n";
    }
    if ($result[4] == 0) {
        print "<strong>$named</strong> does not have a bacon number of two.<br><br>\n";
    }
    if ($result[5] == 0) {
        print "<strong>$namea</strong> and <strong>$namec</strong> have not starred toget
her.<br><br>\n";
    }
    if ($result[6] == 0) {
        print "<strong>$namea</strong> and <strong>$nameb</strong> have starred together.
<br><br>\n";
    }
    if ($result[7] == 0) {
        print "<strong>$namea</strong> and <strong>$named</strong> have starred together.
<br><br>\n";
    }
    if ($result[8] == 0) {
        print "<strong>$nameb</strong> and <strong>$namec</strong> have starred together.
<br><br>\n";
    }
    if ($result[9] == 0) {
        print "<strong>$nameb</strong> and <strong>$named</strong> have not starred toget
her.<br><br>\n";
    }
    if ($result[10] == 0) {
        print "<strong>$namec</strong> and <strong>$named</strong> have not starred toget
her.<br><br>\n";
    }

print "<a href=\"$link\">Please try again</a>.<br>\n";
}
```

```
print "<br><h2>Actor Links to IMdB</h2>\n";

# Print actor A and link to IMdB
open (URL1, "./name2url \"$namea\" | ") || die;
$line = <URL1>;
close (URL1);
print "<A HREF=\"$line\">$namea</A>";

# Print actor B and link ti IMdB
open (URL2, "./name2url \"$nameb\" | ") || die;
$line = <URL2>;
close (URL2);
print "<BR><A HREF=\"$line\">$nameb</A>";

# Print actor C and link to IMdB
open (URL3, "./name2url \"$namec\" | ") || die;
$line = <URL3>;
close (URL3);
print "<BR><A HREF=\"$line\">$namec</A>";

# Print actor D and link to IMdB
open (URL4, "./name2url \"$named\" | ") || die;
$line = <URL4>;
close (URL4);
print "<BR><A HREF=\"$line\">$named</A>";

&printfoot (STDOUT);

exit;
```

```
#! /bin/sh
# Determine the Bacon number of a given actor
# This checks for Bacon numbers of 1 through 8
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 28/12/1998
# Modified for 3->8 on 29/03/99
# Modified to add portability code 09/05/99

# Define variables
AMDPATH=/proj/gp-dfb2/moviedb-3.5/bin
LISTDIR=/proj/gp-dfb2/numbers/data
SEARCHLEV=8

# Check arguments
if test $# -lt 1
then
    echo "Incorrect number of arguments!"
    echo "usage: bacon <actor/actress> [-p listpath | -d amdpath]"
    exit 0 # abort
fi

# Copy contents of $1 to temporary variable
ACTOR=$1

# set path variables according to arguments
while
    test -n "$2"
do
    case `echo $2 | sed -e "s/-//"` in
    p ) LISTDIR=$3;;
    d ) AMDPATH=$3;;
    s ) SEARCHLEV=$3;;
    * ) echo "Invalid flag \"$2\""; break;;
    esac
    shift
    shift
done

# Check for KB
if `test "$ACTOR" = "Bacon, Kevin"`
then
    echo "0"
    exit 0                          # Found, therefore terminate
fi

# Ensure actor is valid
if test `$AMDPATH/list -cast "$ACTOR" | wc -l` -eq 0
then
    echo "Invalid actor name!"
    exit 0
fi

# Determine location of actor
N=1
while test "$N" -le $SEARCHLEV
do
    if `grep -x "^$ACTOR\$" $LISTDIR/BN$N.txt > /dev/null`
    then
        echo $N
        exit 0
    fi
    N=`expr $N + 1`
done

echo "infinity"

exit 0 # exit
```

```
#! /usr/bin/perl

# Bacon number determination program by Anthony J. Truhlar (11/04/1999)
# Re-uses code from Alan Griffiths' Dinner parties program.

# Actor name is stored in $name
# The results of the tests are stored in $result
# $result is initially set to infinity

# Amended to use Alan's output library

# N.B. all \' references are actually just ' - this change is necessary to
# ensure correct syntax highlighting

# URL parsing, using cgi-lib
require "cgi-lib.pl";
require "output-lib.pl"; # use standard HTML header output library

&ReadParse;

# Assign actor to a scaler
$name = $in{'a'};

# Replace %2C with a comma
$name =~ s/%2C/,/;
# End of URL parsing

# Untaint data (remove shell specific chars)
if ($name =~ /\'/ ) {
  &printhead (STDOUT, "Find the Bacon number of an actor");
  print "<center>\n";
  print "<i>The supplied string contains an illegal character, please try again.</i></BR>
\n";
  print "</center>\n";
  &printfoot (STDOUT);
  exit;
}

# Check that actor name is valid (using AMD)
open (CHK, "/proj/gp-dfb2/moviedb-3.5/bin/list -cast \"$name\" | ") || die;
$line = <CHK>;
close <CHK>;
if (! $line) {
  &printhead (STDOUT, "Find the Bacon number of an actor");
  print "<center>\n";
  print "<i><strong>$name</strong> is not a valid actor name</i></BR>\n";
  print "</center>\n";
  &printfoot (STDOUT);
  exit;
}

# Replace brackets with escaped versions (for complete AMD compatability)
$name =~ s/\(/\\(/;
$name =~ s/\)/\\)/;

# Set results to infinity
$result = infinity;

# Set file number counter to 1
$i = 1;

# Deal with Kevin Bacon
if ($name =~ m"Bacon, Kevin" ) {
    $result = 0;
    $i = 9; # don't bother searching files
}

# Tests
```

```
# Step through files and determine the Bacon number of the actor
for (; $i < 9; $i++) {

    open (FD, "/proj/gp-dfb2/numbers/data/BN$i.txt") || die;

    while ($line = <FD>) {
        if ( $line =~ /^$name$/ ) {
            $result = $i;
            $i = 9;
        }
    }

    close (FD);

}

# Replace escaped brackets with normal ones (for aesthetic purposes)
$name =~ s/\\\(/\(/;
$name =~ s/\\\)/\)/;

# Output results as HTML
&printhead (STDOUT, "Find the Bacon number of an actor");

# Print actor and link to IMDb
open (URL1, "./name2url \"$name\" | ") || die;
$line = <URL1>;
close (URL1);

print "<CENTER>\n";
print "<BR><H3><A HREF=\"$line\">$name</A> has a Bacon number of $result.</H3>";
print "</CENTER>\n";

# Output footer of HTML document
print "<P>\n";
&printfoot (STDOUT);
exit;
```

```
#! /usr/bin/perl

# Bacon number determination program by Anthony J. Truhlar (11/04/1999)
# Returns just the number of the actor wrapped up in a short HTML header

# Actor name is stored in $name
# The results of the tests are stored in $result
# $result is initially set to infinity

# Returns number of actor, else -1 for infinity, or -2 for invalid name
# -3 is returned for an illegal character

# N.B. all \‘ references are actually just ‘ - this change is necessary to
# ensure correct syntax highlighting

# URL parsing, using cgi-lib
require "cgi-lib.pl";

&ReadParse;

# Define routine for outputting numbers
sub returnvalue {

  my $value = shift;
  my $out = shift;

  print $out "Content-type: text/html\n\n";
  print $out "<html>\n<body>\n";
  print $out $value;
  print $out "\n</body>\n</html>\n";

  return 1;

}

# Assign actor to a scaler
$name = $in{‘a’};

# Replace %2C with a comma
$name =~ s/%2C/,/;
# End of URL parsing

# Untaint data (remove shell specific chars)
if ($name =~ /\‘/ ) {
  returnvalue(-3, STDOUT);
  exit;
}

# Check that actor name is valid (using AMD)
open (CHK, "/proj/gp-dfb2/moviedb-3.5/bin/list -cast \"$name\" | ") || die;
$line = <CHK>;
close <CHK>;
if (! $line) {
  returnvalue(-2, STDOUT);
  exit;
}

# Set results to infinity
$result = -1;

# Set file number counter to 1
$i = 1;

# Deal with Kevin Bacon
if ($name =~ m"Bacon, Kevin" ) {
    $result = 0;
    $i = 9; # don’t bother searching files
}
```

```
# Replace brackets with escaped versions (for complete AMD compatability)
$name =~ s/\(/\\(/;
$name =~ s/\)/\\)/;

# Tests
# Step through files and determine the Bacon number of the actor
for (; $i < 9; $i++) {

    open (FD, "/proj/gp-dfb2/numbers/data/BN$i.txt") || die;

    while ($line = <FD>) {
        if ($line =~ /^$name$/) {
            $result = $i;
            $i = 9; # break out of loop asap
        }
    }

    close (FD);

}

# Replace escaped brackets with normal ones (for aesthetic purposes)
$name =~ s/\\\(/\(/;
$name =~ s/\\\)/\)/;

# Output results as HTML
returnvalue($result, STDOUT);

exit;
```

```
#! /bin/sh
# Shell script to list all actors that have co-starred with a given actor
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 26/12/1998

# Test the number of arguments
if test $# -ne 1
then
    echo "Incorrect number of arguments!"
    echo "Usage: co-stars <actor/actress>"
    exit 0 # abort
fi

# Define variables
TMP1=/tmp/costars.$$_1.tmp # list of films actor has starred in (no spaces)
TMP2=/tmp/costars.$$_2.tmp # co-starring actors, no dups removed, unsorted etc.

# Generate list of films that actor has starred in
list -cast "$1" | grep -v '(T*VG*)' | tr " " "_" > $TMP1

# For each film; list the cast
for FILM_1 in `cat $TMP1`
do
   FILM_2=`echo $FILM_1 | tr "_" " " | stripdesc`
   title -t "$FILM_2" | stripdesc >> $TMP2
done

# Sort, removing duplicates
sort -uf $TMP2

# Remove all temporary files
rm -f $TMP1 $TMP2

# Exit and return to shell
exit 0
```

```
#! /usr/bin/perl

# Second Year Group Project: gp-dfb2
# PERL script to handle cgi requests from Hall of Fame input page
# Output goes to three files in $temp_path
# The files are removed after the update script has executed
#
# *p_pend.log* contains the successful parties. The first line contains an
# integer which represents the number of parties found since last update. Each
# party has both possible arrangements listed. Hence there are (2x+1) lines
# in p_pend at any one time (where x is number of parties).
#
# *n_pend.log* contains the names of the finders. They are in the same order
# as the parties in p_pend.log. First line is number of names. Hence x+1
# lines.
#
# *ip_pend.log* contains the ip addresses of the finders. Also in same order.
# x+1 lines.
#
# Actor names are separated by '&'.

# N.B. all \' references are actually just ' - this change is necessary to
# ensure correct syntax highlighting

# Get libraries
require "cgi-lib.pl";
require "output-lib.pl";

# Set paths
$TEMP_PATH = "/var/local/ramsey-temp";       # Path to temp dir
$BN_PATH = "/proj/gp-dfb2/numbers/data";     # Path to Bacon no dir
$AMD_PATH = "/proj/gp-dfb2/moviedb-3.5/bin"; # Path to movie database

# Read/Decode input
&ReadParse;

# Put input into local variables
$namea = $in{'a'};
$nameb = $in{'b'};
$namec = $in{'c'};
$named = $in{'d'};
$person = $in{'per'};
$ip = $ENV{'REMOTE_ADDR'};

# Limit name string to 25 chars
$person = substr ($person, 0, 25);


# Check for illegal characters
if ($namea =~ /\'/ || $nameb =~ /\'/ || $namec =~ /\'/ || $named =~ /\'/ || $person =~ /\
'/) {
    &printhead (STDOUT, "Error");
    print "<h1>Error in input data</h1>\n";
    print "<i>Illegal character(s) detected</i><br><br>\n";
    print "Please try again\n";
    &printfoot (STDOUT);
}

# Get current system date
open (DT, "date +'%d/%m/%Y' | ") || die;
$date = <DT>;
chomp ($date);
close (DT);

# Check to make sure that a successful dinner party has been entered
# Check validity of input
# Set error status to 0
$error = 0;
```

```
# Check that Kevin Bacon is not one of the actors
if ($namea =~ m"Bacon, Kevin" || $nameb =~ m"Bacon, Kevin" || $namec =~ m"Bacon\
, Kevin" || $named =~ m"Bacon, Kevin") {
    $error = 1;
}

# Check that no actor is used more than once
if ($namea =~ m'$nameb' || $namea =~ m'$namec' || $namea =~ m'$named' || $nameb =~ m'$nam
ec' || $nameb =~ m'$named' || $namec =~ m'$named') {
    $error = 1;
}

# Tests
# (1)&(2) Do A and B have Bacon numbers of one?
open (FD, "$BN_PATH/BN1.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namea$/) {
        $result[1] = 1;
    }
    if ($line =~ /^$nameb$/) {
        $result[2] = 1;
    }
}

close (FD);

# (3)&(4) Do C and D have Bacon numbers of two?
open (FD, "$BN_PATH/BN2.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namec$/) {
        $result[3] = 1;
    }
    if ($line =~ /^$named$/) {
        $result[4] = 1;
    }
}

close (FD);

# (5) Has A starred with C, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;

$line = <FD>;

if ($line) {
    $result[5] = 1;
}

# (6) Has A starred with B, Right = No
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$nameb\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if (! $line) {
    $result[6] = 1;
}

close (FD);

# (7) Has A starred with D, Right = No
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;
```

```
if (! $line) {
    $result[7] = 1;
}

close (FD);

# (8) Has B starred with C, Right = No
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if (! $line) {
    $result[8] = 1;
}

close (FD);

# (9) Has B starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[9] = 1;
}

close (FD);

# (10) Has C starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namec\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[10] = 1;
}

close (FD);

# Check to see if we have a successful dinner party
$outcome = 1;

for ($i = 1; $i < 11; $i++) {
    if ($result[$i] == 0) {
        $outcome = 0;
    }
}

# Exit if party is not successful
# HTML: Inform user of an error
if ($outcome == 0 || $error == 1) {
    &printhead (STDOUT, "Error");
    print "<H1>Hall of Fame</H1>\n";
    print "<I>Sorry you did not enter a successful Kevin Bacon dinner party.</I><BR><BR>\
n";
    &printfoot (STDOUT);
    exit;
}

# Else continue
# Create both successful permitations
$party1 = $namea . "&" . $nameb . "&" . $namec . "&" . $named;
$party2 = $nameb . "&" . $namea . "&" . $named . "&" . $namec;

# Check if log file(s) already exist
if (-e "$TEMP_PATH/p_pend.log") {
# Read in current logs
    open (LOG_R, "$TEMP_PATH/p_pend.log") || die;
```

```
    @p_pend = <LOG_R>;
    close (LOG_R);

    open (LOG_R, "$TEMP_PATH/n_pend.log") || die;
    @n_pend = <LOG_R>;
    close (LOG_R);

    open (LOG_R, "$TEMP_PATH/ip_pend.log") || die;
    @ip_pend = <LOG_R>;
    close (LOG_R);

# Find current total number of parties
    $total = $p_pend[0];
    chomp ($total);

    $found = 0;

    for ($i = 1; $i <= ($total * 2); $i++) {
        if ($p_pend[$i] =~ /^$party1$/) {
            $found = 1;
            last;
        }
    }

    if ($found == 0) {

        $p_pend[($total*2)+1] = $party1 . "\n";
        $p_pend[($total*2)+2] = $party2 . "\n";

        $n_pend[$total+1] = "$person - $date\n";

        $ip_pend[$total+1] = "$ip\n";

        $total++;
        $p_pend[0] = $total . "\n";
        $n_pend[0] = $total . "\n";
        $ip_pend[0] = $total . "\n";

        open (LOG_W1, ">$TEMP_PATH/p_pend.log") || die;
        open (LOG_W2, ">$TEMP_PATH/n_pend.log") || die;
        open (LOG_W3, ">$TEMP_PATH/ip_pend.log") || die;

        for ($i = 0; $i <= ($total*2); $i++) {
            print LOG_W1 $p_pend[$i];
        }

        for ($i = 0; $i <= $total; $i++) {
            print LOG_W2 $n_pend[$i];
            print LOG_W3 $ip_pend[$i];
        }

        close (LOG_W1);
        close (LOG_W2);
        close (LOG_W3);
    }
}

# Else create log files
else {
# Create log files for this session
    open (LOG_W1, ">$TEMP_PATH/p_pend.log") || die;
    open (LOG_W2, ">$TEMP_PATH/n_pend.log") || die;
    open (LOG_W3, ">$TEMP_PATH/ip_pend.log") || die;

# Set all totals to 1
    print LOG_W1 "1\n";
    print LOG_W2 "1\n";
    print LOG_W3 "1\n";
```

```
# Insert data
    print LOG_W1 "$party1\n";
    print LOG_W1 "$party2\n";
    print LOG_W2 "$person - $date\n";
    print LOG_W3 "$ip\n";

# Close files
    close (LOG_W1);
    close (LOG_W2);
    close (LOG_W3);
}

# Set permissions to allow update script to erase them after each update
system ("chmod a+w $TEMP_PATH/p_pend.log");
system ("chmod a+w $TEMP_PATH/n_pend.log");
system ("chmod a+w $TEMP_PATH/ip_pend.log");


# HTML: Inform of probable success
&printhead (STDOUT, "Hall of Fame");

print "<h1>Hall of Fame</h1>\n";
print "<i>Congratulations, your name has been put on a prelimanary list. If no one has al
ready found this party then your name will be added to the Hall of Fame within the next h
our.</i><br>\n";

&printfoot (STDOUT);

exit;
```

```
/* ------------------------------------------------------------------------ *
 *          Program to generate the Bacon number files from 1 to 8          *
 *            By Anthony Jaroslav Truhlar (21/02/1999-07/04/1999)           *
 *                      Email: ajt97c@cs.nott.ac.uk                         *
 *                   Group Email: gp-dfb2@cs.nott.ac.uk                     *
 * ------------------------------------------------------------------------ *
 */

/* N.B. This code will only work with the AMD tools in $USER's path */

#define _GNU_SOURCE

#include <errno.h>
#include <ctype.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

/* Declare global constants */
#define FALSE 0
#define TRUE 1
#define FILENAME_LENGTH 100
#define TEMP_FN_LENGTH 14
#define LINE_LENGTH 80
#define BUFFER_LENGTH 240
#define AMD_OUT_MAX_SIZE 65536
#define NULL_CHAR '\0'
#define SHELL "/bin/sh"

static char *progname; /* program name needed in all functions */
volatile sig_atomic_t terminate_in_progress = 0;

char *allocate(size_t size) { /* Safe malloc checking for available VM */

  register char *value = (char *) malloc(size);
  if ( value == NULL ) {
    (void) fprintf(stderr, "FATAL error in this process (ID = %d): Virtual memory exhaust
ed!\n", (int) getpid());
    exit (EXIT_FAILURE);
  }
  return value;

}

FILE *open_stream(char *dest, const char *access_mode, int pipe) {

  FILE *stream;

  errno = 0; /* don't output anything to stderr, we'll do that */

  stream = (pipe) ? popen( dest, access_mode ):fopen( dest, access_mode );
  if ( stream == NULL ) {
    (void) fprintf( stderr, "%s: Unrecoverable error opening %s; %s.\n", progname, dest,
strerror(errno) );
    exit (EXIT_FAILURE); /* can't read file - abort execution */
  }

  return stream;

}

void delete_file(char *filename) {
```

```
  errno = 0; /* don't output anything on stderr, we'll do that */

  if ( unlink( filename ) == -1 ) {
    (void) fprintf(stderr, "%s: Cannot delete file %s; %s.\n", progname, filename, strerr
or(errno));
  }

  return; /* return under all conditions - not fatal */

}

void execute(char *shellcommand) {

  int status;
  pid_t pid;

  pid = fork(); /* fork off subprocess and return process ID or error status */

  errno = 0;

  if ( pid == (pid_t) 0 ) {
    /* This is the child process, therefore execute the shell command */
    (void) execl( SHELL, SHELL, "-c", shellcommand, NULL );
    _exit (EXIT_FAILURE);
  } else {
    if ( pid < (pid_t) 0 ) {
      /* Unable to fork off a new process, report failure */
      status = -1;
    } else {
      /* Wait for child to complete, -1 indicates a failure in the child */
      if ( waitpid( pid, &status, 0 ) != pid ) status = -1;
    }
  }

  if ( status == -1 ) {
    (void) fprintf( stderr, "%s: Unable to execute shell command \"%s\"; %s!\n", progname
, shellcommand, strerror(errno));
    exit (EXIT_FAILURE);
  } else {
    return;
  }

}

void terminate_signal( int signal ) {

  char *cleanupcommand = allocate(LINE_LENGTH);
  char *pid_s = allocate(6);

  /* Protect against further errors/other signals whilst in signal handler */
  if( terminate_in_progress ) (void) raise( signal );
  terminate_in_progress = 1;

  /* Clean up */
  (void) strcpy( cleanupcommand, "rm -f /tmp/*" );
  (void) sprintf( pid_s, "%d", (int) getpid() );
  (void) strcat( cleanupcommand, pid_s );
  (void) strcat( cleanupcommand, "*" );
  execute( cleanupcommand );
  free( cleanupcommand );
  free( pid_s );

  /* Re-raise signal, which since blocked will be handled in the default way */
  (void) raise(signal);

}

void films(char *actor, char *films, char *db_path) {
```

```
  /* Allocate memory, and initialise streams */
  char *command = allocate(BUFFER_LENGTH);
  char *linebuf = allocate(BUFFER_LENGTH);
  char *command_cpy = command;
  FILE *cmdout;               /* stream for output from AMD toola */
  char *commandcpy, *linebufcpy;

  /* Keep track of start of arrays */
  commandcpy = command;
  linebufcpy = linebuf;

  /* Nullify strings */
  *command = NULL_CHAR;
  *linebuf = NULL_CHAR;
  *films = NULL_CHAR; /* ensure it's free for use */

  /* Construct AMD command for given actor */
  (void) strcpy( command, db_path );
  if( *(db_path+(strlen(db_path)-2)) != '/' ) (void) strcat( command, "/" );
  (void) strcat( command, "list -cast \"" );
  (void) strcat( command, actor );
  (void) strcat( command, "\"" );

  /* Execute command and capture output */
  cmdout = open_stream( command_cpy, "r", TRUE );
  while( fgets( linebuf, BUFFER_LENGTH, cmdout ) != NULL ) {
    /* only copy cinema films */
    if ( ( strstr( linebuf, "(TV)" ) == NULL ) && ( strstr( linebuf, "(V)" ) == NULL ) &&
 ( strstr( linebuf, "(VG)" ) == NULL ) ) {
      (void) strcat( films, linebuf );
    }
  } /* copy lines into memory */
  pclose( cmdout );

  /* Deallocate memory */
  free(commandcpy);
  free(linebufcpy);
  return;

} /* end of function: films */

void stars(char *movie, char *actors, char *db_path) {

  /* Allocate memory, and initialise streams */
  char *command = allocate(BUFFER_LENGTH);
  char *linebuf = allocate(BUFFER_LENGTH);
  char *command_cpy = command;
  FILE *cmdout;               /* stream for output from AMD toola */
  char *commandcpy, *linebufcpy;

  /* Keep track of start of arrays */
  commandcpy = command;
  linebufcpy = linebuf;

  /* Nullify strings */
  *command = NULL_CHAR;
  *linebuf = NULL_CHAR;
  *actors = NULL_CHAR;

  /* Construct AMD command for given actor */
  (void) strcpy( command, db_path );
  if( *(db_path+(strlen(db_path)-2)) != '/' ) (void) strcat( command, "/" );
  (void) strcat( command, "title -t \"" );
  (void) strcat( command, movie );
  (void) strcat( command, "\"" );

  /* Execute command and capture output */
```

```
  cmdout = open_stream( command_cpy, "r", TRUE );
  while( fgets( linebuf, BUFFER_LENGTH, cmdout ) != NULL ) {
    (void) strcat( actors, linebuf );
  } /* copy lines into memory */
  pclose( cmdout );

  /* Demallocte memory */
  free(commandcpy);
  free(linebufcpy);
  return;

} /* end of fn: stars */

int isroman(char ch) {

  return ( ch == 'I' || ch == 'X' || ch == 'V' );

} /* end of fn: isroman */

void stripdesc(char *tostrip, char *stripped, int films) {

  int round = FALSE, square = FALSE, nl = 0; /* nl - keep track of nesting level */
  int sql = 0;
  int brkt = (films) ? FALSE:TRUE; /* flag for year/(I etc) tags */
  char *temp = stripped;

  /* scan through memory and remove junk */
  do {

    if ( *tostrip == '(' ) nl++;

    if ( *tostrip == '[' ) sql++;

    /* only year and actor number information can be bracketed */
    if ( *tostrip == '(' && (!isdigit(*(tostrip+1))) && (!isroman(*(tostrip+1))) && brkt
) {
      round = TRUE;
      stripped -= ( ! square ) ? 1:0; /* remove uneccessary spaces before bracketing */
      stripped += ( nl >= 2 ) ? 1:0; /* nested brackets aren't preceeded by spaces */
    }

    if ( *tostrip == '[' ) {
      square = TRUE;
      stripped -= ( ! round ) ? 2:0;
      stripped += ( sql >= 2 ) ? 2:0;
    }

    /* Deal with characters that have special significance to the shell ($) */
    if ( *tostrip == '$' && !round && !square ) {
      *stripped++ = '\\';
      *stripped++ = '$';
      tostrip++;
    }

    if ( *tostrip == ''' && !round && !square ) {
      *stripped++ = '\\';
      *stripped++ = ''';
      tostrip++;
    }

    if ( ! round && ! square && *tostrip != ']' ) *stripped++ = *tostrip;

    if ( *tostrip == ')' && nl > 0 ) nl--;

    if ( *tostrip == '\n' && nl > 0 ) {
      nl = 0;
      round = FALSE;
      *stripped++ = '\n';
```

```
    } /* Deal with untermined nesting of parentheses */

    if ( *tostrip == ']' && sql > 0 ) sql--;

    if ( *tostrip == '\n' && sql > 0 ) {
      sql = 0;
      square = FALSE;
      *stripped++ = '\n';
    } /* Same use for square brackets as above */

    if ( *tostrip == '\n' && films ) brkt = FALSE; /* only turn off at eol for films, don
't allow for actors */

    if ( *tostrip == ')' && *(tostrip+1) != ')' && nl == 0 ) {
      round = FALSE;
      brkt = TRUE;
    }

    if ( *tostrip == ']' && *(tostrip+1) != ']' && sql == 0 ) square = FALSE;

  } while( *tostrip++ != NULL_CHAR );

  stripped = temp;
  return;

} /* end of fn: stripdesc */

int bacon(int min, int max, int optimise, char *path, char *db_path) {

  /* Allocate memory for filenames */
  char *prevlvlfn = allocate(FILENAME_LENGTH);
  char *currlvlfn = allocate(FILENAME_LENGTH);
  char *tmpfile = allocate(TEMP_FN_LENGTH);
  char *films1 = allocate(TEMP_FN_LENGTH);
  char *films2 = allocate(TEMP_FN_LENGTH);
  char *actor1 = allocate(TEMP_FN_LENGTH);
  char *actor2 = allocate(TEMP_FN_LENGTH);
  char *diff1 = allocate(TEMP_FN_LENGTH);
  char *diff2 = allocate(TEMP_FN_LENGTH);
  char *filmfiles = allocate(BUFFER_LENGTH);
  char *filmfilescpy = allocate(BUFFER_LENGTH);

  /* Allocate memory for commands */
  char *shellcommand = allocate(BUFFER_LENGTH);

  /* Allocate memory for title & list output */
  char *out = allocate(AMD_OUT_MAX_SIZE);
  char *out2 = allocate(AMD_OUT_MAX_SIZE);

  char *num = allocate(3);
  char *token;
  FILE *prevlvl, *currlvl, *tmp;
  char *linebuf = allocate(BUFFER_LENGTH);

  char *outcpy = out;
  char *out2cpy = out2;

  int i, j;
  int first = TRUE;

  *out = NULL_CHAR;
  *out2 = NULL_CHAR;
  *diff1 = NULL_CHAR;
  *diff2 = NULL_CHAR;

  errno = 0;

  for( i = min; i <= max; i++ ) {
```

```
/* Construct filenames for BN lists */
*prevlvlfn = NULL_CHAR;
*currlvlfn = NULL_CHAR;
(void) strcpy( prevlvlfn, path );
if ( *(path+(strlen(path)-2)) != '/' ) (void) strcat( prevlvlfn, "/" );
(void) strcat( prevlvlfn, "BN" );
(void) sprintf( num, "%d", i-1 );
(void) strcat( prevlvlfn, num );
(void) strcat( prevlvlfn, ".txt" );
(void) strcpy( currlvlfn, path );
if ( *(path+(strlen(path)-2)) != '/' ) (void) strcat( currlvlfn, "/" );
(void) strcat( currlvlfn, "BN" );
(void) sprintf( num, "%d", i );
(void) strcat( currlvlfn, num );
(void) strcat( currlvlfn, ".txt" );

/* Construct temporary filenames  */
*films1 = NULL_CHAR;
*films2 = NULL_CHAR;
*actor1 = NULL_CHAR;
*actor2 = NULL_CHAR;
films1 = tempnam( "/tmp", "films" );
films2 = tempnam( "/tmp", "films" );
actor1 = tempnam( "/tmp", "actor" );
actor2 = tempnam( "/tmp", "actor" );

(void) strcpy( filmfilescpy, filmfiles );
first = TRUE;

/* Construct list of films from previous level file */
prevlvl = open_stream( prevlvlfn, "r", FALSE );
tmp = open_stream( films1, "w", FALSE );
while( fgets( linebuf, BUFFER_LENGTH, prevlvl ) != NULL ) {
  *(linebuf+(strlen(linebuf))-1) = NULL_CHAR;
  films(linebuf,out,db_path);
  stripdesc(out,out2,TRUE);
  (void) fputs(out2,tmp);
}
fclose(tmp);
fclose(prevlvl);

/* Construct sort command for films list */
*shellcommand = NULL_CHAR;
(void) strcpy( shellcommand, "sort -uf " );
(void) strcat( shellcommand, films1 );
(void) strcat( shellcommand, " > " );
(void) strcat( shellcommand, films2 );

/* Sort list of films removing duplicates */
execute( shellcommand );

/* Remove unsorted films list */
delete_file(films1);

/* Remove films that have been referenced at previous levels */
if( optimise && ((max - i) < (max - 1)) ) {
  token = (char *) strtok( filmfilescpy, "$" );
  while ( token ) {

    /* Construct temporary filenames for this iteration */
    if ( !first ) {
      diff1 = diff2;
    } else {
      diff1 = films2;
      first = FALSE;
    }
    diff2 = tempnam( "/tmp", "diff" );
```

```
      /* Construct and execute shell commands */
      *shellcommand = NULL_CHAR;
      (void) strcpy( shellcommand, "diff " );
      (void) strcat( shellcommand, token );
      (void) strcat( shellcommand, " " );
      (void) strcat( shellcommand, diff1 );
      (void) strcat( shellcommand, " | grep \">\" | sed -e \"s/> //\" > " );
      token = (char *) strtok( NULL, "$" );
      if ( !token  ) {
        films2 = tempnam( "/tmp", "films" );
        (void) strcat( shellcommand, films2 );
      } else {
        (void) strcat( shellcommand, diff2 );
      }
      execute( shellcommand );

      /* Remove remaining temporary files from this iteration */
      delete_file(diff1);
    }

    first = TRUE; /* reset first flag for next iteration */

  } /* only do if optimise is flagged */

  /* Generate list of actors from films list */
  tmp = open_stream( films2, "r", FALSE );
  currlvl = open_stream( actor1, "w", FALSE );
  while( fgets( linebuf, BUFFER_LENGTH, tmp ) != NULL ) {
    *(linebuf+(strlen(linebuf))-1) = NULL_CHAR;
    stars(linebuf,out,db_path);
    stripdesc(out,out2,FALSE);
    (void) fputs(out2,currlvl);
  }
  fclose(tmp);
  fclose(currlvl);

  /* Remove sorted films list if optimised not flagged - not needed now */
  if ( !optimise ) delete_file(films2);

  /* Construct sort command for films list */
  *shellcommand = NULL_CHAR; /* nullify */
  (void) strcpy( shellcommand, "sort -uf " );
  (void) strcat( shellcommand, actor1 );
  (void) strcat( shellcommand, " > " );
  (void) strcat( shellcommand, actor2 );

  /* Sort temporary file, ready for removal of exiting BN's */
  execute( shellcommand );

  /* Remove unsorted actors list */
  delete_file(actor1);

  *diff1 = NULL_CHAR;
  *diff2 = NULL_CHAR;

  /* Remove existing BN's */
  for( j = i; j >= 1; j-- ) {

    /* Construct temporary filenames for this iteration */
    if ( j != i ) {
      diff1 = diff2;
    } else {
      diff1 = actor2;
    }
    diff2 = tempnam( "/tmp", "diff" );

    /* Construct and execute shell commands */
```

```
      *shellcommand = NULL_CHAR;
      (void) strcpy( shellcommand, "diff ");
      (void) strcat( shellcommand, path );
      if ( *(path+(strlen(path)-2)) != '/' ) (void) strcat( shellcommand, "/" );
      (void) strcat( shellcommand, "BN" );
      (void) sprintf( num, "%d", j-1 );
      (void) strcat( shellcommand, num );
      (void) strcat( shellcommand, ".txt " );
      (void) strcat( shellcommand, diff1 );
      (void) strcat( shellcommand, " | grep \">\" | grep -v \"[ ]\\{3\\}\" | grep -v \"^>
 \\-*\\$\" | sed -e \"s/> //\" > " );
      if ( j == 1 ) {
        (void) strcat( shellcommand, currlvlfn );
      } else {
        (void) strcat( shellcommand, diff2 );
      }
      execute( shellcommand );

      /* Remove remaining temporary files from this iteration */
      if ( j == 1 ) delete_file(actor2);
      if ( j != i ) delete_file(diff1);

    }

    /* Make note of film list filename from this iteraion */
    (void) strcat( filmfiles, films2 );
    (void) strcat( filmfiles, "$" );

  } /* end loop */

  (void) strcpy( filmfilescpy, filmfiles );

  /* Remove any remaining temporary files (from optimised run) */
  if ( optimise ) {
    token = (char *) strtok( filmfilescpy, "$" );
    while ( token ) {
      delete_file( token );
      token = (char *) strtok( NULL, "$" );
    }
  }

  /* Deallocate memory */
  free(prevlvlfn);
  free(currlvlfn);
  free(tmpfile);
  free(films1);
  free(films2);
  free(actor1);
  free(actor2);
  free(diff1);
  free(diff2);
  free(filmfiles);
  free(filmfilescpy);
  free(shellcommand);
  free(outcpy);
  free(out2cpy);
  free(num);
  free(linebuf);
  return EXIT_SUCCESS;

}

/* ================================ MAIN ================================ */

/* Options:
   ========
   No flags - generate all lists from BN1 to BN8
   -a actor - specify an actor other than K.B (e.g. generate Connery numbers)
```

```
   -o       - optimise generation; uses slightly more space (rmdups on films)
   -p path  - search for and generate the files in the specified directory
   -s n m   - generates a set of files from BNn.txt to BNm.txt (assumes all
              other files that are needed are present).
   -i p q r - generates the individual files BNp.txt, BNq.txt and BNr.txt ...
   -d path  - specify a path for the AMD tools
*/

int main(int argc, char **argv) {

  /* Declare variables */
  char *shortname = allocate(LINE_LENGTH);
  char *fullname = allocate(LINE_LENGTH);
  char *filename = allocate(FILENAME_LENGTH);
  char *actor = allocate(LINE_LENGTH);
  char *db_path = allocate(LINE_LENGTH);
  char *path = allocate(LINE_LENGTH);
  char *set = allocate(LINE_LENGTH);
  char *individual = allocate(LINE_LENGTH);
  char *token;
  FILE *file;
  int optimise = FALSE;
  int aflag = FALSE, dflag = FALSE, pflag = FALSE, sflag = FALSE, iflag = FALSE;
  int min = 1, max = 8;
  int c, tmp;

  /* Set up signal handlers */
  (void) signal( SIGINT, terminate_signal );
  (void) signal( SIGHUP, terminate_signal );
  (void) signal( SIGTERM, terminate_signal );
  (void) signal( SIGPIPE, terminate_signal );
  (void) signal( SIGABRT, terminate_signal );

  /* Set default path to current directory */
  (void) strcpy( path, "." );
  (void) strcpy( db_path, "." );

  /* Strip path from argv[0] to get program_invocation_short_name */
  (void) strcpy( fullname, argv[0] );
  token = (char *) strtok( fullname, "/" );
  (void) strcpy( shortname, token );
  while( token ) {
    (void) strcpy( shortname, token );
    token = (char *) strtok( NULL, "/" );
  }

  progname = shortname; /* make program name available to other functions */

  opterr = FALSE; /* Do NOT print option parsing errors to stderr */
  errno = 0;

  /* Parse command line options */
  while ((c = getopt(argc, argv, "a:d:i:op:s:")) != -1) {

    switch(c) {

      case 'a':
        (void) strcpy( actor, optarg );
        aflag = ( strstr( actor, "Bacon, Kevin" ) ) ? FALSE:TRUE;
        break;

      case 'd':
        *db_path = NULL_CHAR;
        (void) strcpy( db_path, optarg );
        dflag = TRUE;
        break;

      case 'i':
```

```
        if( iflag == 2 ) {
          (void) strcat( individual, "," );
          (void) strcat( individual, optarg );
        } else {
          (void) strcpy( individual, optarg );
        }
        iflag = 2;
        break;

      case 'o':
        optimise = TRUE;
        break;

      case 'p':
        *path = NULL_CHAR;
        (void) strcpy( path, optarg );
        pflag = TRUE;
        break;

      case 's':
        (void) strcpy( set, optarg );
        sflag = TRUE;
        break;

      case '?':
        if (isprint (optopt)) {
          if ( optopt == 'a' || optopt == 'i' || optopt == 's' || optopt == 'p' ) {
            (void) fprintf(stderr,"%s: Option '-%c' has a mandatory parameter missing!\n"
,shortname,optopt);
          } else {
            (void) fprintf(stderr,"%s: Invalid option: '-%c'.\n",shortname,optopt);
          }
        } else {
          (void) fprintf(stderr,"%s: Unknown option character '\\x%x'.\n",shortname,optop
t);
        }

        return EXIT_FAILURE;

    }

  }

  /* initialise variables & files */
  /* ============================ */

  /* construct low level file if not readable */
  (void) strcpy( filename, path );
  if( *(path+(strlen(path)-2)) != '/' ) (void) strcat( filename, "/" );
  (void) strcat( filename, "BN0.txt" );

  file = open_stream( filename, "w", FALSE );
  if ( aflag ) {
    (void) fprintf( file, "%s\n", actor );
  } else {
    (void) fprintf( file, "%s\n", "Bacon, Kevin" );
  }
  fclose(file);

  /* generate any individual files */
  if ( iflag ) {
    token = (char *) strtok( individual, "," );
    while( token ) {
      tmp = atoi(token);
      if ( tmp > 0 ) (void) bacon( tmp, tmp, FALSE, path, db_path );
      token = (char *) strtok( NULL, "," );
    }
  }
```

```
  /* generate any sets of files */
  if ( sflag ) {
    token = (char *) strtok( set, "," );
    min = atoi(token);
    token = (char *) strtok( NULL, "," );
    max = atoi(token);
    if ( min > 0 && max >= min ) (void) bacon( min, max, optimise, path, db_path );
  }

  /* if neither i or s flags set, do BN1-8 */
  if ( !iflag && !sflag ) (void) bacon( min, max, optimise, path, db_path );

  /* clean up - deallocate memory, etc. */
  free(shortname);
  free(fullname);
  free(filename);
  free(actor);
  free(path);
  free(db_path);
  free(set);
  free(individual);

  return EXIT_SUCCESS;

}
```

```
#! /bin/sh
# Generate the lists of actors with given Bacon numbers using the AMD
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 26/12/1998
# Routine to extract BN1s replaced by diff command, 22/03/1999
# Grep commands added to remove films that can appear at the start of lists
# (a problem with AMD), and also the seperating line. - 03/04/1999

# Define Variables
KB="Bacon, Kevin"
BACON_1=/proj/gp-dfb2/numbers/data/BN1u.txt
BACON_2=/proj/gp-dfb2/numbers/data/BN2u.txt
TMP1=/tmp/generate.$$_1.tmp # Actors with BN of 1
TMP2=/tmp/generate.$$_2.tmp # As above but with spaces changed to underscores
TMP3=/tmp/generate.$$_3.tmp # Preliminary list of actors with BN of 2
TMP4=/tmp/generate.$$_4.tmp # As above but sorted and with duplicates removed

##############################################################################
# Act appropriately according to arguments                                   #
# generate -1 : produce list of actors with a Bacon Number of 1 only #
# generate -2 : produce list of actors with a Bacon Number of 2 only #
# generate    : Produce both lists                                           #
##############################################################################

case ‘echo $1 | sed -e "s/-//"‘ in
    1 ) co-stars "$KB" > $BACON_1; exit 0;;
    2 ) ;;
    [3-8] ) echo "Sorry, this feature has not yet been implemented."; exit 0;;
    * ) co-stars "$KB" > $BACON_1;;
esac

#
# Generate list of actors with a Bacon number of 2
#

# If list of actors with BN1 exists, use it; otherwise create a temorary one
if ! ‘cp $BACON_1 $TMP1 2> /dev/null‘
then
     co-stars "$KB" > $TMP1
fi

# For each actor list their co-stars
cat $TMP1 | tr " " "_" > $TMP2
for ACTOR_1 in ‘cat $TMP2‘
do
  ACTOR_2=‘echo $ACTOR_1 | tr "_" " "‘
  co-stars "$ACTOR_2" >> $TMP3
done

# Sort removing duplicates
sort -uf $TMP3 > $TMP4

# Remove those that have a Bacon Number of 1
diff $TMP1 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/> //" >
 $BACON_2

# Remove all temporary files
rm -f $TMP1 $TMP2 $TMP3 $TMP4

# Exit
exit 0
```

```
#! /bin/sh
# Generate the lists of actors with given Bacon numbers using the AMD
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 22/03/1999
# Based on overhauled shell-prototype to generate BN1 & BN2
# Grep commands added to remove films that can appear at the start of lists
# (a problem with AMD), and also the seperating line. - 03/04/1999

# Define Variables
KB="Bacon, Kevin"
BACON_1=/proj/gp-dfb2/numbers/data/BN1.txt
BACON_2=/proj/gp-dfb2/numbers/data/BN2.txt
BACON_3=/proj/gp-dfb2/numbers/data/BN3.txt
BACON_4=/proj/gp-dfb2/numbers/data/BN4.txt
BACON_5=/proj/gp-dfb2/numbers/data/BN5.txt
BACON_6=/proj/gp-dfb2/numbers/data/BN6.txt
BACON_7=/proj/gp-dfb2/numbers/data/BN7.txt
BACON_8=/proj/gp-dfb2/numbers/data/BN8.txt
TMP1=/tmp/gen3to8.$$_1.tmp
TMP2=/tmp/gen3to8.$$_2.tmp
TMP3=/tmp/gen3to8.$$_3.tmp
TMP4=/tmp/gen3to8.$$_4.tmp
TMP5=/tmp/gen3to8.$$_5.tmp
TMP6=/tmp/gen3to8.$$_6.tmp
TMP7=/tmp/gen3to8.$$_7.tmp
TMP8=/tmp/gen3to8.$$_8.tmp
TMP9=/tmp/gen3to8.$$_9.tmp

##########################################################################
# Act appropriately according to arguments                               #
# gen3to8 -3 : produce list of actors with a Bacon Number of 3 only  #
# gen3to8 -4 : produce list of actors with a Bacon Number of 4 only  #
# gen3to8 -5 : produce list of actors with a Bacon Number of 5 only  #
# gen3to8 -6 : produce list of actors with a Bacon Number of 6 only  #
# gen3to8 -7 : produce list of actors with a Bacon Number of 7 only  #
# gen3to8 -8 : produce list of actors with a Bacon Number of 8 only  #
# gen3to8    : Produce all lists                                         #
##########################################################################

case `echo $1 | sed -e "s/-//"` in
    3 ) NUM=3;;
    4 ) NUM=4;;
    5 ) NUM=5;;
    6 ) NUM=6;;
    7 ) NUM=7;;
    8 ) NUM=8;;
    * ) NUM=0;;
esac

#
# Generate list of actors with Bacon numbers of 3
#

if test $NUM -eq 3 -o $NUM -eq 0
then

    # Ensure BN2.txt exists first
    if ! test -r $BACON_2
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_2."
        exit 0
    fi

    # For each actor list their co-stars
    cat $BACON_2 | tr " " "_" > $TMP1
    for ACTOR_1 in `cat $TMP1`
    do
        ACTOR_2=`echo $ACTOR_1 | tr "_" " "`
        co-stars "$ACTOR_2" >> $TMP2
```

```
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3

    # Remove those that have Bacon Numbers of 2, & 1
    diff $BACON_2 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_1 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_3

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4

fi

#
# Generate list of actors with Bacon numbers of 4
#

if test $NUM -eq 4 -o $NUM -eq 0
then

    # Ensure BN3.txt exists first
    if ! test -r $BACON_3
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_3."
        exit 0
    fi

    # For each actor list their co-stars
    cat $BACON_3 | tr " " "_" > $TMP1
    for ACTOR_1 in `cat $TMP1`
    do
        ACTOR_2=`echo $ACTOR_1 | tr "_" " "`
        co-stars "$ACTOR_2" >> $TMP2
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3

    # Remove those that have Bacon Numbers of 3, 2, & 1
    diff $BACON_3 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_2 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP5
    diff $BACON_1 $TMP5 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_4

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4 $TMP5

fi

#
# Generate list of actors with Bacon numbers of 5
#

if test $NUM -eq 5 -o $NUM -eq 0
then

    # Ensure BN4.txt exists first
    if ! test -r $BACON_4
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_4."
        exit 0
    fi
```

```
    # For each actor list their co-stars
    cat $BACON_4 | tr " " "_" > $TMP1
    for ACTOR_1 in `cat $TMP1`
    do
        ACTOR_2=`echo $ACTOR_1 | tr "_" " "`
        co-stars "$ACTOR_2" >> $TMP2
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3

    # Remove those that have Bacon Numbers of 4, 3, 2, & 1
    diff $BACON_4 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_3 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP5
    diff $BACON_2 $TMP5 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP6
    diff $BACON_1 $TMP6 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_5

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4 $TMP5 $TMP6

fi

#
# Generate list of actors with Bacon numbers of 6
#

if test $NUM -eq 6 -o $NUM -eq 0
then

    # Ensure BN5.txt exists first
    if ! test -r $BACON_5
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_5."
        exit 0
    fi

    # For each actor list their co-stars
    cat $BACON_5 | tr " " "_" > $TMP1
    for ACTOR_1 in `cat $TMP1`
    do
        ACTOR_2=`echo $ACTOR_1 | tr "_" " "`
        co-stars "$ACTOR_2" >> $TMP2
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3

    # Remove those that have Bacon Numbers of 5, 4, 3, 2, & 1
    diff $BACON_5 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_4 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP5
    diff $BACON_3 $TMP5 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP6
    diff $BACON_2 $TMP6 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP7
    diff $BACON_1 $TMP7 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_6

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4 $TMP5 $TMP6 $TMP7

fi
```

```
#
# Generate list of actors with Bacon numbers of 7
#

if test $NUM -eq 7 -o $NUM -eq 0
then

    # Ensure BN6.txt exists first
    if ! test -r $BACON_6
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_6."
        exit 0
    fi

    # For each actor list their co-stars
    cat $BACON_6 | tr " " "_" > $TMP1
    for ACTOR_1 in ‘cat $TMP1‘
    do
        ACTOR_2=‘echo $ACTOR_1 | tr "_" " "‘
        co-stars "$ACTOR_2" >> $TMP2
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3

    # Remove those that have Bacon Numbers of 6, 5, 4, 3, 2, & 1
    diff $BACON_6 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_5 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP5
    diff $BACON_4 $TMP5 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP6
    diff $BACON_3 $TMP6 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP7
    diff $BACON_2 $TMP7 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP8
    diff $BACON_1 $TMP8 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_7

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4 $TMP5 $TMP6 $TMP7 $TMP8

fi

#
# Generate list of actors with Bacon numbers of 8
#

if test $NUM -eq 8 -o $NUM -eq 0
then

    # Ensure BN7.txt exists first
    if ! test -r $BACON_7
    then
        echo "gen3to8: Fatal error! Could not find file $BACON_7."
        exit 0
    fi

    # For each actor list their co-stars
    cat $BACON_7 | tr " " "_" > $TMP1
    for ACTOR_1 in ‘cat $TMP1‘
    do
        ACTOR_2=‘echo $ACTOR_1 | tr "_" " "‘
        co-stars "$ACTOR_2" >> $TMP2
    done

    # Sort removing duplicates
    sort -uf $TMP2 > $TMP3
```

```
    # Remove those that have a Bacon Numbers of 7, 6, 5, 4, 3, 2, & 1
    diff $BACON_7 $TMP3 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP4
    diff $BACON_6 $TMP4 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP5
    diff $BACON_5 $TMP5 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP6
    diff $BACON_4 $TMP6 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP7
    diff $BACON_3 $TMP7 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP8
    diff $BACON_2 $TMP8 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $TMP9
    diff $BACON_1 $TMP9 | grep ">" | grep -v "[ ]\{3\}" | grep -v "^> \-*\$" | sed -e "s/
> //" > $BACON_8

    # Remove temporary files
    rm -f $TMP1 $TMP2 $TMP3 $TMP4 $TMP5 $TMP6 $TMP7 $TMP8 $TMP9

fi

# Exit
exit 0
```

```
#! /usr/bin/perl

# Kevin Bacon Dinner Party Program by Alan Griffiths
# CGI script works with the Java applet

# Actors names are stored in $namex, where x is their letter in the table
# The results of the tests are stored in $result[num]
# where num is the number of the test
# All results are initially set to zero

# N.B. all \‘ references are actually just ‘ - this change is necessary to
# ensure correct syntax highlighting

# URL parsing, using cgi-lib
require "cgi-lib.pl";

# Set constants
$AMD_PATH = "/proj/gp-dfb2/moviedb-3.5/bin";   # Path to database
$BN_PATH = "/proj/gp-dfb2/numbers/data";       # Path to Bacon no files

&ReadParse;

# Assign eash actor to a scaler
$namea = $in{‘a’};
$nameb = $in{‘b’};
$namec = $in{‘c’};
$named = $in{‘d’};

# Error status 0 == good, 1 == bad
$error = 0;

# Check for nasties
if ($namea =~ /\‘/ || $nameb =~ /\‘/ || $namec =~ /\‘/ || $named =~ /\‘/) {
    $error = 1;
    print "Content-type: text/html\n\n";
    print "<html>\n<body>\n";
    print "20000000000\n";
    print "</body>\n";
    print "</html>\n";
    exit;
}


# Set all results to zero
for ($i = 1; $i < 11; $i++) {
    $result[$i] = 0;
}

for ($i = 1; $i < 12; $i++) {
    $bit[$i] = 0;
}


# Check with AMD that actor names are valid
$chk1 = 1;
$chk2 = 1;
$chk3 = 1;
$chk4 = 1;

# Check A
open (CHK1, "$AMD_PATH/list -cast \"$namea\" | ") || die;
$line = <CHK1>;
close (CHK1);
if (! $line) {
    $chk1 = 0;
}

# Check B
```

```
open (CHK2, "$AMD_PATH/list -cast \"$nameb\" | ") || die;
$line = <CHK2>;
close (CHK2);
if (! $line) {
    $chk2 = 0;
}

# Check C
open (CHK3, "$AMD_PATH/list -cast \"$namec\" | ") || die;
$line = <CHK3>;
close (CHK3);
if (! $line) {
    $chk3 = 0;
}

# Check D
open (CHK4, "$AMD_PATH/list -cast \"$named\" | ") || die;
$line = <CHK4>;
close (CHK4);
if (! $line) {
    $chk4 = 0;
}

if ($chk1 + $chk2 + $chk3 + $chk4 != 4) {
    $error = 1;
}


# Check that Kevin Bacon is not one of the actors
if ($namea =~ m"Bacon, Kevin" || $nameb =~ m"Bacon, Kevin" || $namec =~ m"Bacon, Kevin" |
| $named =~ m"Bacon, Kevin") {
    $error = 1;
}


# Check that duplicate actor names have not been provided
if ($namea =~ /^$nameb$/ || $namea =~ /^$namec$/ || $namea =~ /^$named$/) {
    $error = 1;
}

if ($nameb =~ /^$namec$/ || $nameb =~ /^$named$/) {
    $error = 1;
}

if ($namec =~ /^$named$/) {
    $error = 1;
}

# Catch errors
if ($error == 1) {
    print "Content-type: text/html\n\n";
    print "<html>\n<body>\n";
    print "20000000000\n";
    print "</body>\n</html>\n";
    exit;
}

# Tests
# (1)&(2) Do A and B have Bacon numbers of one?
open (FD, "$BN_PATH/BN1.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namea$/) {
        $result[1] = 1;
        $bit[2] = 1;
    }
```

```
    if ($line =~ /^$nameb$/) {
        $result[2] = 1;
        $bit[3] = 1;
    }


    if ($line =~ /^$namec$/) {
        $bit[4] = 1;
    }

    if ($line =~ /^$named$/) {
        $bit[5] = 1;
    }
}

close (FD);


# (3)&(4) Do C and D have Bacon numbers of two?
open (FD, "$BN_PATH/BN2.txt") || die;

while ($line = <FD>) {
    if ($line =~ /^$namec$/) {
        $result[3] = 1;
    }


    if ($line =~ /^$named$/) {
        $result[4] = 1;
    }

}

close (FD);

# (5) Has A starred with C, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;

$line = <FD>;

if ($line) {
    $result[5] = 1;
}


# (6) Has A starred with B, Right = No
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$nameb\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[6] = 1;
}

close (FD);

# (7) Has A starred with D, Right = No
open (FD, "$AMD_PATH/list -cast \"$namea\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[7] = 1;
}

close (FD);
```

```
# (8) Has B starred with C, Right = No
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$namec\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[8] = 1;
}

close (FD);

# (9) Has B starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$nameb\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[9] = 1;
}

close (FD);

# (10) Has C starred with D, Right = Yes
open (FD, "$AMD_PATH/list -cast \"$namec\" -cast \"$named\" | grep -v \'(T*VG*)\' | ") ||
 die;
$line = <FD>;

if ($line) {
    $result[10] = 1;
}

close (FD);



# Check to see if we have a successful dinner party
$outcome = 1;

# Check all Bacon number correct (1-4)
for ($i = 1; $i < 5; $i++) {
    if ($result[$i] == 0) {
        $outcome = 0;
    }
}
# Check co-staring relationships
# A - B
if ($result[6] == 1) {
    $outcome = 0;
    $bit[6] = 1;
}

else {
    $bit[6] = 0;
}

# A - C
if ($result[5] == 1) {
    $bit[7] = 1;
}

else {
    $outcome = 0;
    $bit[7] = 0;
}

# A - D
if ($result[7] == 1) {
    $outcome = 0;
```

```
    $bit[8] = 1;
}

else {
    $bit[8] = 0;
}

# B - C
if ($result[8] == 1) {
    $outcome = 0;
    $bit[9] = 1;
}

else {
    $bit[9] = 0;
}

# B - D
if ($result[9] == 1) {
    $bit[10] = 1;
}

else {
    $outcome = 0;
    $bit[10] = 0;
}

# C - D
if ($result[10] == 1) {
    $bit[11] = 1;
}

else {
    $outcome = 0;
    $bit[11] = 0;
}


if ($outcome == 1 && $error == 0) {
    $bit[1] = 1;
}
elsif ($error == 1) {
    $bit[1] = 2;
}
else {
    $bit[1] = 0;
}


# Output
print "Content-type: text/html\n\n";
print "<html>\n<body>\n";

for ($i = 1; $i < 12; $i++) {
    print $bit[$i];
}

print "\n</body>\n</html>\n";
exit;
```

```
#! /bin/sh
# Determine the Bacon numbers of a given list of actors
# This checks for Bacon numbers of 1 through 8
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 28/12/1998
# Modified for BN3 -> BN8, 29/03/1999
# Modified to add portability code 09/05/99

# Define variables
AMDPATH=/proj/gp-dfb2/moviedb-3.5/bin
LISTDIR=/proj/gp-dfb2/numbers/data
TMP1=/tmp/mbacon.$$.tmp # temp file
SEARCHLEV=8

# If arguments are specified, set path variables
while
    test -n "$1"
do
    case `echo $1 | sed -e "s/-//"` in
    p ) LISTDIR=$2;;
    d ) AMDPATH=$2;;
    s ) SEARCHLEV=$2;;
    * ) echo "Invalid flag \"$1\""; break ;;
    esac
    shift
    shift
done

# Step through list and report Bacon number of each actor
tr " " "_" > $TMP1

for ACTOR_1 in `cat $TMP1`
do
  ACTOR_2=`echo $ACTOR_1 | tr "_" " "`
  F=0

  # Check for KB
  if `test "$ACTOR_2" = "Bacon, Kevin"`
  then
      echo "0"
      continue                          # Found, therefore break out of loop
  fi

  # Ensure actor is valid
  if test `$AMDPATH/list -cast "$ACTOR_2" | wc -l` -eq 0
  then
      echo "Invalid actor name!"
      continue
  fi

  # Determine location of actor
  N=1
  while test "$N" -le $SEARCHLEV
  do
      if `grep -x "^$ACTOR_2\$" $LISTDIR/BN$N.txt > /dev/null`
      then
          echo $N
          N=9
          F=1
      fi
      N=`expr $N + 1`
done

# Actor not in file, therefore report as infinity
if test $F -eq 0
then
    echo "infinity"
fi
```

```
done # end for loop

# Remove temporary file
rm -f $TMP1

# Exit
exit 0
```

```
/* ------------------------------------------------------------------------ *
 * Program to convert the name of an actor to the appropriate IMDb          *
 * information page URL                                                     *
 *                                                                          *
 * By Anthony Jaroslav Truhlar, (30/01/1999)                                *
 * ------------------------------------------------------------------------ *
 */

#include<stdio.h>
#include<ctype.h>

int main( int argc, char **argv ) {

    /* Declare variables */
    char out[100];          /* array for URL to be output */
    char *inptr = argv[1]; /* pointer to argument */
    char *outptr = out;    /* pointer to output array */

    /* Check number of arguments */
    if ( argc != 2 ) { /* incorrect, so print error & quit */

        (void) printf("Incorrect number of arguments!\n");
        (void) printf("usage: name2url <\"Surname, Forenames\">\n");
        return( 0 );

    } // end if

    /* Take name and convert to URL */
    /* ---------------------------- */

    /* Copy path to CGI script at IMDb */
    (void) strcpy( out, "http://www.imdb.com/M/person-exact?" );
    outptr += 35; /* adjust pointer to end of copied string */

    /* Step through and convert name */
    for( ; *inptr != 0; inptr++, outptr++ ) {

        if ( isalnum( *inptr ) ) { /* if alphanumeric, copy character over */

            *outptr = *inptr;

        } else { /* if non-alphanumeric, copy ascii hex value preceeded by % */

            (void) sprintf( outptr, "%%%2X", *inptr & 255 );
            outptr += 2; /* increment pointer appropriately - 3 chars added */

        } // end if

    } // end for

    *outptr = 0; /* null terminate the array */

    /* Output URL */
    (void) printf("%s\n", out);

    /* End of program */
    return( 0 );

} // end of main
```

```
// Kevin Bacon's Dinner Parties
// Java interface
// Tom Muskett (tam97p)

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;

public class newinter extends Applet {

    // Initialise variables
    private Button clear_button, submit_button;
    private TextField actor_A, actor_B, actor_C, actor_D;
    private Choice actor_links;
    private Panel text_panel, button_panel;
    private Label label_a, label_b, label_c, label_d;
    private Image table, success, fail;
    private int AppletState = 0;
    private int PartyStatus[] = new int[ 11 ];
    private int A_nodex = 20 + 110, A_nodey = 75 + 60;
    private int B_nodex = 208 + 110, B_nodey = 75 + 60;
    private int C_nodex = 20 + 110, C_nodey = 189 + 60;
    private int D_nodex = 208 + 110, D_nodey = 189 + 60;
    private int K_nodex = 114 + 110, K_nodey = 23 + 60;

    // Initialisation method
    public void init() {

        // Set colours
        this.setBackground(Color.white);
        Color brown = new Color(128,64,0);

        // Load pictures
        table = getImage(getDocumentBase(),"javatable.jpg");
        success = getImage(getDocumentBase(),"successtable.jpg");
        fail = getImage(getDocumentBase(),"failtable.jpg");

        // Declare panels
        text_panel = new Panel();
        button_panel = new Panel();

        // Position panels using layout manager
        setLayout(new BorderLayout(1,50));
        add("North", text_panel);
        add("South", button_panel);

        // Do the button panel
        button_panel.setLayout(new GridLayout(1,3));

        submit_button = new Button("Submit");
        submit_button.setBackground(brown);
        submit_button.setForeground(Color.yellow);
        submit_button.disable();
        button_panel.add(submit_button);

        actor_links = new Choice();
        actor_links.addItem("<Actor Info>");
        actor_links.addItem("Actor A");
        actor_links.addItem("Actor B");
        actor_links.addItem("Actor C");
        actor_links.addItem("Actor D");
        actor_links.setBackground(brown);
        actor_links.setForeground(Color.yellow);
        actor_links.disable();
        actor_links.hide();
        button_panel.add(actor_links);
```

```
        clear_button = new Button("Clear");
        clear_button.setBackground(brown);
        clear_button.setForeground(Color.yellow);
        button_panel.add(clear_button);

        // Do the text panel
        text_panel.setLayout(new GridLayout(2,4));
        label_a = new Label ("Actor A", Label.RIGHT);
        label_a.setForeground(brown);
        text_panel.add(label_a);
        actor_A = new TextField(20);
        text_panel.add(actor_A);

        actor_B = new TextField(20);
        text_panel.add(actor_B);
        label_b = new Label ("Actor B", Label.LEFT);
        label_b.setForeground(brown);
        text_panel.add(label_b);

        label_c = new Label ("Actor C", Label.RIGHT);
        label_c.setForeground(brown);
        text_panel.add(label_c);
        actor_C = new TextField(20);
        text_panel.add(actor_C);

        actor_D = new TextField(20);
        text_panel.add(actor_D);
        label_d = new Label("Actor D", Label.LEFT);
        label_d.setForeground(brown);
        text_panel.add(label_d);

    } // end of initialiser


    // The paint method, responsible for drawing the table
    public void paint(Graphics g) {

        // If the applet has just been started...
        if (AppletState == 0 ) {
            g.drawImage(table,110,60,this);
        } // end if

        // If the party is a failure...
        if ( AppletState == 1 ) {

            g.drawImage(fail,110,60,this);

            if ( PartyStatus[ 1 ] == 1 ) {
                g.setColor(Color.green);
                g.drawLine(K_nodex, K_nodey, A_nodex, A_nodey);
                g.drawLine(K_nodex, K_nodey + 1, A_nodex, A_nodey + 1);
            }

            if ( PartyStatus[ 2 ] == 1 ) {
                g.setColor(Color.green);
                g.drawLine(K_nodex, K_nodey, B_nodex, B_nodey);
                g.drawLine(K_nodex, K_nodey + 1, B_nodex, B_nodey + 1);
            }

            if ( PartyStatus[ 3 ] == 1 ) {
                g.setColor(Color.yellow);
                g.drawLine(K_nodex, K_nodey, C_nodex, C_nodey);
                g.drawLine(K_nodex, K_nodey + 1, C_nodex, C_nodey + 1);
            }

            if ( PartyStatus[ 4 ] == 1 ) {
                g.setColor(Color.yellow);
```

```
            g.drawLine(K_nodex, K_nodey, D_nodex, D_nodey);
            g.drawLine(K_nodex, K_nodey + 1, D_nodex, D_nodey + 1);
        }

        if ( PartyStatus[ 5 ] == 1 ) {
            g.setColor(Color.yellow);
            g.drawLine(A_nodex, A_nodey, B_nodex, B_nodey);
            g.drawLine(A_nodex, A_nodey + 1, B_nodex, B_nodey + 1);
        }

        if ( PartyStatus[ 6 ] == 1 ) {
            g.setColor(Color.green);
            g.drawLine(A_nodex, A_nodey, C_nodex, C_nodey);
            g.drawLine(A_nodex + 1, A_nodey, C_nodex + 1, C_nodey);
        }

        if ( PartyStatus[ 7 ] == 1 ) {
            g.setColor(Color.yellow);
            g.drawLine(A_nodex, A_nodey, D_nodex, D_nodey);
            g.drawLine(A_nodex, A_nodey + 1, D_nodex, D_nodey + 1);
        }

        if ( PartyStatus[ 8 ] == 1 ) {
            g.setColor(Color.yellow);
            g.drawLine(B_nodex, B_nodey, C_nodex, C_nodey);
            g.drawLine(B_nodex, B_nodey + 1, C_nodex, C_nodey + 1);
        }

        if ( PartyStatus[ 9 ] == 1 ) {
            g.setColor(Color.green);
            g.drawLine(B_nodex, B_nodey, D_nodex, D_nodey);
            g.drawLine(B_nodex + 1, B_nodey, D_nodex + 1, D_nodey);
        }

        if ( PartyStatus[ 10 ] == 1 ) {
            g.setColor(Color.green);
            g.drawLine(C_nodex, C_nodey, D_nodex, D_nodey);
            g.drawLine(C_nodex, C_nodey + 1, D_nodex, D_nodey + 1);
        }

    } // end for if party is a failure

    // If the party is a success...
    if ( AppletState == 2 ) {
        g.drawImage(success,110,60,this);
    } // end if

} // end of paint method


// This method tests when the user moves the mouse
// It is used to enable the "submit" button
public boolean mouseMove(Event e, int x, int y) {

    if ( AppletState == 0 ) {

        if (        ( this.actor_A.getText().length() == 0 ) ||
                    ( this.actor_B.getText().length() == 0 ) ||
                    ( this.actor_C.getText().length() == 0 ) ||
                    ( this.actor_D.getText().length() == 0 )) {
            this.submit_button.disable();
        } else {
            this.submit_button.enable();
        }

    } // end if applet state is 0

    return true;
```

```
    } // end method


    // This method is for when the user clicks the mouse
    public boolean action(Event event, Object arg) {

        // If clear button is clicked on...
        if (event.target == clear_button) {

            switch( AppletState ) {

            case 0:
                this.actor_A.setText("");
                this.actor_B.setText("");
                this.actor_C.setText("");
                this.actor_D.setText("");
                this.submit_button.disable();
                break;

            default:
                URL urlInstruct = null;
                try { urlInstruct = new URL("http://crilly.cs.nott.ac.uk/bacon/instructio
ns.html"); } catch (MalformedURLException e) { this.setBackground( Color.red );}

                this.getAppletContext().showDocument(urlInstruct);
                break;

            } // end of switch

            return true;

        } // end clear button

        // If submit button is clicked on...
        if (( event.target == submit_button) && (AppletState == 0)) {

            // Set up required variables
            DataInputStream dis = null;
            int flag = 0;
            String s;

            // Get the user input and format it correctly
            URL url = null;
            String actora, actorb, actorc, actord, urlString, urlStringA;

            actora = new String(this.actor_A.getText().replace(' ', '+'));
            actorb = new String(this.actor_B.getText().replace(' ', '+'));
            actorc = new String(this.actor_C.getText().replace(' ', '+'));
            actord = new String(this.actor_D.getText().replace(' ', '+'));

            // Construct the URL to query the script
            urlString = new String("http://crilly.cs.nott.ac.uk/cgi-bin/java-bac.pl?a=" +
 actora + "&b=" + actorb + "&c=" + actorc + "&d=" + actord);
            urlStringA = new String("http://crilly.cs.nott.ac.uk/cgi-bin/bacon.pl?a=" + a
ctora + "&b=" + actorb + "&c=" + actorc + "&d=" + actord);

            try {
                url = new URL(urlString);
            } catch (MalformedURLException e ) {
                this.setBackground(Color.red);
            } // end url assignment

            // Open the connection to the script
            try {
                dis = new DataInputStream( url.openConnection().getInputStream() );
```

```
        // Enter a while loop to read the output
        while ( (s=dis.readLine())!=null ) {

            // If the data is imminent, read it!
            if ( flag == 1 ) {

                // This next for loop gets the array
                // which represents which lines are to be drawn
                // and then set flags in array PartyStatus
                for ( int i = 0; i < 11; i++ ) {
                    PartyStatus[ i ] = s.charAt(i) - 48;
                } // end loop

                flag = 0;

            } // end if for flag == 1

            // This next part detects if the data is
            // about to be sent from the script
            if ( s.equalsIgnoreCase("<body>")) {
                flag = 1;
            } // end if

        } // end while loop

        // Make sure any exceptions are caught
    } catch ( IOException e ) { this.setBackground(Color.red); }

    // Display actor links box
    if ( PartyStatus[ 0 ] != 2 ) {

        actor_links.enable();
        actor_links.show();

        // Modify the text boxes
        actor_A.setEditable(false);
        actor_B.setEditable(false);
        actor_C.setEditable(false);
        actor_D.setEditable(false);

    } // end if

    // Set the applet state and call repaint
    switch ( PartyStatus[ 0 ]) {

    case 0:
        AppletState = 1;
        submit_button.setLabel("Try again");
        clear_button.setLabel("Read instructions");
        repaint();
        break;

    case 1:
        AppletState = 2;
        submit_button.setLabel("Enter hall of fame");
        clear_button.setLabel("Read instructions");
        repaint();
        break;

    case 2:
        try {
            url = new URL(urlStringA);
        } catch (MalformedURLException e ) {
            this.setBackground( Color.red );
        } // end url assignement

        Applet.this.getAppletContext().showDocument(url);
        break;
```

```
        } // end switch

        return true;
    } // end submit button

    // If try again is selected...
    if (( event.target == submit_button ) && ( AppletState == 1 )) {

            AppletState = 0;
            submit_button.setLabel("Submit");
            clear_button.setLabel("Clear");
            repaint();

            actor_links.hide();
            actor_links.disable();

            actor_A.setEditable(true);
            actor_B.setEditable(true);
            actor_C.setEditable(true);
            actor_D.setEditable(true);

            return true;

    } // end if

    // If hall of fame link is selected...
    if (( event.target == submit_button ) && ( AppletState == 2 )) {

        AppletState = 0;
        URL urlHall = null;

        try {
            urlHall = new URL("http://crilly.cs.nott.ac.uk/bacon/enter.html");
        } catch (MalformedURLException e) {
            this.setBackground( Color.red );
        }

        this.getAppletContext().showDocument(urlHall);
        return true;

    } // end if

    // If an actor link was selected...
    if ( event.target == actor_links ) {

        // Initialise variables
        String name = null;
        URL linkurl = null;
        StringBuffer mdfy = new StringBuffer( 50 );
        char c;

        // Set a string to the actors name to be queried
        if (arg.equals("Actor A")) name = new String(actor_A.getText());
        else if (arg.equals("Actor B")) name = new String(actor_B.getText());
        else if (arg.equals("Actor C")) name = new String(actor_C.getText());
        else if (arg.equals("Actor D")) name = new String(actor_D.getText());

        if (name != null ) {

            // Find where the comma is in the proceedings
            for ( int k = 0; k < name.length(); k++ ) {

                c = name.charAt( k );

                if ( Character.isLetterOrDigit(c) && ((int) c < 127) ) {
                    mdfy.append(c);
                } else {
```

```
                        mdfy.append('%');
                        mdfy.append(Integer.toHexString(c & 255).toUpperCase());
                    }

                } // end for

                // Form the IMDB URL
                try {
                    linkurl = new URL("http://www.imdb.com/M/person-exact?" + mdfy.toStri
ng() );
                } catch (MalformedURLException e) {
                    this.setBackground( Color.red );
                }

                // Display the document
                this.getAppletContext().showDocument(linkurl);
            } // end if

            return true;
        } // end choice box code

        else return super.action(event, arg);

    } // end action method

} // end applet
```

```
my $url_path = "/bacon";
my $cgi_path = "/cgi-bin";

sub printhead {
    my ($out) = shift;
    my ($title) = shift;

    print "Content-type: text/html\n\n";
    print $out "<html>\n";
    print $out "<head>\n";
    print $out "<title>Kevin Bacon's Dinner Parties - $title</title>\n";
    print $out "</head>\n\n";

    print $out "<BODY BGCOLOR=\"#FFFFFF\" TEXT=\"#000000\" ALINK=\"#990033\" VLINK=\"#990
033\" LINK=\"#003366\">\n";
    print $out "<A NAME=\"topofpage\">\n\n";

    print $out "<center>\n\n";

    print $out "<!-- Begin Content -->\n";
    print $out "<table width=\"600\">\n";
    print $out "<td width=\"600\">\n";
    print $out "<!-- Begin Header -->\n\n";

    print $out "<!-- Begin Body -->\n";
    print $out "<table width=\"600\">\n\n";

    print $out "<!-- Start Left Column -->\n";
    print $out "<td width=\"106\" valign=top>\n";
    print $out "<p><img src=\"$url_path/kevinbacon.jpg\" WIDTH=\"75\" HEIGHT=\"95\"></p>\
n";
    print $out "<font size=\"2\" face=\"Arial\">\n";
    print $out "<font size=\"4\" face=\"Arial\"><b>Kevin's Dinner Parties</b></font><p>\n
";
    print $out "<b><a href=\"$url_path/game.html\">Play the game!</a><p>\n";
    print $out "<a href=\"$url_path/instructions.html\">Instructions</a><p>\n";
    print $out "<a href=\"$url_path/about.html\">About the game</a><p>\n";
    print $out "<a href=\"$url_path/maths.html\">The maths behind the game</a><p>\n";
    print $out "<a href=\"$url_path/hallofame.html\">Kevin's \"Hall 'O' Fame\"</a><p>\n";
    print $out "<p>\n";
    print $out "<font size=\"4\" face=\"Arial\"><b>Bacon Numbers</b></font><p>\n";
    print $out "<b><a href=\"$url_path/bn.html\">Find a Bacon number</a><p>\n";
    print $out "<b><a href=\"$cgi_path/stat.pl\">Bacon number statistics</a><p>\n";
    print $out "<p>\n";
    print $out "<font size=\"4\" face=\"Arial\"><b>Other links</b></font><p>\n";
    print $out "<b><a href=\"http://us.imdb.com/Name?Bacon,+Kevin\">Kevin Bacon's bio</a>
<p>\n";
    print $out "<a href=\"http://www.cs.virginia.edu/oracle\">The Oracle Of Kevin Bacon</
a><p>\n";
    print $out "<a href=\"http://www.imdb.com\">The Internet Movie Database</a><p>\n";
    print $out "</b></font>\n";
    print $out "<p><img src=\"$url_path/nottlogo.gif\" WIDTH=\"75\" HEIGHT=\"91\"></p>\n"
;
    print $out "</td>\n";
    print $out "<!-- End Left Column -->\n\n";

    print $out "<!-- Start Spacer Column -->\n";
    print $out "<td width=\"2\" bgcolor=\"#003366\">\n";
    print $out " \n";
    print $out "</td>\n";
    print $out "<!-- End Spacer Column -->\n\n";

    print $out "<!-- Start Body Column -->\n";
    print $out "<td width=\"492\" valign=top>\n\n";

    print $out "<P><P>\n";

    print $out "</b>\n";
```

```
    print $out "<font face=\"Verdana, Arial, Helvetica\">\n";


}


sub printfoot {
    my $out = shift;

    print $out "<hr>\n";
    print $out "<center>\n";
    print $out "<font size=\"2\">[<a href=\"$url_path/game.html\">Play the game!</a>][<a
href=\"$url_path/instructions.html\">Instructions</a>][<a href=\"$url_path/about.html\">A
bout the game</a>]<br>\n";
    print $out "[<a href=\"$url_path/maths.html\">Maths behind the game</a>][<a href=\"$u
rl_path/hallofame.html\">Kevin's Hall O' Fame</a>]<br>\n";
    print $out "[<a href=\"$url_path/bn.html\">Find a Bacon number</a>][<a href=\"$cgi_pa
th/stat.pl\">Bacon number statistics</a>]<br>\n";
    print $out "[<a href=\"http://www.imdb.com\">The Internet Movie Database</a>]<br><br>
\n";
    print $out "<!--  (c) IMDb.com 1998 - this search box provides access to -->\n";
    print $out "<!--  the Internet Movie Database search facility please use as is -->\n"
;
    print $out "<!--  This code may be used by other web sites as is without -->\n";
    print $out "<!--  additional permission, please note no guarentees are -->\n";
    print $out "<!--  made for this code.  -->\n";
    print $out "<!--  -->\n";

    print $out "<table BORDER CELLPADDING=0 >\n";
    print $out "<tr>\n";
    print $out "<td><form METHOD=\"POST\" ACTION=\"http://us.imdb.com/M/multi-search\" >\
n";
    print $out "<table CELLSPACING=0 CELLPADDING=0 VALIGN=\"TOP\" >\n";
    print $out "<tr>\n";
    print $out "<td VALIGN=CENTER>\n";
    print $out "<center><b>Search IMDb for title/name:</b></center>\n";
    print $out "</td>\n";
    print $out "</tr>\n";

    print $out "<tr>\n";
    print $out "<td VALIGN=CENTER>\n";
    print $out "<table BORDER=0 CELLSPACING=2 CELLPADDING=0 >\n";
    print $out "<tr>\n";
    print $out "<td VALIGN=CENTER>\n";
    print $out "<center><input NAME=\"for\" SIZE=\"15\" ></center>\n";
    print $out "</td>\n";

    print $out "<td VALIGN=CENTER>\n";
    print $out "<center><input NAME=\"title\" VALUE=\"GO\" TYPE=\"image\" SRC=\"http://ic
ons.imdb.com/Icons/GO.gif\" BORDER=\"0\" WIDTH=\"21\" VALIGN=\"BASELINE\" HEIGHT=\"17\" A
LT=\"GO\"></center>\n";
    print $out "</td>\n";

    print $out "<td VALIGN=TOP ROWSPAN=\"2\"><input NAME=\"title\" VALUE=\"GO\" TYPE=\"im
age\" SRC=\"http://us.imdb.com/Icons/logo.gif\" BORDER=0 height=27 width=57 ALT=\"Search
the IMDb\"></td>\n";
    print $out "</tr>\n";

    print $out "<tr>\n";
    print $out "<td ALIGN=LEFT VALIGN=TOP COLSPAN=\"2\" >\n";
    print $out "<center><input NAME=\"type\" TYPE=\"radio\" VALUE=\"title\" CHECKED>Movie
 <input name=\"type\" TYPE=\"radio\" VALUE=\"name\">\n";
    print $out "Person</center>\n";
    print $out "</td>\n";
    print $out "</tr>\n";
    print $out "</table>\n";
    print $out "</td>\n";
    print $out "</tr>\n";
```

```
    print $out "<tr>\n";
    print $out "<td ALIGN=LEFT VALIGN=TOP >\n";
    print $out "<center><font size=-1><a href=\"http://us.imdb.com/search\">more MOVIE se
arch\n";
    print $out "options</a></font>\n";
    print $out "<br>Powered by <a href=\"http://us.imdb.com\">www.IMDb.com</a></center>\n
";
    print $out "</td>\n";
    print $out "</tr>\n";
    print $out "</table>\n";
    print $out "</td>\n";
    print $out "</tr>\n";
    print $out "</table>\n";

    print $out "<!-- End of IMDb search box code -->\n";

    print $out "</font></center>\n\n";

    print $out "</td>\n";
    print $out "<!-- End Body Column -->\n\n";

    print $out "</table>\n";
    print $out "<!-- End Body -->\n\n";

    print $out "</td>\n";
    print $out "</table>\n";
    print $out "<!-- End Content -->\n\n";

    print $out "</center>\n\n";

    print $out "</html>\n";
}

return 1;
```

```
#! /usr/bin/perl

# Get bacon 1's
open (BN1, "BN1.txt") || die;
@bn1 = <BN1>;
close (BN1);

# Get bacon 2's
open (BN2, "BN2.txt") || die;
@bn2 = <BN2>;
close (BN2);

# A-----B
# |     |
# |     |
# C-----D

# --------------Outer loop on A--------------
foreach $actorA (@bn1) {
    $actoA = $actorA;
#    $actoA =~ s/\(/\\(/;
#    $actoA =~ s/\)/\\)/;
#    print "$actorA\n";

# Find actors A has starred with
    open (COSTA, "echo \"$actorA\" | costars | ") || die;
    @costA = <COSTA>;
    close (COSTA);

# Find all of A's costars with BN 2
    foreach $actorC (@costA) {
#       print "?$actorC\n";
        $icount = 0;
        foreach $b2 (@bn2) {
            if ($b2 =~ /^$actorC$/) {
                $costA2[$icount] = $actorC;
                $icount++;
#               print "***$actorC\n";
                last;
            }
        }
    }

# ---------------Inner loop 1 on C-------------
    foreach $actorC (@costA2) {
#       $actorC =~ s/\(/\\(/;
#       $actorC =~ s/\)/\\)/;
#       print "2$actorC\n";
# Find actors C has starred with
        open (COSTC, "echo \"$actorC\" | costars | ") || die;
        @costC = <COSTC>;
        close (COSTC);

# Find all of C's costars with BN2
        foreach $actorD (@costC) {
#           print "??$actorD\n";
            $icount = 0;
            foreach $b2 (@bn2) {
                if ($actorD =~ /^$b2$/) {
                    $costC2[$icount] = $actorD;
                    $icount++;
#                   print "2***$actorD\n";
                    last;
                }
            }
        }

# --------------Inner loop 2 on D-----------------
```

```
        foreach $actorD (@costC2) {
#           $actorD =~ s/\(/\\(/;
#           $actorD =~ s/\)/\\)/;
#           print "3$actorD\n";
# Find actors D has starred with
            open (COSTD, "echo \"$actorD\" | costars | ") || die;
            @costD = <COSTD>;
            close (COSTD);

# Find all of D's costars with BN1
            foreach $actorB (@costD) {
#               print "???$actorB\n";
                $icount = 0;
                foreach $b1 (@bn1) {
                    if ($actorB =~ /^$b1$/) {
                        $costD2[$icount] = $actorB;
                        $icount++;
#                       print "1**$actorB\n";
                        last;
                    }
                }
            }

# -------------Inner loop 3 on B-----------------
            foreach $actorB (@costD2) {
#               $actorB =~ s/\(/\\(/;
#               $actorB =~ s/\)/\\)/;
#               print "4$actorB\n";
# Find all B's who have not starred with A
                chomp ($actorA);
                chomp ($actorB);
                chomp ($actorC);
                chomp ($actorD);
                open (TEST, "list -cast \"$actorA\" -cast \"$actorB\" |") || die;
                $res = <TEST>;
                close (TEST);

                if (! $res) {
                    open (TEST2, "list -cast \"$actorA\" -cast \"$actorD\" | ") || die;
                    $res2 = <TEST2>;
                    close (TEST2);

                    if (! $res2) {
                       open (TEST3, "list -cast \"$actorB\" -cast \"$actorC\" | ") || die
;
                       $res3 = <TEST3>;
                       close (TEST3);

                       if (! $res3) {
                           chomp ($actorC);
                           $party = $actorA . "&" . $actorB . "&" . $actorC . "&" . $acto
rD . "\n";
                           open (SUCC, ">>search.log") || die;
#                          print $party;
                           print SUCC $party;
                           close (SUCC);
                       }
                    }
                }
            }
        }
    }
}
```

```
#! /bin/sh
# Shell script to remove AMD's descriptive tags from output
# Takes stdin, stream edits it to stdout
# By Anthony Jaroslav Truhlar (ajt97c@cs.nott.ac.uk), 26/12/1998

tr "()" "<>" | \
sed -e "s/\[.*\]//" \
    -e "s/<AA>//" \
    -e "s/<AAN>//" \
    -e "s/<AFI>//" \
    -e "s/<AFIN>//" \
    -e "s/<as.*>//" \
    -e "s/<as.*//" \
    -e "s/<archive.*>//" \
    -e "/Bacon, Kevin/d" \
    -e "s/<BFA>//" \
    -e "s/<C:GG>//" \
    -e "s/<C:GGN>//" \
    -e "s/<cameo>//" \
    -e "s/<Cesar>//" \
    -e "s/<David.*>//" \
    -e "s/<director's cut>//" \
    -e "s/<credited .n Director's Cut>//" \
    -e "s/<cut from US version>//" \
    -e "s/<cut from most prints>//" \
    -e "s/<deleted from US print>//" \
    -e "s/<episode .>//" \
    -e "s/<episode \".*\">//" \
    -e "s/<English version>//" \
    -e "s/<Felix.>//" \
    -e "s/<French version>//" \
    -e "s/<.*footage>//" \
    -e "s/<German version>//" \
    -e "s/<GG>//" \
    -e "s/<GGN>//" \
    -e "s/<GLV>//" \
    -e "s/<Golden Horse>//" \
    -e "s/<Guldbagge>//" \
    -e "s/<Goya>//" \
    -e "s/<interviewer>//" \
    -e "s/<long version>//" \
    -e "s/<N:GG>//" \
    -e "s/<part.*>//" \
    -e "s/<restored version>//" \
    -e "s/<s>//" \
    -e "s/<S:AA>//" \
    -e "s/<S:AAN>//" \
    -e "s/<S:AFI>//" \
    -e "s/<S:AFIN>//" \
    -e "s/<S:GG>//" \
    -e "s/<S:GGN>//" \
    -e "s/<S:Golden Horse>//" \
    -e "s/<S:HKFA>//" \
    -e "s/<scenes.*>//" \
    -e "s/<special.*>//" \
    -e "s/<segment.*>//" \
    -e "s/<Spanish version>//" \
    -e "s/<stock footage from.*>//" \
    -e "s/<TV version>//" \
    -e "s/<unconfirmed>//" \
    -e "s/<uncred.ted>//" \
    -e "s/<uncredited.* cameo>//" \
    -e "s/<unfinished>//" \
    -e "s/<unreleased>//" \
    -e "s/<US version>//" \
    -e "s/<US release>//" \
    -e "s/<US TV version>//" \
    -e "s/<VG>//" \
```

```
    -e "s/<voice.*>//" \
    -e "s/<voice.*//" \
    -e "s/<[a-z].* voice>//" \
    -e "s/<[A-Z].* voice>//" \
    -e "s/ *$//" | \
tr "<>" "()"

# Exit
exit 0
```

```
/* ------------------------------------------------------------------------ *
 * Program to convert the name of a movie to the appropriate IMDb           *
 * information page URL                                                     *
 *                                                                          *
 * By Anthony Jaroslav Truhlar, (31/01/1999)                                *
 * ------------------------------------------------------------------------ *
 */

#include<stdio.h>
#include<ctype.h>

int main( int argc, char **argv ) {

    /* Declare variables */
    char out[160];         /* array for URL to be output */
    char *inptr = argv[1]; /* pointer to argument */
    char *outptr = out;    /* pointer to output array */

    /* Check number of arguments */
    if ( argc != 2 ) { /* incorrect, so print error & quit */

        (void) printf("Incorrect number of arguments!\n");
        (void) printf("usage: title2url <\"Film Title (Year)\">\n");
        return( 0 );

    } // end if

    /* Take name and convert to URL */
    /* ---------------------------- */

    /* Copy path to CGI script at IMDb */
    (void) strcpy( out, "http://www.imdb.com/M/title-exact?" );
    outptr += 34; /* adjust pointer to end of copied string */

    /* Step through and convert name */
    for( ; *inptr != 0; inptr++, outptr++ ) {

        if ( isalnum( *inptr ) ) { /* if alphanumeric, copy character over */

            *outptr = *inptr;

        } else { /* if non-alphanumeric, copy ascii hex value preceeded by % */

            (void) sprintf( outptr, "%%%2X", *inptr & 255 );
            outptr += 2; /* increment pointer appropriately - 3 chars added */

        } // end if

    } // end for

    *outptr = 0; /* null terminate the array */

    /* Output URL */
    (void) printf("%s\n", out);

    /* End of program */
    return( 0 );

} // end of main
```

```
#!/bin/sh

##############################################################################
#              All-in-One update script for AMD and BN generators            #
#  Fetches files from ftp.fu-berlin.de (fast European mirror of AMD) cleans   #
#  the old files out, regenerating AMD from retrieved files. Following this   #
#  the lists of Bacon numbers are generated, and the old ones renamed.        #
#  The script takes AMD off-line temporarily to do this (ensures accurate     #
#  responses are returned) - this should now be for less than a minute rather #
#  5-10 minutes in the past (also actor lists should fully correlate with AMD.#
# ------------------------------------------------------------------------- #
#          By Anthony J. Truhlar (ajt97c@cs.nott.ac.uk), 01/02/1999          #
##############################################################################
#           Modified to provide better error control - 07/02/1999            #
#            Further modifications for error control - 23/02/1999            #
#           Modified to work with C-based executable - 10/04/1999            #
#          Modified to wotk with two copies of the DB - 08/05/1999           #
#           Modified to allow paths to be specified - 09/05/1999             #
##############################################################################

AMDPATH=/proj/gp-dfb2/moviedb-3.5  # directory containing the AMD
LISTDIR=/proj/gp-dfb2/numbers      # directory containing actor list tools
LOG=$LISTDIR/cmdout.log            # command output log
TMPDIR=/tmp/gen$$     # temporary directory for generation of actor lists
TMPARC=/tmp/upd$$.tgz # temporary file for archive of previous weeks lists

# Set path variables according to arguments
while
    test -n "$1"
do
    case `echo $1 | sed -e "s/-//"` in
    c ) export PATH=$2;;
    d ) AMDPATH=$2;;
    o ) LOG=$2;;
    p ) LISTDIR=$2;;
    * ) echo "Invalid flag \"$1\""; break;;
    esac
    shift
    shift
done

# Get the latest lists (attempt to)
cd $AMDPATH/moviedb-3.5/lists > $LOG
$AMDPATH/moviedb-3.5/etc/lfetch -auto -user $USER@cs.nott.ac.uk -acr -acs -movie -castcom
 -link >> $LOG

# Check that update is necessary, if not exit
if test `grep "newer" $LOG | wc -l` -eq 0
then
   echo "weekly_upd: Update not required, process terminated." >> $LOG
   make -C $AMDPATH/moviedb-3.5 cleanlists >> $LOG
   exit 0
fi

cd $LISTDIR >> $LOG

# Generate new database files in copy of AMD
# Clean out old database files
make -C $AMDPATH/moviedb-3.5 cleandbs >> $LOG

# Regenerate database from new lists
make -C $AMDPATH/moviedb-3.5 databases >> $LOG
chgrp gp-dfb2 $AMDPATH/moviedb-3.5/dbs/* >> $LOG

# Test that update has been sucessful
if ! `$AMDPATH/moviedb-3.5/bin/list -cast "Bacon, Kevin" > /dev/null`
then
   echo "weekly_upd: Fatal Error: Update failed, existing database untouched." >> $LOG
```

```
   make -C $AMDPATH/moviedb-3.5 cleanlists >> $LOG
   make -C $AMDPATH/moviedb-3.5 cleandbs >> $LOG
   exit 0
fi

# Remove lists now that they are no longer needed
make -C $AMDPATH/moviedb-3.5 cleanlists >> $LOG

# Regenerate Bacon numbers using newly generate database
# N.B. BN1.txt to BN8.txt takes a while to generate so generate in /tmp
# then move to $LISTDIR/data when completed
mkdir $TMPDIR >> $LOG
$LISTDIR/fastgen -o -p $TMPDIR -d $AMDPATH/moviedb-3.5/bin >> $LOG  # generate new number
s

# archive last week's data (tar then gzip, naming file with current date)
tar czvf $TMPARC $LISTDIR/data/*.txt >> $LOG
mv $TMPARC $LISTDIR/data/archive/‘date | tr " " "_"‘.tgz >> $LOG
chgrp gp-dfb2 $LISTDIR/data/archive/*.tgz >> $LOG
rm $LISTDIR/data/*.txt >> $LOG

# move generated files to destination directory
N=1
while test "$N" -le 8
do
    mv $TMPDIR/BN$N.txt $LISTDIR/data >> $LOG
    N=‘expr $N + 1‘
done

# remove temporary generate directory
rm -rf $TMPDIR >> $LOG

# adjust permissions and group of the actor lists
chmod 644 $LISTDIR/data/*.txt >> $LOG
chgrp gp-dfb2 $LISTDIR/data/*.txt >> $LOG

# Lock out access while AMD is moved to 'live' position
chmod 400 $AMDPATH/bin/*

# Move newly generated files to 'live' position
mv $AMDPATH/moviedb-3.5/dbs/*.titles $AMDPATH/dbs
mv $AMDPATH/moviedb-3.5/dbs/*.names $AMDPATH/dbs
mv $AMDPATH/moviedb-3.5/dbs/*.index $AMDPATH/dbs
mv $AMDPATH/moviedb-3.5/dbs/*.key $AMDPATH/dbs
mv $AMDPATH/moviedb-3.5/dbs/*.data $AMDPATH/dbs
mv $AMDPATH/moviedb-3.5/dbs/*.counts $AMDPATH/dbs

# Re-enable access to AMD
chmod 755 $AMDPATH/bin/*
chmod 644 $AMDPATH/bin/Makefile

# Finished!
exit 0
```

Appendix D

The following pages contain the manual pages for the project.

NAME

bacon − Return the Bacon number of an actor.

SYNOPSIS

bacon "*actor*" [-p *path* | -d *path*] [-s *n*]

DESCRIPTION

Background Information

Bacon employs the concept of Bacon numbers, which are in turn based on the concept of Erdos numbers (initially devised by mathematician Paul Erdos). Erdos numbers are based on the co-authoring of mathematical papers, i.e. somebody with an Erdos number of 1 has directly co-authored a paper with Paul Erdos; whilst somebody with an Erdos number of 2 has co-authored a paper with somebody who has co-authored a paper with Paul Erdos, but has not directly co-authored a paper with Paul Erdos himself. For the purposes of the software engineering group project "The connected world of Frank Ramsey", this concept of Erdos numbers has been mapped onto the co-starring relationships between the various actors within the film industry. In particular the actor Kevin Bacon, who at the outset of the project was the subject of a considerable amount of conjecture regarding the roles that he had played, and how "connected" he was to other Hollywood actors (he was rumoured to be the best connected bit-part actor in Hollywood - investigation of this has shown this not to be the case). Hence the concept of Bacon numbers was born.

What it does

The purpose of *bacon* is to query the lists of actors generated by the tool *fastgen* to allow a user to determine the Bacon number of a given actor. The term 'Bacon number' is employed loosely here, since although the default operation of *fastgen* is to generate lists of actors connected to Kevin Bacon, it can be used to generate lists of actors connected to another actor. In such cases one might have generated the actor lists for Sean Connery, and in using this tool would then effectively be determining the Connery number of an actor. Hence the term Bacon number is purely conceptual, since the default mode of operation of all the tools (including this one) is to work for Kevin Bacon since they were designed for the game Kevin Bacon's Dinner Parties.

Options

-d *path* Allows the user to specify the path to the *AMD* binaries. This is useful when running on systems where tne *AMD* isn't in the path, or you want to query a different version of the database running on the same system. The default when this option is not specified is /proj/gp-dfb2/moviedb-3.5/bin, which relates to the installation on Crilly - the machine the code was developed on.

-p *path* Allows the user to specify the path to the location of the actor lists as generated by *fastgen*. As with the path to the *AMD* binaries, the default value of this (i.e. when the option is not specified) is /proj/gp-dfb2/numbers/data, which (as with the path to the *AMD* binaries) relates to the installation on Crilly - the machine that the game was developed on.

-s *n* This value represents the depth to which you wish to search for a connection with a given actor. The default value is 8, which relates to the actor Kevin Bacon. If however, you have a set of files for a different actor which may have a larger or smaller connectivity, then you will need to set this value.

In all cases where frequently running in a different location, it would probably be easier to modify the variables directly in the script (rather than re-specifying the relevant options each time using this option). More details are available in the **Variables** section.

FILES

These are the files that are queried by the program.

BN[1-8].txt

The files containing actors with Bacon (or other) numbers of 1-8 respectively (in the case of other actors, the depth of connection could be greater than eight). The complete set of files for a given actor (in the case of Kevin Bacon that is BN1.txt to BN8.txt) must be present. If files are missing then the user is informed of which ones are missing.

VARIABLES

AMDPATH

This variable contains the path to the AMD binaries on the system on which you wish to run this command. The default value is /proj/gp-dfb2/moviedb-3.5/bin.

LISTDIR

This variable contains the path to the AMD binaries on the system on which you wish to run this command. The default value is /proj/gp-dfb2/numbers/data.

SEARCHLEV

This variable contains the search depth (i.e. the maxmium level at which to search for actors). The default value is 8 (for searching for actors connected to Kevin Bacon).

SEE ALSO

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we’d like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

AUTHOR

The program and it’s accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The purpose of the program within the project is to allow people to query the Bacon number of an actor within the UNIX shell. A Perl version of this tool has been developed (*bn.pl*) to accept queries from a form (contained on *bn.html*), and return results using the standard game interface.

The *AMD* has been used with the very kind permission of the Internet Movies Database, for not only this program, but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies Database.

NAME

fastgen − Generate lists of actor names connected to a specified actor.

SYNOPSIS

fastgen [-o] [-s *n,m* | -i *a,..*] [-a *actor* | -p *path* | -d *dbpath*]

DESCRIPTION

Background Information

Fastgen is based on a concept of Bacon numbers, which are in turn based on the concept of Erdos numbers (initially devised by mathematician Paul Erdos). Erdos numbers are based on the co-authoring of mathematical papers, i.e. somebody with an Erdos number of 1 has directly co-authored a paper with Paul Erdos; whilst somebody with an Erdos number of 2 has co-authored a paper with somebody who has co-authored a paper with Paul Erdos, but has not directly co-authored a paper with Paul Erdos himself. For the purposes of the software engineering group project "The connected world of Frank Ramsey", this concept of Erdos numbers has been mapped onto the co-starring relationships between the various actors within the film industry. In particular the actor Kevin Bacon, who at the outset of the project was the subject of a considerable amount of conjecture regarding the roles that he had played, and how "connected" he was to other Hollywood actors (he was rumoured to be the best connected bit-part actor in Hollywood - investigation of this has shown this not to be the case). Hence the concept of Bacon numbers was born.

What it does

The purpose of *fastgen* is to generate lists of actors that fall into the various categories of Bacon number, e.g. list all those who have co-starred with Kevin Bacon (i.e. those with a Bacon number of 1) and then their co-stars in turn (i.e. those with a Bacon number of 2) and so on, until all actors that could be connected to Kevin Bacon have been listed (there are currently 8 such levels). *Fastgen* uses the off-line version of the Internet Movies Database (IMDb) known as the *A*lternative *M*ovies *D*atabase (*AMD*), to generate the lists, and as a result *requires* this package to be installed. The default operation of the program (when no arguments are given) is to generate the lists of actors with Bacon numbers from 1 to 8. This is done without any optimisation, and the time it takes to complete is largely dependant upon the processing power and disk access speed of the host machine. *Fastgen* isn't just limited to producing lists of actors connected to Kevin Bacon; by specifying the actor using the -a option, it is possible to set an actor of your choice; so for example, you could generate lists of actors with all the various Connery numbers if you chose Sean Connery. Also there is absolutely no restriction on how deep the level of connections can go, so you could find out who really is the best connected actor in Hollywood if you wished! ;-)

Options

-a *actor*

Allows the user to specify an actor other than the default (Kevin Bacon). One should remember that in order for the change to be meaningful, the actor's name should be specified in the standard *AMD* format, which is: Surname, Forename(s). In most cases it will be necessary to enclose the name in quotation marks, so that the whole name (including whitespace) is accepted. It should be noted that in cases where there is more than one occurrance of an actor's name within the *AMD*, a tagging system is used to differentiate between different actors with the same name. These tags take the form of a roman numeral enclosed in parentheses, where the roman numeral denotes the rank of the actor within the *AMD*. So for example, the actor Robert Powell (noted for a wide range of film and TV work in films such as The Italian Job (1969)) is listed in the *AMD* as: Powell, Robert (I). If you wish to generate files for such an actor, you will need to obtain the rank information, since without this, the *AMD* returns nothing (and therefore so will *fastgen*).

-p *path* Specify the path to be used when generating the files. The default is the current directory.

-d *dbpath*

Specify the path to the AMD tools when generating the actor lists. The default is the current directory.

-s *n,m* Generate all the data files from BNn.txt to BNm.txt inclusive. *Fastgen* uses the values 1 and 8 by default, which in combination with the default actor setting would generate the lists containing all the actors within the *AMD* that are linked to Kevin Bacon (at the time of writing). N.B.: If you specify the option more than once for the same invocation, the parameters specified with the last

occurrence of the option will be used.

-i *a,b,c,d,...*

Generate the individual data files BNa.txt, BNb.txt, BNc.txt, BNd.txt. The parameters to this option can either be specified as shown (a comma seperated list) or individually as single numbers following -i appearing several times on the command line (thus *fastgen* -i 1,2,3 is equivalent to *fastgen* -i 1 -i 2 -i 3). There only restriction on the number of individual data files you can request is the length of a line (80 characters).

-o This option turns on a optimisation that can increase the speed of the generation of a set of actor lists (the more you are generating the more worthwhile the use of this option becomes). On fast machines you may notice little difference in performance between a run with optimisation, and a run without it. However, on slower machines the performance increase can be significant (on an old system I had lying around, the difference is around 4.5 hours when generating the Bacon numbers from 1 to 8!). This optimisation is not enabled by default, because it requires extra disk space during generation (around 5-10Mb), which although not much these days, might be a problem for those with small /tmp partitions. Currently the program's default is optimised for both space and speed, which is a little slower, but doesn't require more than about 12-15Mb of space on /tmp during generation.

FILES

The following files are created in the current directory or the directory specified using the path (-p) option:

BN0.txt The base-level file containing just the name of Kevin Bacon or the actor specified by the actor (-a) option - in the usual *AMD* format. If not present when the program is executed, it will be generated. It is used during the generation of the lists. Once generation of lists is complete, it is left in the directory for reference, so that it is possible to see which actor is the source of a given set of lists.

BN[1-8].txt

The files containing actors with Bacon (or other) numbers of 1-8 respectively (in the case of other actors, the depth of connection could be greater than eight). It should be noted that in order to generate a list containing all the actors with a Bacon (or other) number of n, all the lists from 1 to n-1 must be present in the same directory. If this is not the case, generation will fail at the first missing file.

SEE ALSO

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we'd like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

BUGS

Due to the slightly crude nature of *fastgen*'s signal handler, it is necessary to send a SIGKILL (can be done using kill -9 PID in the shell) to terminate the execution of *fastgen* when it is being run using nohup. There are no problems with termination of the program when run in the either the foreground or background without nohup. The implementation of a signal handler was deemed necessary to clean up temporary files when the process was interrupted (by SIGINT) or terminated (by SIGHUP, SIGTERM, etc.). The current implementation does the job very well apart from when *fastgen* is run using nohup; where a termination signal results in the initial of a cleanup of /tmp, but does not terminate *fastgen* or any of it's child processes. It is not really a major problem, just a little inconvenient. It should be noted that the use of SIGKILL is **only** required if you wish to terminate the execution of *fastgen* before it has finished generating the requested lists. Normal termination due to an internal condition (i.e. the lists have been successfully completed, or an error has occurred during generation), is **not** affected and should pose no problems whatsoever (using nohup or otherwise). If you do happen to come across any bugs during use, then a bug report can be emailed to the author at: *ajt97c@cs.nott.ac.uk*, with a subject line containing "BUGS".

AUTHOR

The program and it's accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The purpose of the program within the project is to generate the Bacon numbers upon which the web-based game we are developing are based.

The *AMD* has been used with the very kind permission of the Internet Movies Database, for not only this program, but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies Database.

NAME

mbacon − Return the Bacon number of an actor.

SYNOPSIS

mbacon [-p *path* | -d *path*] [-s *n*]

DESCRIPTION

Background Information

Mbacon employs the concept of Bacon numbers, which are in turn based on the concept of Erdos numbers (initially devised by mathematician Paul Erdos). Erdos numbers are based on the co-authoring of mathematical papers, i.e. somebody with an Erdos number of 1 has directly co-authored a paper with Paul Erdos; whilst somebody with an Erdos number of 2 has co-authored a paper with somebody who has co-authored a paper with Paul Erdos, but has not directly co-authored a paper with Paul Erdos himself. For the purposes of the software engineering group project "The connected world of Frank Ramsey", this concept of Erdos numbers has been mapped onto the co-starring relationships between the various actors within the film industry. In particular the actor Kevin Bacon, who at the outset of the project was the subject of a considerable amount of conjecture regarding the roles that he had played, and how "connected" he was to other Hollywood actors (he was rumoured to be the best connected bit-part actor in Hollywood - investigation of this has shown this not to be the case). Hence the concept of Bacon numbers was born.

What it does

The purpose of *mbacon* is to query the lists of actors generated by the tool *fastgen* to allow a user to determine the Bacon number of a given actor. The term 'Bacon number' is employed loosely here, since although the default operation of *fastgen* is to generate lists of actors connected to Kevin Bacon, it can be used to generate lists of actors connected to another actor. In such cases one might have generated the actor lists for Sean Connery, and in using this tool would then effectively be determining the Connery number of an actor. Hence the term Bacon number is purely conceptual, since the default mode of operation of all the tools (including this one) is to work for Kevin Bacon since they were designed for the game Kevin Bacon's Dinner Parties. This tool differs from *bacon* in so far as the actors are taken from the standard input (they must be specified one per line) rather than being specified as an argument. *mbacon* returns the results to the standard output in the same order as that in which the actor names were received.

Options

-d *path* Allows the user to specify the path to the *AMD* binaries. This is useful when running on systems where tne *AMD* isn't in the path, or you want to query a different version of the database running on the same system. The default when this option is not specified is /proj/gp-dfb2/moviedb-3.5/bin, which relates to the installation on Crilly - the machine the code was developed on.

-p *path* Allows the user to specify the path to the location of the actor lists as generated by *fastgen*. As with the path to the *AMD* binaries, the default value of this (i.e. when the option is not specified) is /proj/gp-dfb2/numbers/data, which (as with the path to the *AMD* binaries) relates to the installation on Crilly - the machine that the game was developed on.

-s *n* This value represents the depth to which you wish to search for a connection with a given actor. The default value is 8, which relates to the actor Kevin Bacon. If however, you have a set of files for a different actor which may have a larger or smaller connectivity, then you will need to set this value.

In all cases where frequently running in a different location, it would probably be easier to modify the variables directly in the script (rather than re-specifying the relevant options each time using this option). More details are available in the **Variables** section.

FILES

These are the files that are queried by the program.

BN[1-8].txt

The files containing actors with Bacon (or other) numbers of 1-8 respectively (in the case of other actors, the depth of connection could be greater than eight). The complete set of files for a given actor (in the case of Kevin Bacon that is BN1.txt to BN8.txt) must be present. If files are missing

then the user is informed of which ones are missing.

VARIABLES

AMDPATH

This variable contains the path to the AMD binaries on the system on which you wish to run this command. The default value is /proj/gp-dfb2/moviedb-3.5/bin.

LISTDIR

This variable contains the path to the AMD binaries on the system on which you wish to run this command. The default value is /proj/gp-dfb2/numbers/data.

SEARCHLEV

This variable contains the search depth (i.e. the maxmium level at which to search for actors). The default value is 8 (for searching for actors connected to Kevin Bacon).

SEE ALSO

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we'd like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

AUTHOR

The program and it's accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The purpose of the program within the project is to allow people to query the Bacon number of an actor within the UNIX shell. This tool is a variant of the tool *bacon*.

The *AMD* has been used with the very kind permission of the Internet Movies Database, for not only this program, but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies Database.

NAME

name2url − Outputs the URL of the IMDb filmography page for a specified actor.

SYNOPSIS

name2url "*name*"

DESCRIPTION

What it does

The program takes one argument (mandatory), which is the name of an actor within the *Alternative Movies Database (AMD)*, and returns the URL for that actor's filmography page on the *IMDb* web site (http://www.imdb.com).

ARGUMENT FORMAT

The argument for *name2url* is **mandatory**, and should consist of an actor's name, in the same format that is used by the *AMD* software:

Surname, Forename(s) (#)

N.B.: In cases where there is more than one occurrance of an actor's name within the *AMD*, a tagging system is used to differentiate between different actors with the same name. These tags take the form of a roman numeral enclosed in parentheses, where the roman numeral denotes the rank of the actor within the *AMD*. So for example, the actor Robert Powell (noted for a wide range of film and TV work in films such as The Italian Job (1969)) is listed in the *AMD* as: Powell, Robert (I). If you wish to link to the correct actor at the Internet Movies Database website, then you would be advised to obtain the tagging information for the actor. Normally this is not a problem, since *name2url* is used by scripts that have the full details about an actor.

EXAMPLES

Entering:

name2url "Hanks, Tom"

Will return:

http://www.imdb.com/M/person-exact?Hanks%2C%20Tom

Entering:

name2url "Albiñana, Francisco (I)"

Will return:

http://www.imdb.com/M/person-exact?Albi%F1ana%2C%20Francisco%20%28I%29

SEE ALSO

Related manual pages

title2url(1), fastgen(1), weekly_upd(1), bacon(1), mbacon(1)

Other information

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we'd like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

BUGS

If any bugs are found during the use of this program, then a bug report can be emailed to the author at *ajt97c@cs.nott.ac.uk* with a subject line containing "BUGS" and the name of the program in question.

AUTHOR

The program and it's accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The purpose of the program within the project is to generate the URLs for each actor's filmography page at the Internet Movies Database website. It is used by the PERL scripts that support the whole HTML based game and parts of the Java applet.

The *AMD* is used with the very kind permission of the Internet Movies Database, for not only this program,

but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies Database.

NAME

title2url − Outputs the URL of the IMDb information page for a specified film.

SYNOPSIS

title2url "*name*"

DESCRIPTION

What it does

The program takes one argument (mandatory), which is the title of a film within the *Alternative Movies Database (AMD)*, and returns the URL for that film's information page on the *IMDb* web site (http://www.imdb.com).

ARGUMENT FORMAT

The argument for *title2url* is **mandatory**, and should consist of a films title, in the same format that is used by the *AMD* software:

Title (Year of release)

N.B.: In cases where there is more than one occurrance of a film's title within the *AMD*, a tagging system is used to differentiate between different films with the same title. These tags take the form of the year of release enclosed in parentheses. So for example, there have been two versions of the film Cape Fear, one in 1962 and the other in 1991. If you wish to link to the correct film information page at the Internet Movies Database website, then you would be advised to obtain the year of release before sending a query as omission of this will result in a general page consisting of all of the releases under a particular title.

EXAMPLES

Entering:

title2url "Cape Fear (1991)"

Will return:

http://www.imdb.com/M/title-exact?Cape%20Fear%20%281991%29

Entering:

title2url "Champions, The (1978)"

Will return:

http://www.imdb.com/M/title-exact?Champions%2C%20The%20%281978%29

SEE ALSO

Related manual pages

name2url(1), fastgen(1), weekly_upd(1), bacon(1), mbacon(1)

Other information

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we'd like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

BUGS

If any bugs are found during the use of this program, then a bug report can be emailed to the author at *ajt97c@cs.nott.ac.uk* with a subject line containing "BUGS" and the name of the program in question.

AUTHOR

The program and it's accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The program was produced to generate the URLs to the film information pages at the Internet Movies Database website.

The *AMD* is used with the very kind permission of the Internet Movies Database, for not only this program, but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies

Database.

NAME

weekly_upd − Update the game's supporting systems

SYNOPSIS

weekly_upd [-p *path* | -d *path* | -o *path* | -c *path*]

DESCRIPTION

What it does

The purpose of *weely_upd* is to fetch the AMD plain text data files for the current week and then use them to regenerate the AMD for the current week in a seperate instance of the database which lives within the main live instance. Once the AMD has been generated for the week, it is the used to regenerate the actor lists for the week using *fastgen*. Once both the AMD and the actor lists have been regenerated, they can both be moved live so that the complete set of data required for the game can go live all at once. Whilst the data is being moved live, execute permissions on the AMD binaries are switched off, then once the data is moved live, the execute permissions are switched back on. Just before the actor lists are moved, those for the previous week are tarred and then gzipped and placed in the archive subdirectory (named with the date and time of the update).

Options

-c *path* Allows the user to set up a PATH if the script is to be run under a program with it's own environment like *cron*. The path supplied should be the standard colon seperated list.

-d *path* Allows the user to specify the path to the *AMD*. This is useful when running on systems where tne *AMD* isn't in the path, or you want to regenerate a different version of the database running on the same system (done by default). The default when this option is not specified is /proj/gp-dfb2/moviedb-3.5, which relates to the installation on Crilly - the machine the code was eveloped on.

-p *path* Allows the user to specify the path to the location of the system maintenance tools like *fastgen*. As with the path to the *AMD* binaries, the default value of this (i.e. when the option is not specified) is /proj/gp-dfb2/numbers, which (as with the path to the *AMD* binaries) relates to the installation on Crilly - the machine that the game was developed on.

-o *path* Allows the user to specify the path for the output from the commands executed by *weekly_upd*. The default value for this is a file called cmdout.log in the directory containing the system maintenance tools (as specified by the **-p** option). The complete path could be changed to something like */dev/null* to prevent any output, which can be useful when running under *cron*.

In all cases where frequently running in a different location, it would probably be easier to modify the variables directly in the script (rather than re-specifying the relevant options each time using this option). More details are available in the **Variables** section.

VARIABLES

AMDPATH

This variable contains the path to the version of AMD on the system on that you wish to update. The default value is /proj/gp-dfb2/moviedb-3.5.

LISTDIR

This variable contains the path to the actor list maintenance tools (e.g. *fastgen*) on the system on which you wish to run this command. The default value is /proj/gp-dfb2/numbers.

LOG The path including filename if appropriate of where to send the output from the commands executed within this script. The default value is a file called cmdout.log in the same directory as the system maintenance tools.

SEE ALSO

The web page for the "connected world of Frank Ramsey" project that this software was produced for: *http://crilly.cs.nott.ac.uk/bacon*. The Internet Movies Database, whom we'd like to thank for very kindly letting us use the tools for the completion of this program and other areas of the project. Their main web page is: *http://www.imdb.com*, whilst details of their off-line database the *AMD* can be found at: *http://us.imdb.com/interfaces*.

AUTHOR

The program and it’s accompanying manual page were solely written by Anthony Jaroslav Truhlar, as part of a second year software engineering group project (the connected world of Frank Ramsey) at the University of Nottingham. The purpose of the program within the project is to regenerate the AMD and actor lists upon which the game is based. On the actual site, it is run within the cron daemon every day to ensure that as soon as the data at IMDb is updated, ours is updated.

The *AMD* has been used with the very kind permission of the Internet Movies Database, for not only this program, but also other areas of the project. Any enquiries concerning this program or any other area of the project and should be addressed to either the author of the program (email address above) or the group as a whole at: *gp-dfb2@cs.nott.ac.uk*. Enquires regarding the *AMD* should be addressed to the Internet Movies Database.

Appendix E

A

Adler, Matt
Allan, Richie
Alvarado, Angela
Anderson, Louie
Anderson, Stanley
Appel, Peter
Aquino, John
Asner, Edward
Bank, Yudie
Barry, Thom
Belcher, Patricia
Bellaver, Harry
Berezin, Tanya
Bergen, Candice
Bergeron, Loys T.
Bodison, Wolfgang
Bolender, Bill
Bond, Sudie
Bowz, Eddie
Brenner, Eve
Brogger, Ivar
Brolin, Josh
Brothers, Dr. Joyce
Burnette, Olivia

B

Twomey, Anne
Waits, Tom
Sedgwick, Kyra
Turturro, Aida
Turturro, Aida
White, Stanley
Warner, Rick
Winchester, Maud
Moriarty, Cathy
Webster, Byron
Webster, Byron
Williams, Meadow
Whaley, Frank
Zelniker, Michael
O'Grady, Gail
Tammi, Tom
Warlock, Dick
Plana, Tony
Whaley, Frank
Bellaver, Harry
Koteas, Elias
Pendleton, Austin
Webster, Byron
Wiest, Dianne

C

Zucker, Charles
Yerkes, Mark
Zahrn, Will
Zamora, Dion
Zajonc, Robert ‘Bobby Z’
Zito, Sonny
Zwerling, Darrell
Zerbe, Anthony
Winkler, Margo
Zucker, Charlotte
Zadora, Pia
Zimmer, Norma
Youmans, William
Zuber, Marc
Wilhoite, Kathleen
York, Michael
Zinn, Donald
Zorich, Louis
York, Rachel
Young, Alan
Zimmer, Constance
Zima, Yvonne
Zucker, Charlotte
Wright, Amy

D

Zuniga, Daphne
York, Jay
Zobel, Richard
Wortell, Holly
Zúñiga, José
Zito, Chuck
Zabriskie, Grace
Zappa, Moon Unit
Zimmerman, Natalie
Zuckert, Bill
Zuckert, Bill
Young, Alan
Woods, Richard
Wincott, Michael
Zorek, Michael
Zazula, Tony
Z'Dar, Robert
Zumwalt, Rick
Zorn, Danny
Zimmer, Norma
Wynter, Sarah
Zima, Madeline
Zuckert, Bill
Zussin, Victoria

A

Cacciotti, Tony
Campbell, Neve
Cassell, Paul
Castillo, Gerald
Chaban, Michael
Chamberlin, Beth
Citera, Tommy
Clayburgh, Jill
Clemenson, Christian
Coco, James
Combs, Gary
Cory, Kenneth
Coryell, Bradley
Cray, Robert
Criscuolo, Lou
Cromwell, David
Crone, Penny
Curry, Russell
Curtis, Liane Alexandra
Cushman, Jessica
D'Annibale, Frank
Dangler, Anita
Daniel, Joshua
Davine, Carolyn
De Angelis, Rosemary
De Sosa, Ruth
Dixon, MacIntyre

B

Woodard, Charlayne
Tammi, Tom
Pendleton, Austin
Walter, Perla
Webb, Chloe
Stern, Jenna
Plana, Tony
Plana, Tony
Twomey, Anne
Plana, Tony
Reiner, Tracy
Wimmer, Brian
Stern, Jenna
Wright, Jenny
Parks, Van Dyke
Tobolowsky, Stephen
Roberts, Julia
White, Stanley
Warlock, Dick
Sutherland, Kiefer
Warner, Rick
Webster, Byron
Twomey, Anne
Windom, William
Stone, Sharon
Strathairn, David
Sutherland, Donald

C

Wynne, Jonathan
York, Michael
Zima, Yvonne
Yue, Marion Kodama
Zemeckis, Alexander
Poppel, Marc
Zorich, Louis
Zorich, Louis
Zane, Lisa
Zorich, Louis
Ziker, Dick
Ziering, Ian
Wilson, Rita
Springsteen, Bruce
Zobel, Richard
Smith, Charles Martin
Wheeler, Ed
Z'Dar, Robert
Zagarino, Frank
Wagner, Lindsay
Young, William Allen
Zucker, Charlotte
Winslow, Michael
Viviani, Joe
Zomina, Sonia
Zumwalt, Rick
Zimmerman, J.R.

D

Yaghjian, Kurt
Zazula, Tony
Zima, Madeline
Ziskie, Dan
Wright, Robin
Zully, Stewart J.
Zumwalt, Rick
Zumwalt, Rick
Zuniga, Daphne
Zumwalt, Rick
Zozzora, Carmine
Yeager, Biff
Zully, Stewart J.
Waters, Roger
Zegler, Paul
Zale, Dan
Zayas, David
Zito, Chuck
Z'Dar, Robert
Young, Richard
Zabriskie, Grace
Zuckert, Bill
Zuniga, Daphne
Zerbe, Anthony
Woodruff, Largo
Zorich, Louis
Zeppieri, Richard

A

Donovan, Jeffrey
Douglas, Diana
Dugan, Dennis
Edwards, Daryl
Falconer, Deborah
Feagin, Hugh
Fleming, Connie
Flockhart, Calista
Forsberg, Grant
Frank, Marilyn Dodds
Franklin, Don
Gardner, Ashley
Garion, Buddy
Geter, Leo
Gibbs, Keith

B

Tammi, Tom
Vaughn, Ned
Winchester, Maud
Plana, Tony
Schell, Maximilian
Warner, Rick
Strathairn, David
Tammi, Tom
Plana, Tony
Margolis, Mark
Vernon, John
Twomey, Anne
Webster, Byron
Webster, Byron
Webster, Byron

C

Wheeler, Timothy
Zimbalist Jr., Efrem
Zurk, Steve
Zorich, Louis
Zech, Rosel
Weenick, Annabelle
Woodward, Joanne
Zelman, Daniel
Zorich, Louis
Zaremby, Justin
Wright, Teresa
Zane, Lisa
Wynant, H.M.
Young, Sean
Zucker, Charlotte

D

Zazula, Tony
Young, Bruce A.
Zeigler, Jesse
Zumwalt, Rick
Zischler, Hanns
Zabriskie, Grace
Zorich, Louis
Zazula, Tony
Zumwalt, Rick
Zion, Rabbi Dr. Joel Y.
Zapata, Carmen
Zuniga, Daphne
Zuckert, Bill
Zuckert, Bill
Zuckert, Bill

Appendix F
Problems with AMD

This appendix will attempt to give an insight into some of the problems that we have had as a group with the Alternative Movies Database. Most of the problems are caused by the way in which films are classified within the AMD, in particular, the tagging system that it uses.

First of all, all films within the AMD are classified using a tag to denote the year of release, this is mandatory for our purposes since failure to specify the appropriate tag will result in the entries for all variants of a given film title to be displayed. In addition to this, actor names can contain a classification tag; if there is more than one instance of a given name within the AMD, the various actors are differentiated by a tag containing a roman numeral to denote the particular instance of a given name.

In addition to the main classification tags, other tags can be added for awards that a given film or actor has received, and then also for character names, etc. within a particular instance of a film (character names are generally enclosed in square brackets). Unfortunately there is no real standard beyond the basic tags, and there are a huge number of tags that go beyond the ones documented in the manual. The situation is made worse by the fact that the database is added to by volunteers which results in inaccuracy of the data, and problems with the tags such as spelling mistakes, unclosed parenthesis and so on.

One of the major problems with all of this extra tagging information (apart from the year of release and instance number which are crucial) is that it has to be stripped off if it is to be used in subsequent queries. Unfortunately this is difficult in the shell, because as we found through more in-depth investigation, you cannot get away with using the basic UNIX tools. The first method we used to strip off the tags within the shell was crude, it simply used a long `sed` command that attempted to match individual tags and then remove them. Of course with the plethora of tags around this really isn't practical, and we really needed to strip off all but those brackets containing either Arabic or Roman numerals. However, it wasn't quite as simple as this since subsequent code that attempted to do this was not matching all the films properly – the reason being that certain films have text in brackets within their title. To further muddle the picture, there are cases of square brackets appearing within round brackets and so on. Clearly this began to rule out a Bourne shell based implementation, because the pattern matching that we needed to do went beyond what could be safely done with regular expressions; i.e. you needed to keep track of the number of brackets that had occurred, and also within what context each bracket occurred. This lead us to conclude that an implementation using C might be better, albeit a little messy. The complexity of the `stripdesc` function is clearly a testament to the problems experienced – in fact it took much longer to effectively deal with this problem than with many other areas of the program.

Beyond the tags

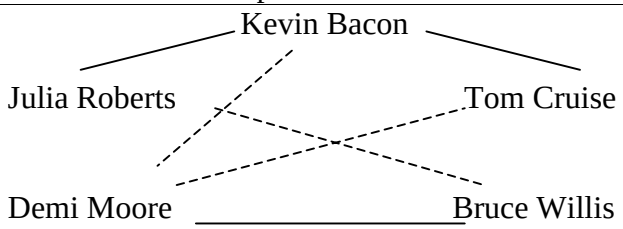
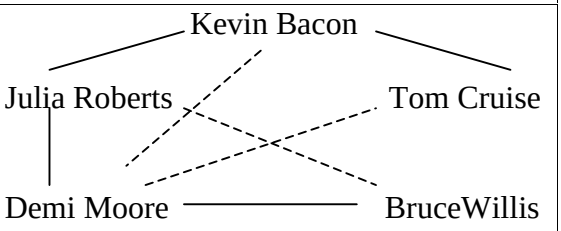
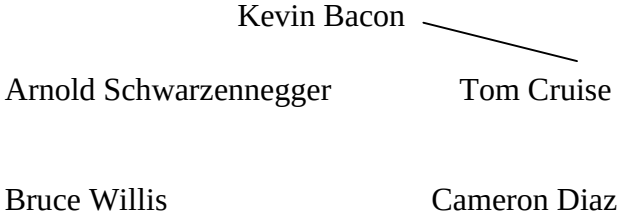
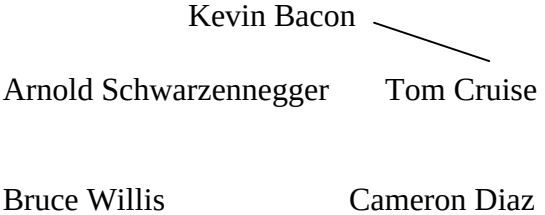
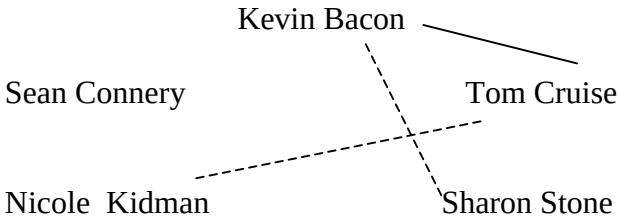
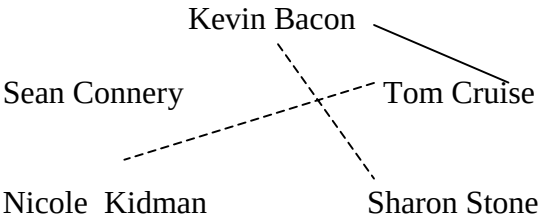
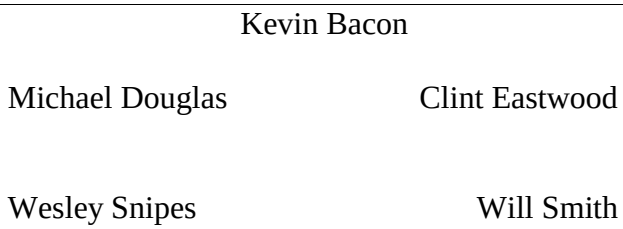
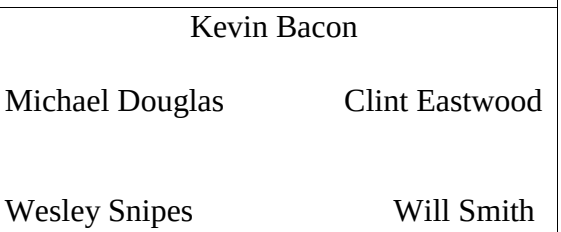
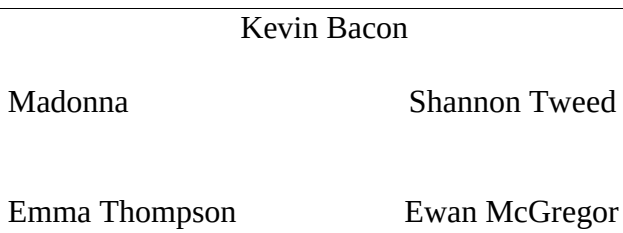
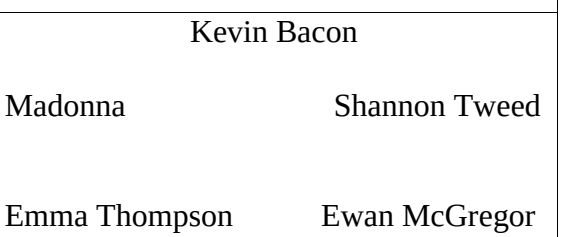
It is not just the removal of tags that causes problems. There are certain major problems with the database due to the use of user supplied data.

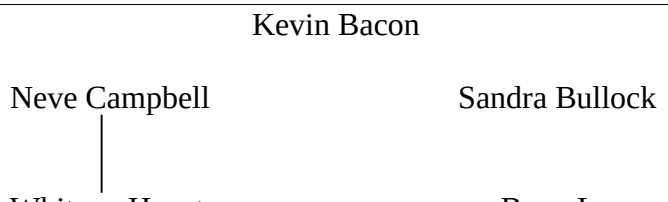
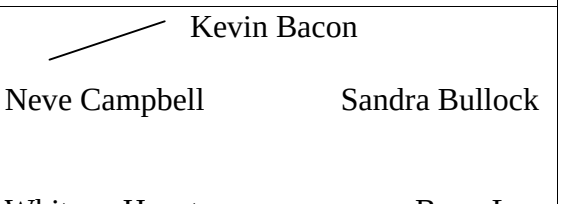
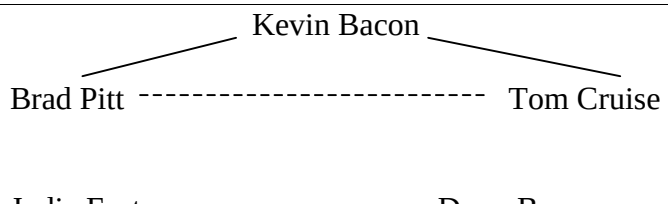
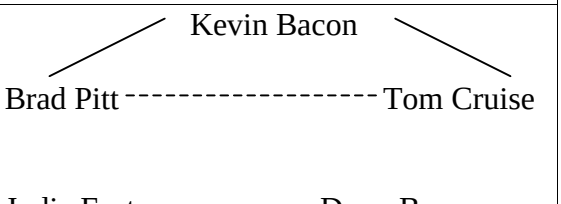
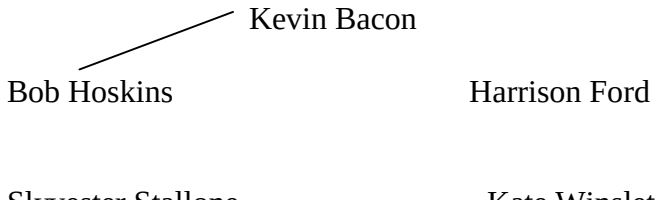
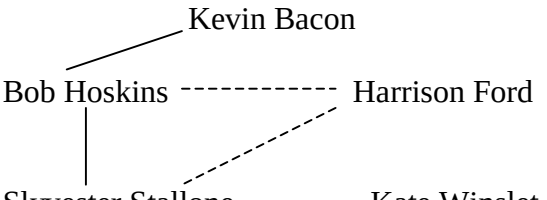
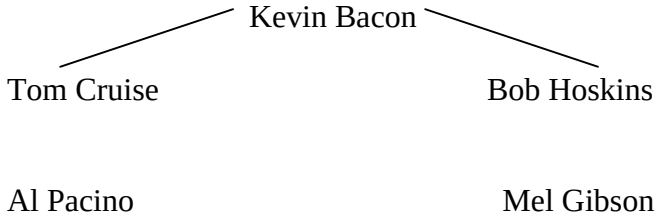
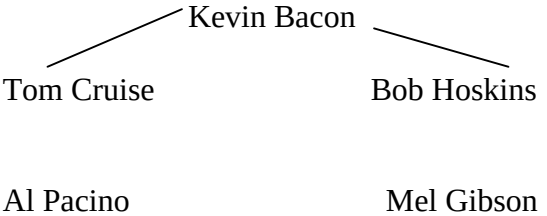
One major problem we found was that the system would admit that Elizabeth Taylor and Elizabeth Perkins had co-starred, but would not do it in either order. This of course causes problems when it comes to the dinner parties.

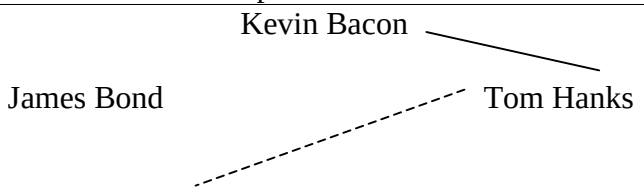
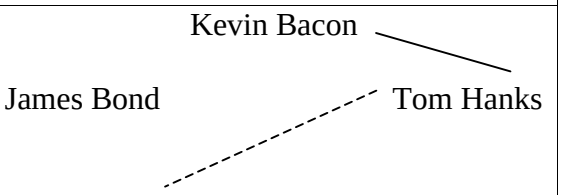
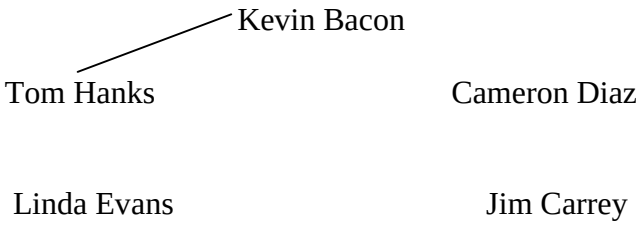
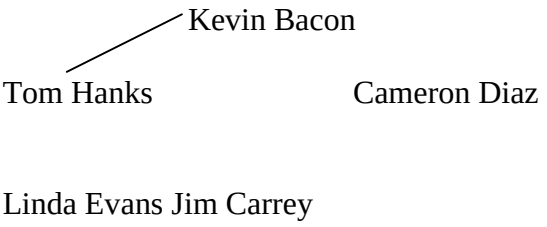
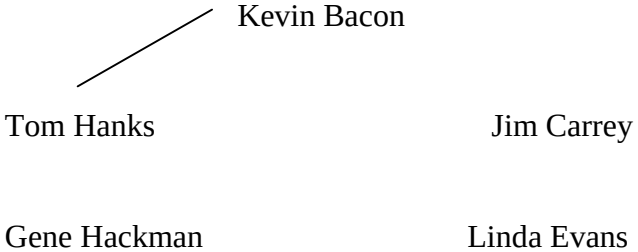
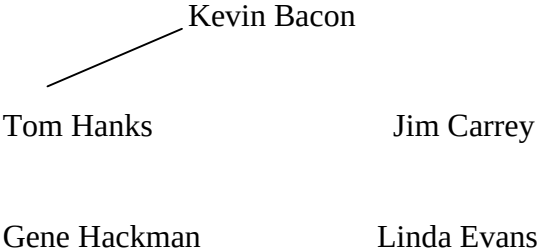
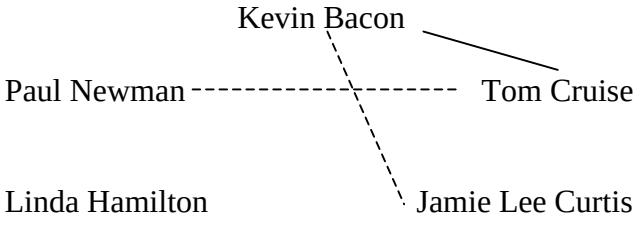
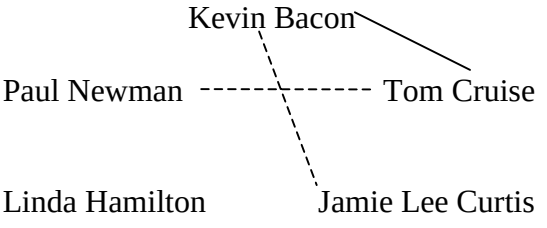
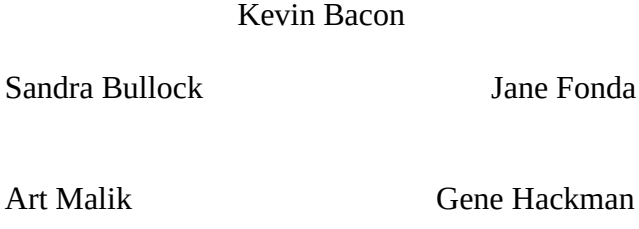
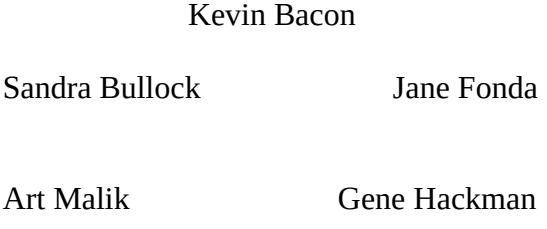
Another problem is that despite that fact that the database is only supposed to return all matches for a film when the year isn't specified, there are a number of cases when it does this even when the year is specified. There isn't much we can do about this apart from stripping off the titles of all films in the category since it is a problem with their system.

Another known issue is with a documentary called a century of cinema which isn't technically a film but is listed as such (it is a documentary). The major problem of course is that since it is a documentary on films, it treats a large number of actors as having starred in it.

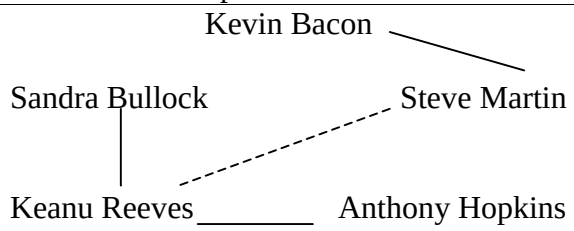
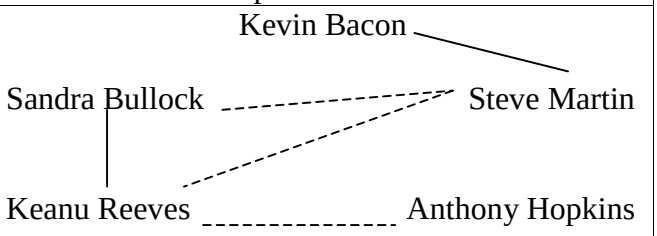
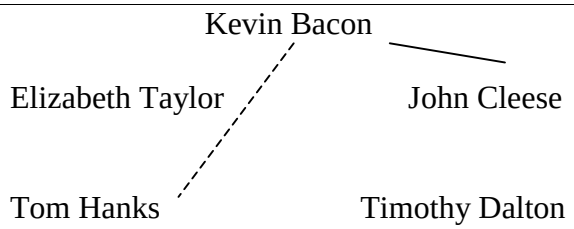
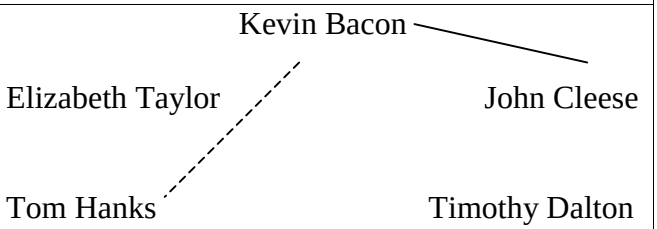
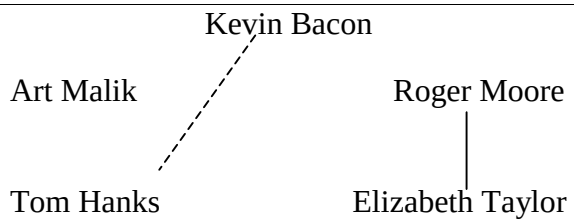
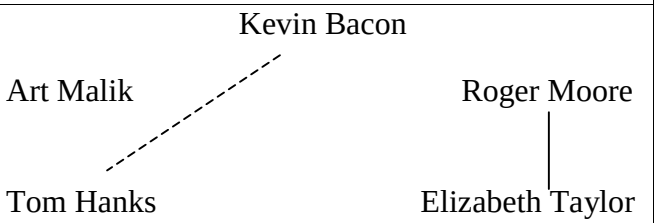
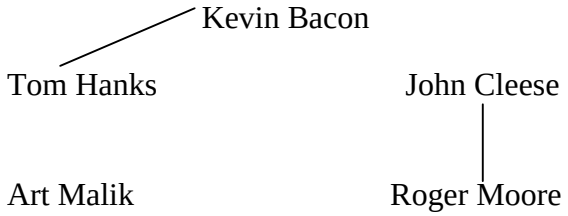
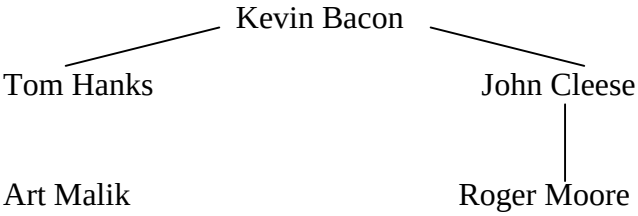
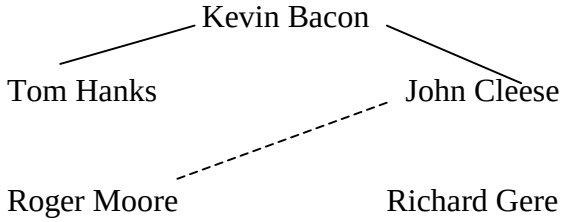
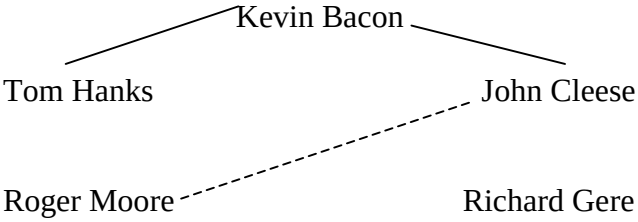
Appendix G - Research / Testing on the Final Web Game

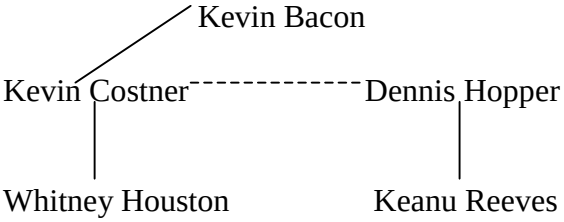
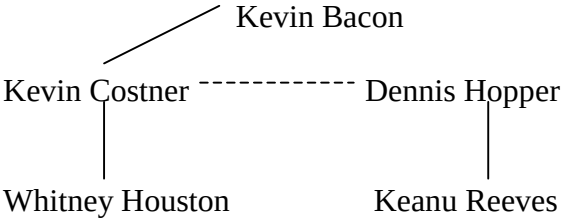
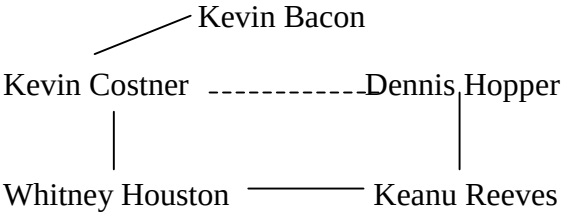
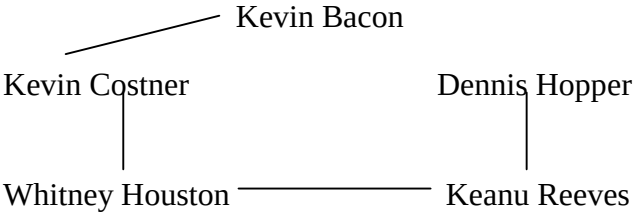
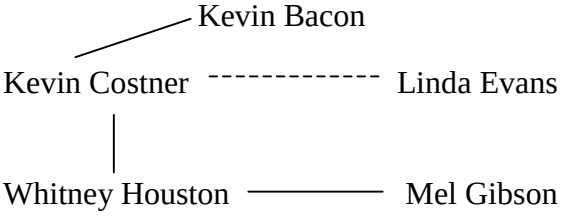
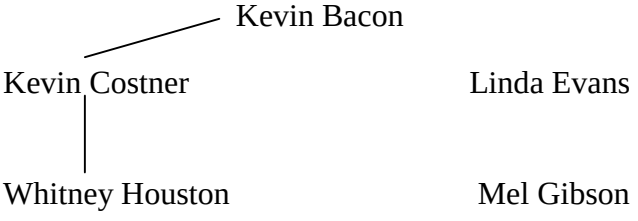
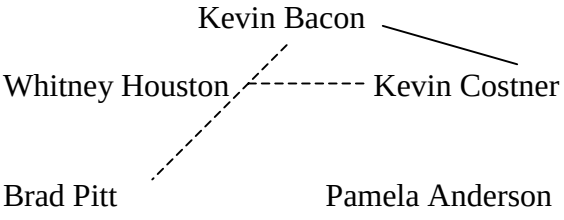
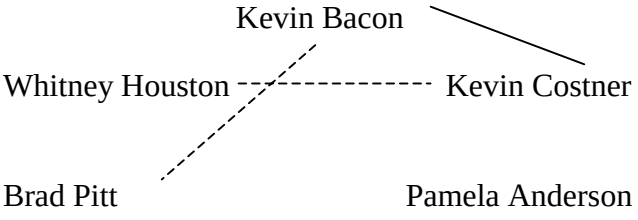
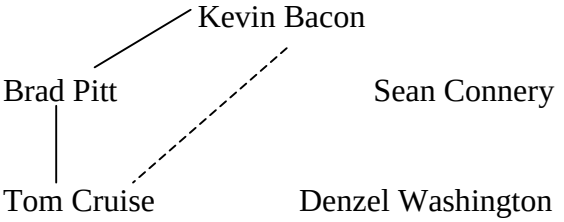
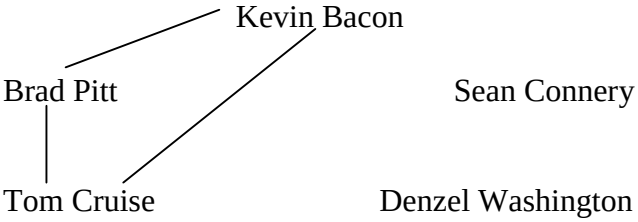
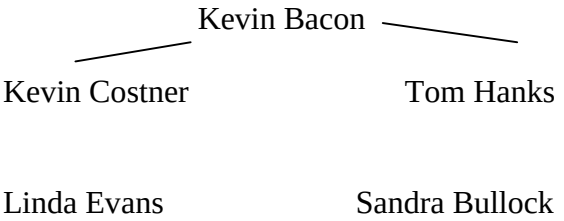
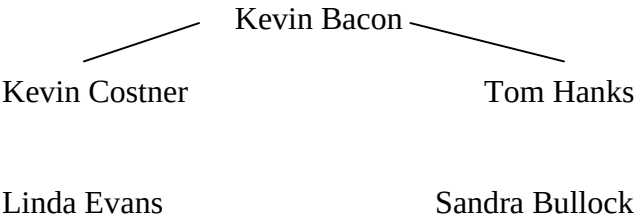
	Expected result	Actual Web result
Party 1		
Party 2		
Party 3		
Party 4		
Party 5		

	Expected result	Actual Web result
Party 6		
Party 7		
Party 8		
Party 9		
Party 10		

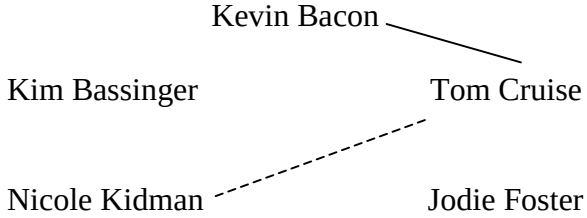
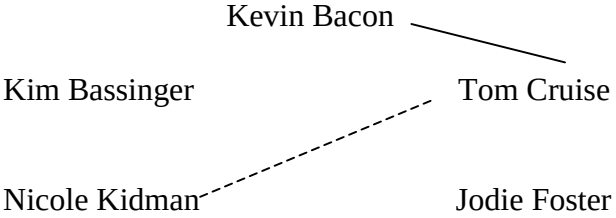
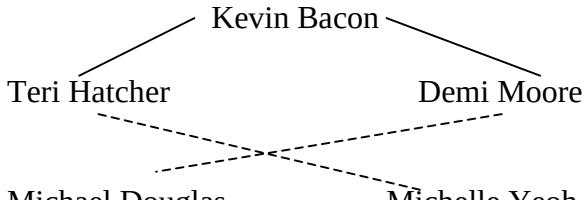
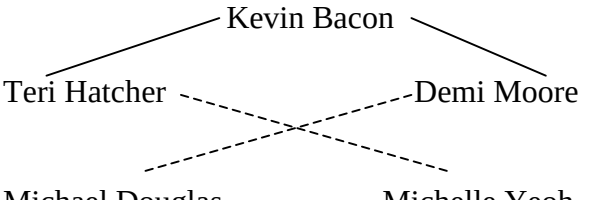
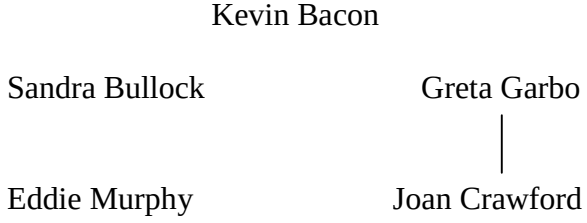
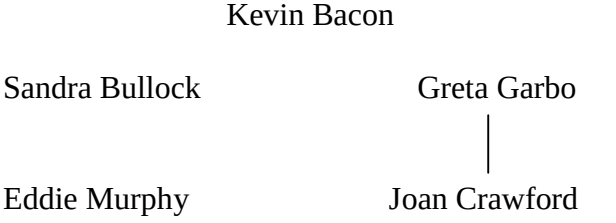
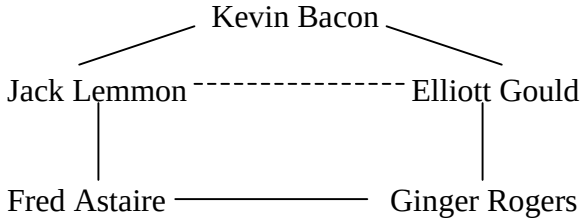
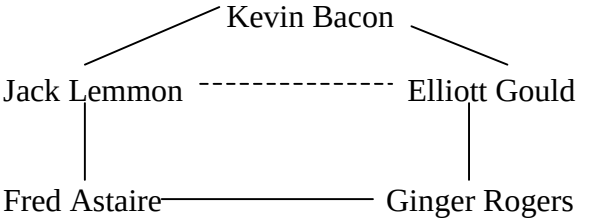
	Expected result	Actual Web result
Party 11		
Party 12		
Party 13		
Party 14		
Party 15		

	Expected Result	Expected Result
Party 16	Kevin Bacon Fred Astaire ---------- Ginger Rogers Pele ———————— BobbyMoore	Kevin Bacon Fred Astaire ---------- GingerRogers Pele ———————— Bobby Moore
Party 17	Kevin Bacon Peter Sellers Michael Palin | Bob Hope Christopher Lloyd	Kevin Bacon Peter Sellers Michael Palin | Bob Hope ChristopherLloyd
Party 18	Kevin Bacon \ Michael Caine Noah Wyle Brent Spiner Billy Dee Williams	Kevin Bacon \ Michael Caine Noah Wyle Brent Spiner Billy Dee Williams
Party 19	Kevin Bacon John Cussack Brian Glover Kim Basinger Harrison Ford	Kevin Bacon John Cussack Brian Glover Kim Basinger Harrison Ford
Party 20	Kevin Bacon River Phoenix Kathy Bates Bill Pullman Mick Jagger	Kevin Bacon River Phoenix Kathy Bates Bill Pullman Mick Jagger

	Expected Result	Expected Result
Party 21		
Party 22		
Party 23		
Party 24		
Party 25		

	Expected Result	Actual Web Result
Party 26		
Party 26		
Party 27		
Party 28		
Party 29		
Party 30		

	Expected Results	Actual Web Results
Party 31	Kevin Bacon Uma Thurman ---------- Richard Gere Emma Thompson Sean Bean	Kevin Bacon Uma Thurman ..... Richard Gere Emma Thompson Sean Bean
Party 32	Kevin Bacon Bean Bridges ..... Jeff Bridges Don Marino Greta Garbo	Kevin Bacon Bean Bridges ..... Jeff Bridges Don Marino Greta Garbo
Party 33	Kevin Bacon Neve Campbell Sharon Stone Robin Williams Joe Pesci	Kevin Bacon Neve Campbell Sharon Stone Robin Williams Joe Pesci
Party 34	Kevin Bacon Alec Baldwin David Bowie Andy Garcia Bo Derek	Kevin Bacon Alec Baldwin David Bowie Andy Garcia Bo Derek
Party 35	Kevin Bacon Jennifer Aniston Woody Allen Gary Oldman Madonna	Kevin Bacon Jennifer Aniston Woody Allen Gary Oldman Madonna

	Expected Results	Actual Web Results
Party 36		
Party 37		
Party 38		
Party 39		
Party 40