CSCI131

Introduction to Computer Systems

Task Group 4: Libraries and the build process

Complete the exercises, and try to get them marked, before attempting the assignment!

There are several exercise tasks in this task group as detailed below. At the end of this document there is a link to the assignment. You should try to complete all the exercise tasks before attempting the assignment task.

These tasks provide a little more experience in the use of libraries, and with the build process using "make".

Task 1: More or less 'standard libraries' – gd and gdbm: 3 marks

For this task, you simply copy my example code for manipulating data records in a "dbm" style 'database'. The example also uses the gd graphics library along with some functions to "base-64" encode data. You will build the example first in NetBeans; when it works, you will recreate it using your own 'makefile'.

dbm libraries

*"The dbm library was a simple database engine, originally written by Ken Thompson and released by AT&T in 1979. The name is a three letter acronym for DataBase Manager, and can also refer to the family of database engines with APIs and features derived from the original dbm." (*So says Wikipedia*)*

A "dbm" database is simple. There are many records in a disk file; each is characterised by a unique primary key. There are no other indexes. Records are just blocks of bytes; a dbm system doesn't interpret them in any way. You cannot search for records where fields have specific values. All you can do is store and retrieve complete records. But, a dbm system will be fast – lots faster than a relational database such as MySQL.

Most current implementations seem to stem from Berkeley DB – a software package that was released ~1986 as part of the BSD Unix variant. On Ubuntu, the current implementation is via the gdbm library.

gdbm

The gdbm library includes the following functions (details suppressed until later):

gdbm_open (…)
gdbm_close (…)

Open and close a program's connection to a gdbm database.

gdbm_store (…)
gdbm_fetch (…)
Store and retrieve keyed records.

gdbm_exists (…)
Check if there is a record matching a specified key.
gdbm_delete (…)
Delete a record identified by a key.

gdbm_firstkey (…)
gdbm_nextkey (…)
These functions allow a program to iterate through all keys (in some random order).

gdbm_reorganize (…)
Programs that create and delete records at a high frequency should occasionally call "reorganize"; the gdbm code will tidy up its disk file reclaiming space used by deleted records.

gdbm_sync (…)
Effectively a flush operation; make sure all changes have been written to disk.

gdbm_strerror (…)
Many of the gdbm calls will return an integer error code when something fails; strerror converts this error code to a message.

gdbm_setopt (…)
gdbm_fdesc (…)
These are for use by specialists. The setopt method allows control of aspects of caching etc. The fdesc method returns the "file descriptor" used to access the file; this might be needed if you wanted to try to lock the file to prevent concurrent updates by different processes.

Gdbm files are not sparse like those illustrated in an earlier exercise.

gd library

"The GD Graphics Library is a graphics software library by Thomas Boutell and others for dynamically manipulating images. Its native programming language is ANSI C, but it has interfaces for many other programming languages." (*So says Wikipedia*)

The gd code is widely used (from C, C++, PHP, Python, Perl, …). The documentation isn't great; T Boutell lost interest long ago and now it is maintained by Pierre Joye and others who are primarily interested in its use as a graphics library for PHP. Sometimes, it's easiest to read the PHP documentation!

The use in this example is limited. The program will read in an image from a file (.gif, .jpg, .png formats). It scales the image. The scaled image is converted to .jpg format. The data are then "base64" encoded into a text string that can be added to the database, and gets used directly when displaying a picture in a generated HTML page.

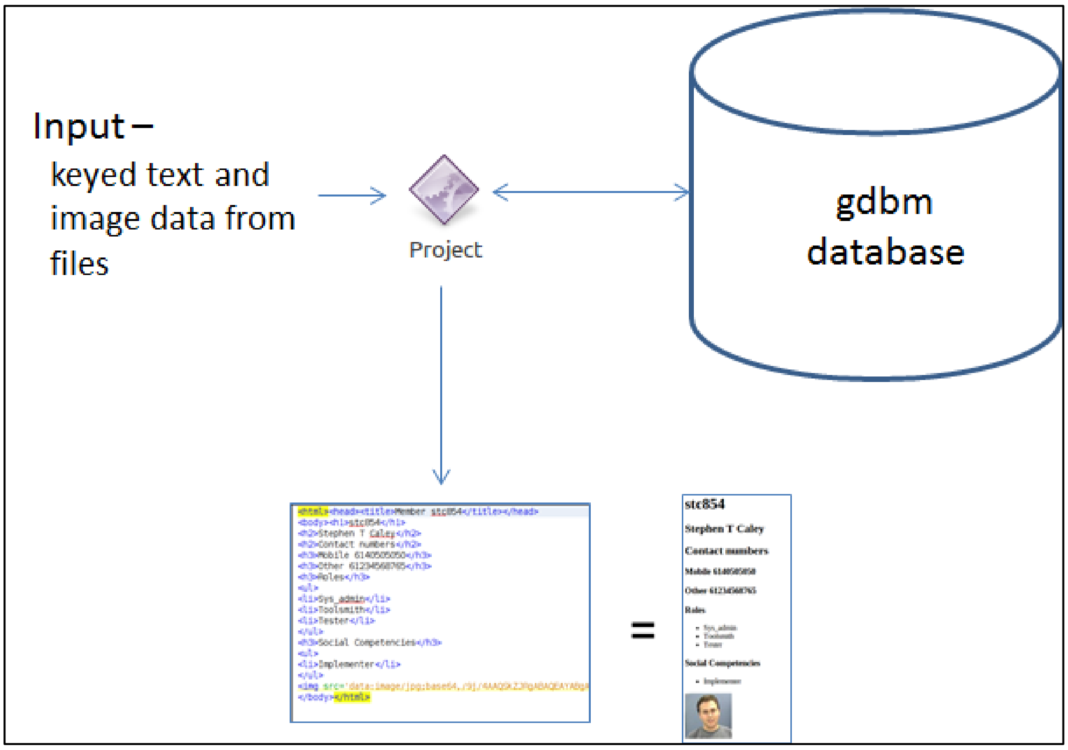
base64

Functions to encode binary data into base64 text strings, and decode base64 text string to retrieve the original binary data, should be part of some standard C library. Base64 text strings have had an important role in encoding data since at least the early 1990s – e.g. they are used when adding data to email.

But there are no standard base64 functions. Apache, and others have published versions of the code on the Internet. We will be using the versions from polarssl.org. The files base64.h and base64.c are in the Resources folder on the web site.

Application

The contrived application is for keeping records of members of a project group – user-ids, names, contact numbers, role in project, social competencies. (*Role in project? Social Competencies? You will learn more about project work in CSCI222.*) The application allows for records to be entered, and can on demand create "pretty" HTML pages with details of members.



```
<html><head><title>Member stc854</title></head>
<body><h1>stc854</h1>
<h2>Stephen T Caley</h2>
<h2>Contact numbers</h2>
<h3>Mobile 6140505050</h3>
<h3>Other 61234568765</h3>
<h3>Roles</h3>
<ul>
<li>Sys_admin</li>
<li>Toolsmith</li>
<li>Tester</li>
</ul>
<h3>Social Competencies</h3>
<ul>
<li>Implementer</li>
</ul>
<img src='data:image/jpg;base64,/9j/4AAQSkZJRgABAQEAYABgA
</body></html>
```

Use

Typical "menu select" loop (you must have done things like this in CSCI114):

```
Processing options:
0 => Quit; 1 => Add data; 2 => Generate member page; 3 => List members
Opt>1
Enter member details:
```

Entering data

Supply details of project group member:

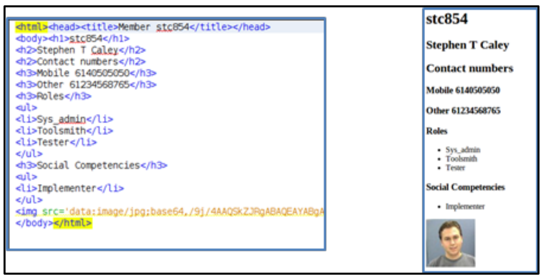
```
Opt>1
Enter member details:
Family name    : Caley
Given name     : Stephen
Initials       : T
Mobile #       : 6140505050
Other phone #  : 61234568765
Unix id        : stc854
Enter roles>?
Choices are:    Manager
        Sys_admin
        Dba
        Webmaster
        Librarian
        Leadprogrammer
        Designer
        UMLArtist
        Programmer
        Documenter
        Toolsmith
        Tester
        Versionmanager
        Pizza_boy
>Sys_admin
>Toolsmith
>Tester
>-
Enter competencies>?
Choices are:    Originator
        Coordinator
        Driver
        Monitor
        Implementer
```

Get list of all project group members:

```
Opt>3
wcl666
ctc333
stc854
dwp395
```

Pick a member and generate a HTML output file that can be viewed in a browser:

```
Opt>2
Enter id : stc854
```



```
<html><head><title>Member stc854</title></head>
<body><h1>stc854</h1>
<h2>Stephen T Caley</h2>
<h2>Contact numbers</h2>
<h3>Mobile 6140505050</h3>
<h3>Other 61234568765</h3>
<h3>Roles</h3>
<ul>
<li>Sys_admin</li>
<li>Toolsmith</li>
<li>Tester</li>
</ul>
<h3>Social Competencies</h3>
<ul>
<li>Implementer</li>
</ul>
<img src='data:image/jpg;base64,/9j/4AAQSkZJRgABAQEAYABgA
</body></html>
```

stc854

Stephen T Caley

Contact numbers

Mobile 6140505050

Other 61234568765

Roles

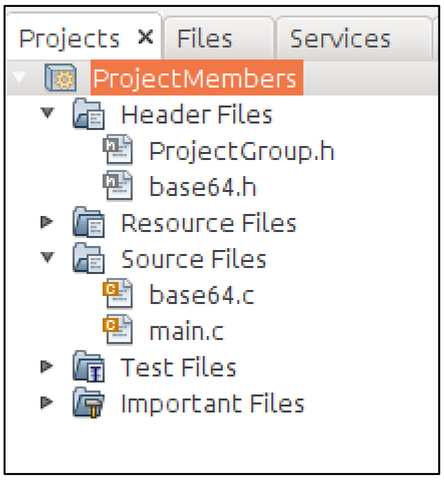
- Sys_admin
- Toolsmith
- Tester

Social Competencies

- Implementer

NetBeans project

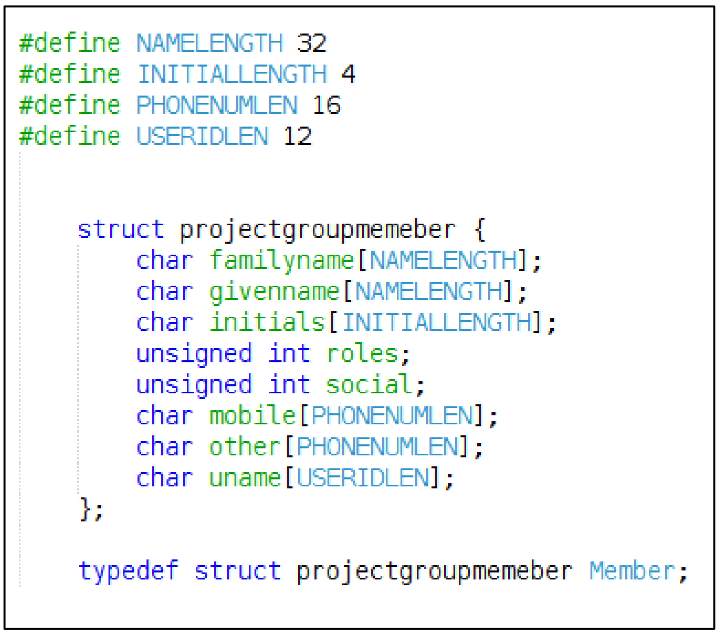
Create a new NetBeans project:



The files "base64.h" and "base64.c" should be downloaded from the Resources section of the CSCI131 web site (create files base64.h and base64.c in NetBeans and paste in the downloaded code). The header file ProjectGroup.h simply contains a struct declaration. All the application specific code is in main.c

ProjectGroup.h

This contains the struct declaration:

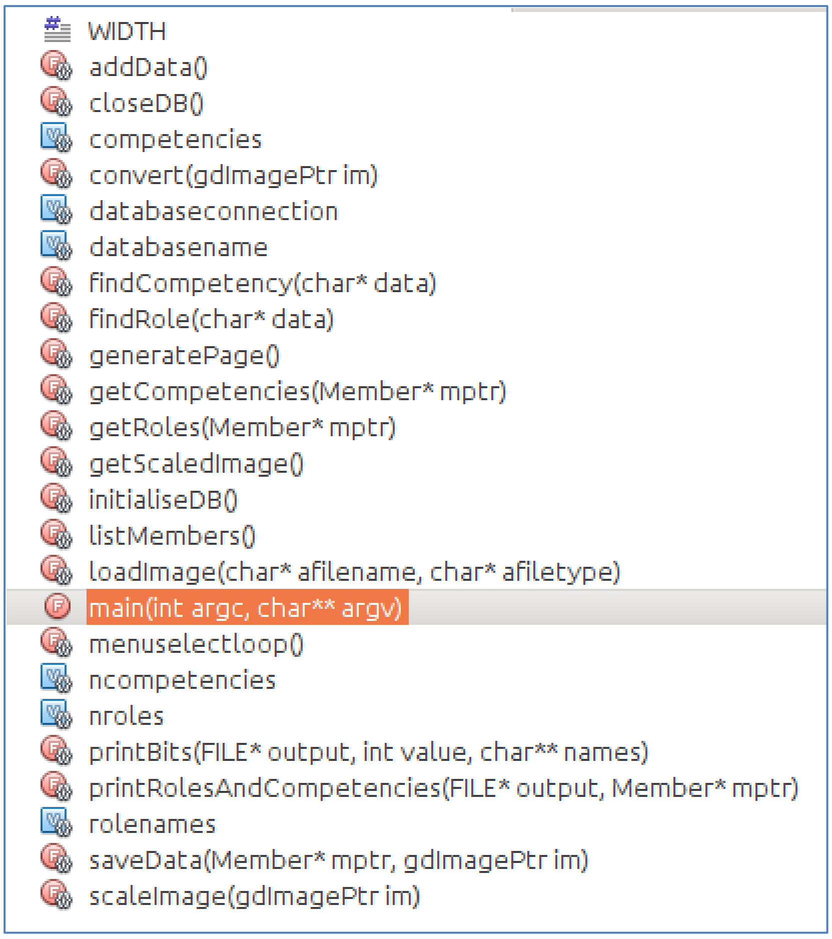


```
#define NAMELENGTH 32
#define INITIALLENGTH 4
#define PHONENUMLEN 16
#define USERIDLEN 12


    struct projectgroupmemeber {
        char familyname[NAMELENGTH];
        char givenname[NAMELENGTH];
        char initials[INITIALLENGTH];
        unsigned int roles;
        unsigned int social;
        char mobile[PHONENUMLEN];
        char other[PHONENUMLEN];
        char uname[USERIDLEN];
    };

    typedef struct projectgroupmemeber Member;
```

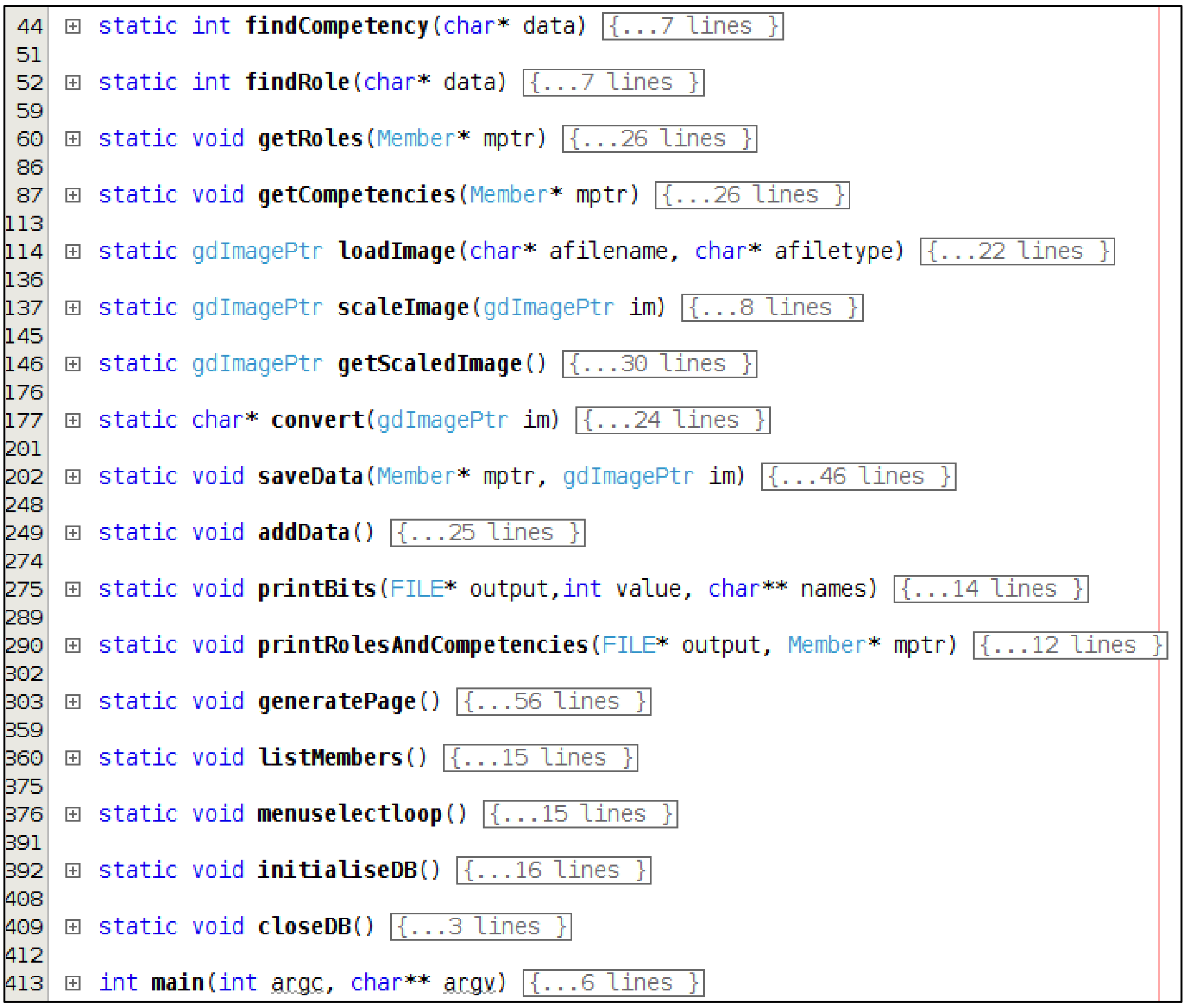
The unsigned int fields "roles", and "social" will be used as bit-maps. Each project group member will be expected to fill several roles, e.g. "Documenter", "Toolsmith", and will profess a number of social competencies, e.g. "Coordinator", "Finisher". Bits set in the unsigned int fields will identify the assigned roles and competencies.

The "uname" field is for the member's unix-id (email-id). This value will also be used as the key for the dbm key-value database.

main.c



```
WIDTH
addData()
closeDB()
competencies
convert(gdImagePtr im)
databaseconnection
databasename
findCompetency(char* data)
findRole(char* data)
generatePage()
getCompetencies(Member* mptr)
getRoles(Member* mptr)
getScaledImage()
initialiseDB()
listMembers()
loadImage(char* afilename, char* afiletype)
main(int argc, char** argv)
menuselectloop()
ncompetencies
nroles
printBits(FILE* output, int value, char** names)
printRolesAndCompetencies(FILE* output, Member* mptr)
rolenames
saveData(Member* mptr, gdImagePtr im)
scaleImage(gdImagePtr im)
```



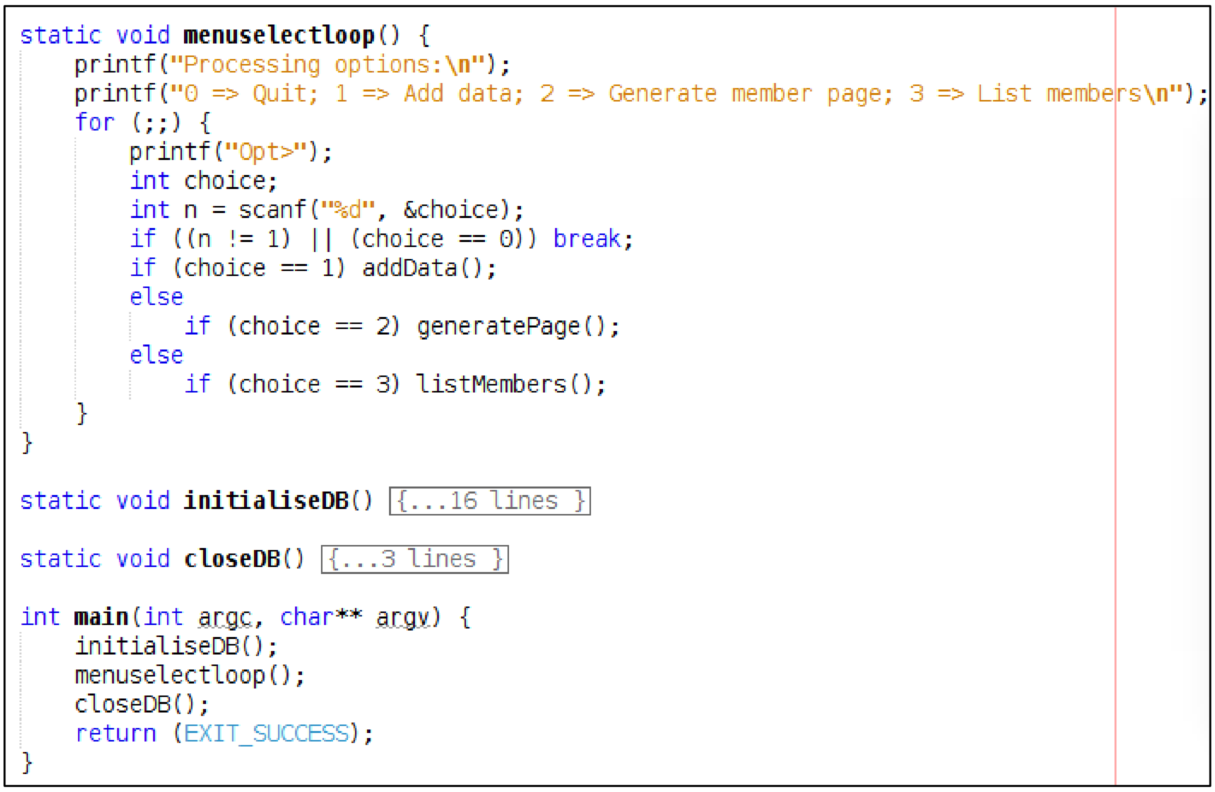
```
static int findCompetency(char* data) {...7 lines }

static int findRole(char* data) {...7 lines }

static void getRoles(Member* mptr) {...26 lines }

static void getCompetencies(Member* mptr) {...26 lines }

static gdImagePtr loadImage(char* afilename, char* afiletype) {...22 lines }

static gdImagePtr scaleImage(gdImagePtr im) {...8 lines }

static gdImagePtr getScaledImage() {...30 lines }

static char* convert(gdImagePtr im) {...24 lines }

static void saveData(Member* mptr, gdImagePtr im) {...46 lines }

static void addData() {...25 lines }

static void printBits(FILE* output,int value, char** names) {...14 lines }

static void printRolesAndCompetencies(FILE* output, Member* mptr) {...12 lines }

static void generatePage() {...56 lines }

static void listMembers() {...15 lines }

static void menuselectloop() {...15 lines }

static void initialiseDB() {...16 lines }

static void closeDB() {...3 lines }

int main(int argc, char** argv) {...6 lines }
```

includes

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gd.h>
#include <gdbm.h>
#include <math.h>
#include <ctype.h>
#include "base64.h"
#include "ProjectGroup.h"
```

All the libraries used are in /usr/include; so in this example there is no need to provide additional “build meta-data” telling the compiler where to find the header files. (Of course, the linker will have to be told to link with the extra libraries; this will be illustrated later.)

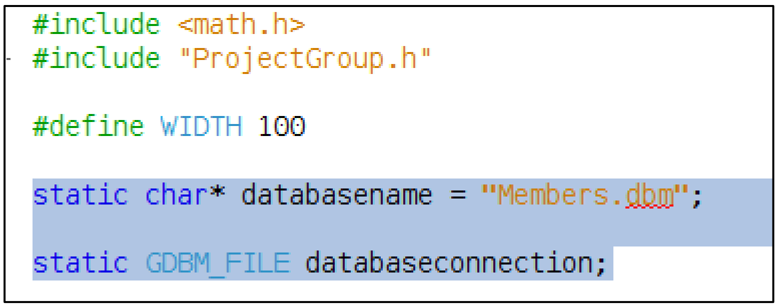
Main and menuselectloop



```
static void menuselectloop() {
    printf("Processing options:\n");
    printf("0 => Quit; 1 => Add data; 2 => Generate member page; 3 => List members\n");
    for (;;) {
        printf("Opt>");
        int choice;
        int n = scanf("%d", &choice);
        if ((n != 1) || (choice == 0)) break;
        if (choice == 1) addData();
        else
            if (choice == 2) generatePage();
        else
            if (choice == 3) listMembers();
    }
}

static void initialiseDB() {...16 lines }

static void closeDB() {...3 lines }

int main(int argc, char** argv) {
    initialiseDB();
    menuselectloop();
    closeDB();
    return (EXIT_SUCCESS);
}
```

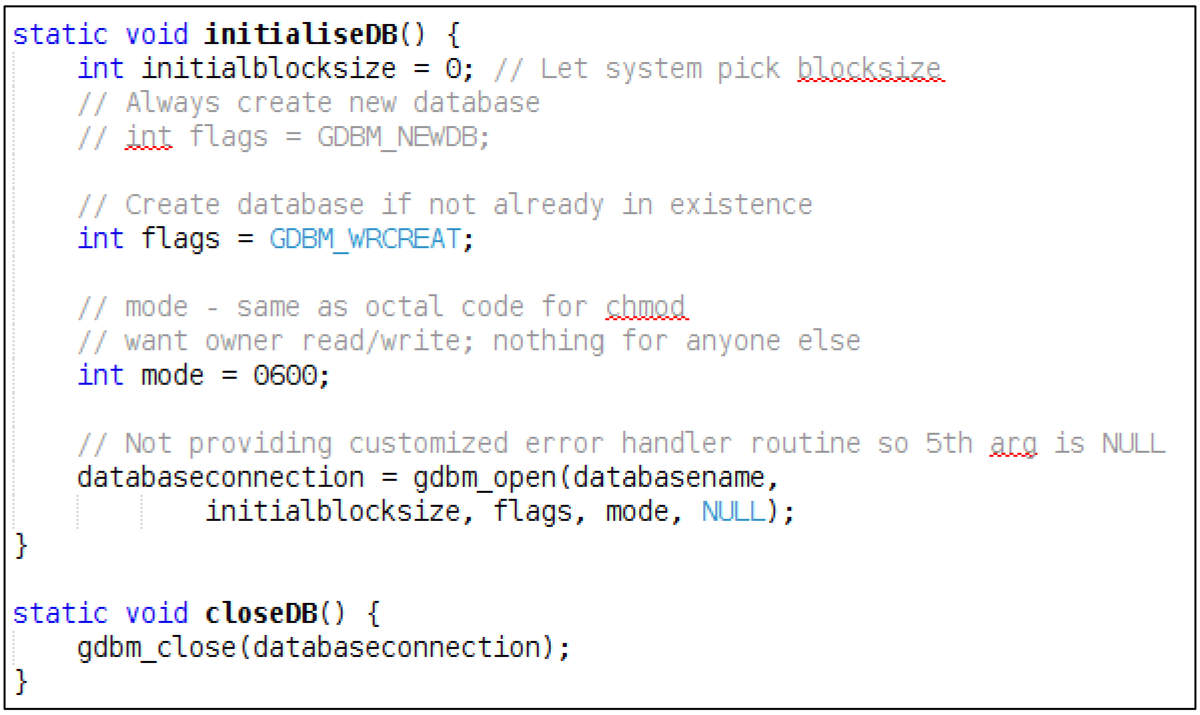
database

A dbm database is simply a file in your directory. You give it some filename (there are no restrictions on the name, there is no requirement for a specific suffix like ‘dbm’). When you “open” a connection you get back a pointer type GDBM_FILE (it is a typedef type). Typically, this pointer is used as quasi-global variable accessed in all the functions of your application.



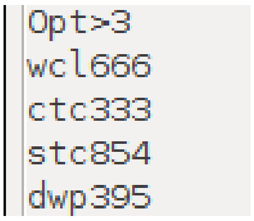
```
#include <math.h>
#include "ProjectGroup.h"

#define WIDTH 100

static char* databasename = "Members.dbm";

static GDBM_FILE databaseconnection;
```

There are several parameters for the gdbm_open() function. The code provided illustrates standard use; things get a bit more complex if you need to have a gdbm database that can be accessed concurrently by different process.



```
static void initialiseDB() {
    int initialblocksize = 0; // Let system pick blocksize
    // Always create new database
    // int flags = GDBM_NEWDB;

    // Create database if not already in existence
    int flags = GDBM_WRCREAT;

    // mode - same as octal code for chmod
    // want owner read/write; nothing for anyone else
    int mode = 0600;

    // Not providing customized error handler routine so 5th arg is NULL
    databaseconnection = gdbm_open(databasename,
            initialblocksize, flags, mode, NULL);
}

static void closeDB() {
    gdbm_close(databaseconnection);
}
```

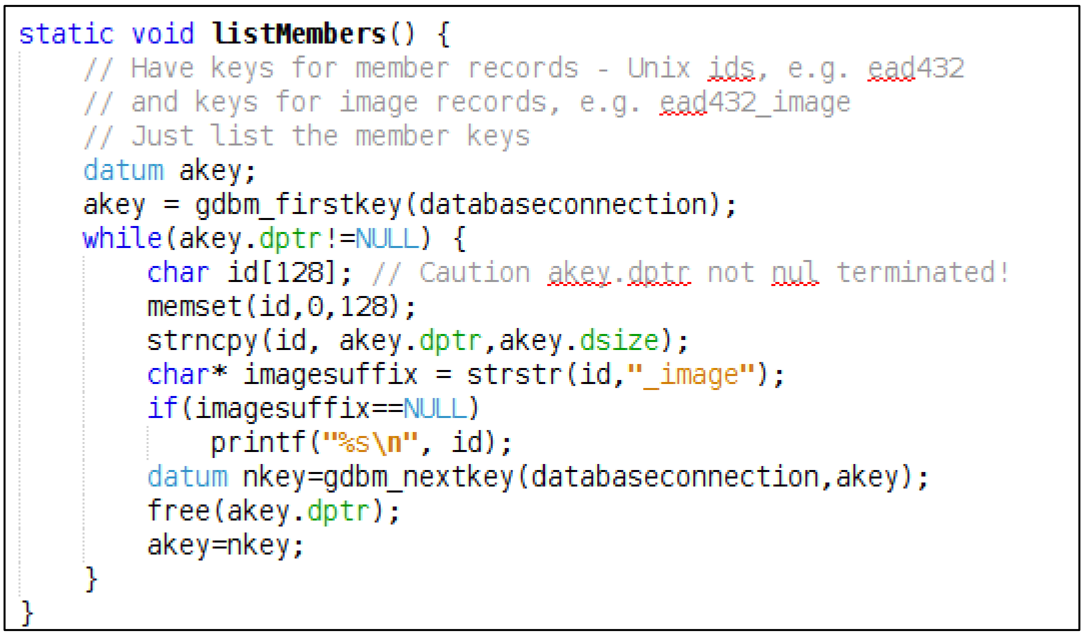
listmembers



```
Opt>3
wcl666
ctc333
stc854
dwp395
```

You should start by implementing the code to *create a new member record, and the code to list the members.* Listing is simpler to explain, so here is that code first.

Dbm libraries will offer functions that provide access to the keys of all records currently in the database. With gdbm, you have the gdbm_firstkey and gdm_nextkey functions. These can be used in a function that iterates through all keys, such as the following:



```
static void listMembers() {
    // Have keys for member records - Unix ids, e.g. ead432
    // and keys for image records, e.g. ead432_image
    // Just list the member keys
    datum akey;
    akey = gdbm_firstkey(databaseconnection);
    while(akey.dptr!=NULL) {
        char id[128]; // Caution akey.dptr not nul terminated!
        memset(id,0,128);
        strncpy(id, akey.dptr,akey.dsize);
        char* imagesuffix = strstr(id,"_image");
        if(imagesuffix==NULL)
            printf("%s\n", id);
        datum nkey=gdbm_nextkey(databaseconnection,akey);
        free(akey.dptr);
        akey=nkey;
    }
}
```

Struct datum is defined by the gdbm library. The following description is from http://www.vivtek.com/gdbm/api.html

datum

The datum is a simple structure:

```
typedef struct {
   char *dptr;
   int  dsize;
} datum;
```

It's used to pass keys and data back and forth from all functions in gdbm. Note that this means you're not dependent on the null terminator, but for ease of use, many programs still include the null terminator in keys and data. If your program must interact with others, be sure you understand whether the others are including those terminators or not.

Each key gets returned by filling in the values in the datum akey variable. Gdbm treats the key, which in our case the characters of a unix id, as a block of bytes. It will allocate a block of bytes using malloc, and copy the characters into this block – but does not add a null terminator. The size of the block is set in akey.dsize. The code here has to be careful to create a null terminated string before printing the value of the key.

Since the loop invokes malloc each time, it must also invoke free.

Why "char*"? Dbm really wants "**pointer to bytes**".

There is no "byte" type in C. So, dbm uses "char*". Many other libraries also require "pointer to bytes". Some library authors prefer to use "unsigned char*" as a way of flagging that the data really are not part of a string.

So you get some libraries where all the function signatures all use "char*", and other libraries where the signatures have "unsigned char*". Then you have to write code using both libraries – so you pick one style and pass the pointers around.

If the compiler is set at its most pedantic warning style, it will grumble about non-matching data types. But there shouldn't be any real problems resulting from mixing "char*" and "unsigned char*".

Creating records

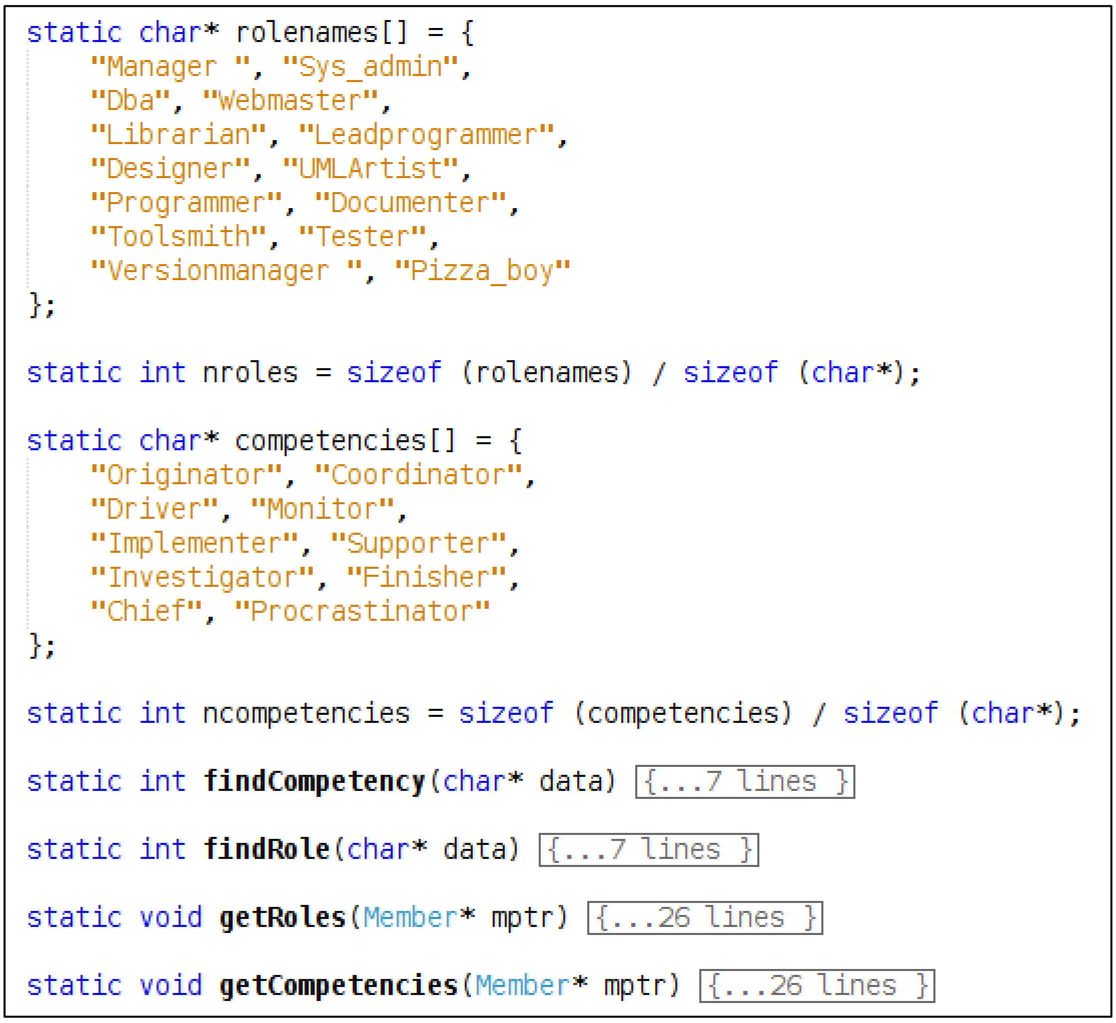
```
static int findCompetency(char* data) {...7 lines }

static int findRole(char* data) {...7 lines }

static void getRoles(Member* mptr) {...26 lines }

static void getCompetencies(Member* mptr) {...26 lines }

static gdImagePtr loadImage(char* afilename, char* afiletype) {...22 lines }

static gdImagePtr scaleImage(gdImagePtr im) {...8 lines }

static gdImagePtr getScaledImage() {...30 lines }

static char* convert(gdImagePtr im) {...24 lines }

static void saveData(Member* mptr, gdImagePtr im) {...46 lines }

static void addData() {...25 lines }
```

The main function in this group is addData. This function will prompt for the names of the additional member, then invoke auxiliary functions to get "roles" and "competencies", and to load any image data.

addData

```
static void addData() {
    Member amember;
    gdImagePtr im;
    memset(&amember, 0, sizeof (Member));
    printf("Enter member details:\n");
    // Careless - not checking for name length limits
    printf("Family name   : ");
    scanf("%s", amember.familyname);
    printf("Given name    : ");
    scanf("%s", amember.givenname);
    printf("Initials      : ");
    scanf("%s", amember.initials);
    printf("Mobile #      : ");
    scanf("%s", amember.mobile);
    printf("Other phone # : ");
    scanf("%s", amember.other);
    printf("Unix id       : ");
    scanf("%s", amember.uname);
    getRoles(&amember);
    getCompetencies(&amember);
    im = getScaledImage();
    saveData(&amember, im);
    if(im!=NULL)
        gdFree(im);
}
```

Roles and competencies



```
static char* rolenames[] = {
    "Manager ", "Sys_admin",
    "Dba", "Webmaster",
    "Librarian", "Leadprogrammer",
    "Designer", "UMLArtist",
    "Programmer", "Documenter",
    "Toolsmith", "Tester",
    "Versionmanager ", "Pizza_boy"
};

static int nroles = sizeof (rolenames) / sizeof (char*);

static char* competencies[] = {
    "Originator", "Coordinator",
    "Driver", "Monitor",
    "Implementer", "Supporter",
    "Investigator", "Finisher",
    "Chief", "Procrastinator"
};

static int ncompetencies = sizeof (competencies) / sizeof (char*);

static int findCompetency(char* data) {...7 lines }

static int findRole(char* data) {...7 lines }

static void getRoles(Member* mptr) {...26 lines }

static void getCompetencies(Member* mptr) {...26 lines }
```

The code for handling roles and competencies is similar. Only the “role” code is shown here.

The code involves a loop that repeatedly prompts for another role name. The loop terminates if the user enter “-“.   If the user enters “?”, the known roles are listed.

The value (string) entered by the user is checked against the set of known roles. If the user mistypes something, or enters a made up role, the entry will be ignored.

Otherwise, the role number is identified and used to set a bit in the bit map “unsigned int roles”.

The code for “competencies” differs only in prompts and the array data used to check inputs.

```
static int findRole(char* data) {
    int i;
    for (i = 0; i < nroles; i++) {
        if (0 == strcmp(data, rolenames[i])) return i;
    }
    return -1;
}

static void getRoles(Member* mptr) {
    printf("Enter roles");

    for (;;) {
        char instr[128];
        printf(">");
        scanf("%s", instr);
        if (0 == strlen(instr)) continue;
        if (instr[0] == '-') break;
        if (instr[0] == '?') {
            int j;
            printf("Choices are:");
            for (j = 0; j < nroles; j++)
                printf("\t%s\n", rolenames[j]);
            continue;
        }
        int ndx = findRole(instr);
        if (ndx < 0) {
            printf("Unrecognized input %s\n", instr);
            printf("Enter role name, or ? to list names, or - to terminate\n");
            continue;
        }
        int bit = 1 << ndx;
        mptr->roles |= bit;
    }
}
```

Image data

```
static gdImagePtr loadImage(char* afilename, char* afiletype) {...22 lines }

static gdImagePtr scaleImage(gdImagePtr im) {...8 lines }

static gdImagePtr getScaledImage() {...30 lines }
```

If a photo of a group member is available, the program will load the image data from a file, and then scale the image to a standard size.

The getScaledImage function allows the user to indicate whether an image is to be loaded and, if appropriate, provide a filename for that image. The file must be in one of the standard image formats – gif, jpg, png; the format being identified by a file suffix.

```
static gdImagePtr getScaledImage() {

    char input[128];
    printf("Is an image available (y|n) : ");
    scanf("%s", input);
    if (input[0] != 'y')
        return NULL;


    char afiletype[6];
    char afilename[128];

    printf("Name of image file : ");
    scanf("%s", afilename);
    char* typesuffix = strrchr(afilename, '.');
    if (typesuffix == NULL) {
        printf("No filetype suffix?\n");
        printf("Ignoring request of image\n");
        return NULL;
    }
    strcpy(afiletype, typesuffix + 1);
    char *p;
    for (p = afiletype; *p != '\0'; p++)
        *p = (char) tolower(*p);
    gdImagePtr im = loadImage(afilename, afiletype);
    gdImagePtr ims = scaleImage(im);
    gdImageDestroy(im);

    return ims;
}
```

Loading an image is straightforward using the functions from the gd graphics library:

```c
static gdImagePtr loadImage(char* afilename, char* afiletype) {
    FILE *afile;
    afile = fopen(afilename, "r");
    if (afile == NULL) {
        perror("File failed to open");
        exit(1);
    }
    gdImagePtr im = NULL;
    if (0 == strcmp(afiletype, "gif")) {
        im = gdImageCreateFromGif(afile);
    } else
        if (0 == strcmp(afiletype, "png")) {
        im = gdImageCreateFromPng(afile);

    } else
        if ((0 == strcmp(afiletype, "jpg")) || (0 == strcmp(afiletype, "jpeg"))) {
        im = gdImageCreateFromJpeg(afile);
    } else {
        printf("Sorry - cannot handle image type %s\n", afiletype);
    }
    fclose(afile);
    return im;
}
```

An image will be a complex data structure created in the heap; there is a primary “image” structure (this is identified by the pointer returned). There may be other structures such as colour palettes that are also created. Code should always free the space for images that are no longer required by using the gdImageDestroy() function; other structures created by gd should be freed using gdFree() (don’t simply call free()).

The gd library provides several functions to scale images. They differ in how details of the new image are calculated; these differences are only important in more advanced use. A simple scaling function is as follows:

```c
static gdImagePtr scaleImage(gdImagePtr im) {
    int w = gdImageSX(im);
    int h = gdImageSY(im);

    int nh = (int) ceil(WIDTH * h / w);
    gdImagePtr ims = gdImageScale(im, WIDTH, nh);
    return ims;
}
```

The functions gdImageSX and gdImageSY return the width and height of an image.

Saving data

```
static char* convert(gdImagePtr im) {...24 lines }

static void saveData(Member* mptr, gdImagePtr im) {...46 lines }
```

Once data for a new member have been entered, they must be saved to the gdbm database. The same database (dbm file) is going to be used to store both member records, and image data. Member records will be stored with keys that are the member's unix-ids, e.g. ead432. Images will be stored with keys like ead432**_image**. (Dbm systems don't care about the data – they are just a block of bytes. So there are no problems with storing different types of data record in the one file.)

The image data could be saved as they exist in memory as the scaled image record (i.e. a pixel array). But when they get used for generating HTML pages, they have to be as a base-64 encoded version of a standard image format such as jpg. So here the program does the conversion to jpeg encoding and base-64 conversion prior to storing the image.

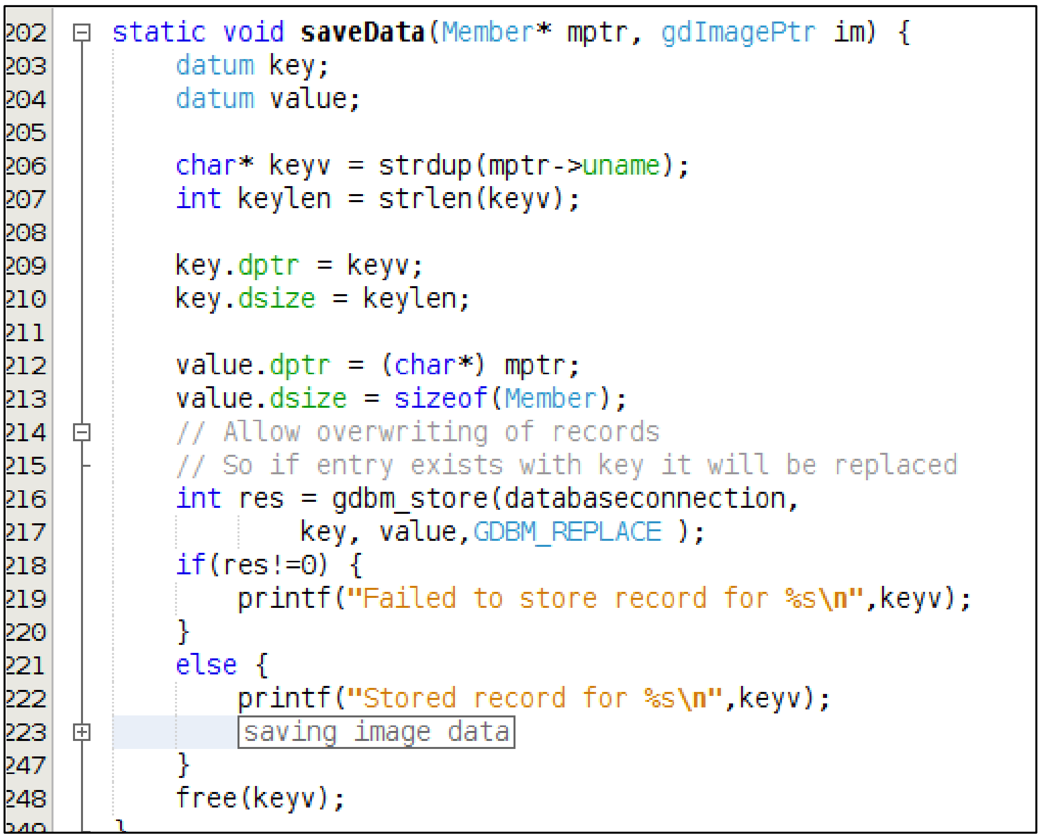
convert

The conversion function first generates an array of bytes with the .jpg format representation of the image. (This uses gdImageJpegPtr a listed but undocumented function in the gd library). This array of bytes is then encoded in base-64.

```
static char* convert(gdImagePtr im) {
    // Convert the image to jpeg, default quality, getting
    // back pointer to allocated bytes and length
    int size;
    void* jpegbytes = gdImageJpegPtr(im, &size, 75);

    size_t encodedsize = 0;
    unsigned char* source = (unsigned char*) jpegbytes;
    char* b64codes = NULL;
    // Invoke base64 encoder first time just to get length of
    // base 64 string
    base64_encode(b64codes, &encodedsize, source, size);
    // Allocate space
    b64codes = malloc(encodedsize + 1);
    // Convert
    int res = base64_encode(b64codes, &encodedsize, source, size);

    gdFree(jpegbytes);

    if (res != 0) {
        printf("Failed to base 64 encode data\n");
        if (b64codes != NULL)
            free(b64codes);
        return NULL;
    }
    return b64codes;
}
```

saveData

The saveData function is a bit long, so first we have the main code for saving the member record (with the code to save the image hidden in an "editor-fold").



```
static void saveData(Member* mptr, gdImagePtr im) {
    datum key;
    datum value;

    char* keyv = strdup(mptr->uname);
    int keylen = strlen(keyv);

    key.dptr = keyv;
    key.dsize = keylen;

    value.dptr = (char*) mptr;
    value.dsize = sizeof(Member);
    // Allow overwriting of records
    // So if entry exists with key it will be replaced
    int res = gdbm_store(databaseconnection,
            key, value,GDBM_REPLACE );
    if(res!=0) {
        printf("Failed to store record for %s\n",keyv);
    }
    else {
        printf("Stored record for %s\n",keyv);
        saving image data
    }
    free(keyv);
}
```

Two datum structures have to be used; one is for the key, the other is for the value. Each datum has a pointer to bytes member, and a length.

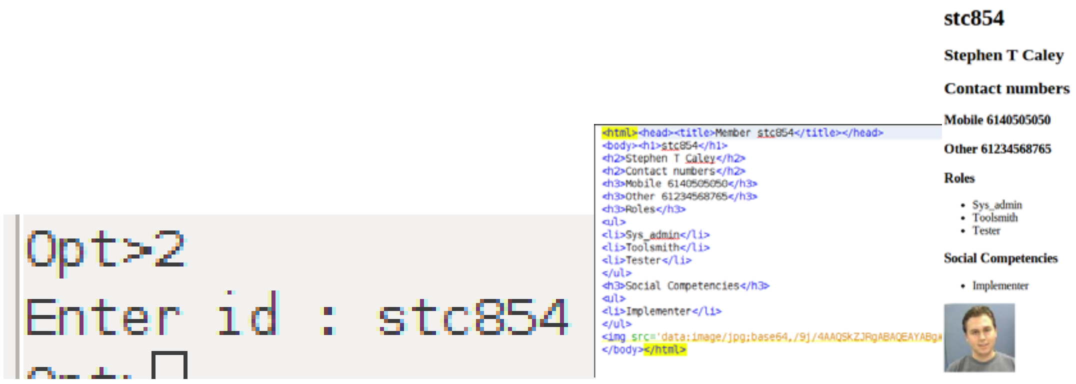
The gdbm_store function is used to write the key/value combination to file.

The code that stores a record for image data is similar:

```
    if(res!=0) {
        printf("Failed to store record for %s\n",keyv);
    }
    else {
        printf("Stored record for %s\n",keyv);
        //<editor-fold desc="saving image data">
        if(im!=NULL) {
            char* imageastr = convert(im);
            char otherkey[128];
            strcpy(otherkey,keyv);
            strcat(otherkey,"_image");
            datum imagekey;
            datum imagedata;

            imagekey.dptr = otherkey;
            imagekey.dsize = strlen(otherkey);

            imagedata.dptr = imageastr;
            imagedata.dsize = strlen(imageastr);

            int ires = gdbm_store(databaseconnection,
            imagekey, imagedata,GDBM_REPLACE );

            if(ires!=0)
                printf("\t(but failed to store image)\n");

            free(imageastr);
        }
        //</editor-fold>
    }
    free(keyv);
}
```

Output

The program has to generate HTML pages that can be viewed in a browser. This is simply a matter of retrieving the required record and then taking values from the record and slotting them into standard HTML markup text.



```
Opt>2
Enter id : stc854
```

```
<html><head><title>Member stc854</title></head>
<body><h1>stc854</h1>
<h2>Stephen T Caley</h2>
<h2>Contact numbers</h2>
<h3>Mobile 6140505050</h3>
<h3>Other 61234568765</h3>
<h3>Roles</h3>
<ul>
<li>Sys_admin</li>
<li>Toolsmith</li>
<li>Tester</li>
</ul>
<h3>Social Competencies</h3>
<ul>
<li>Implementer</li>
</ul>
<img src='data:image/jpg;base64,/9j/4AAQSkZJRgABAQEAYABg
</body></html>
```

stc854

Stephen T Caley

Contact numbers

Mobile 6140505050

Other 61234568765

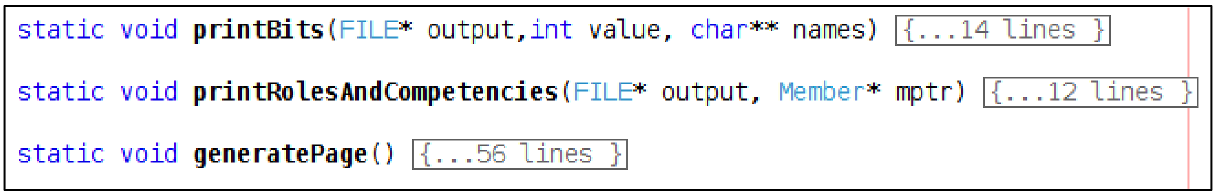
Roles

- Sys_admin
- Toolsmith
- Tester

Social Competencies

- Implementer

Output is via the generatePage function along with two auxiliary functions that create HTML lists from the data in the roles and competencies fields of a user record.



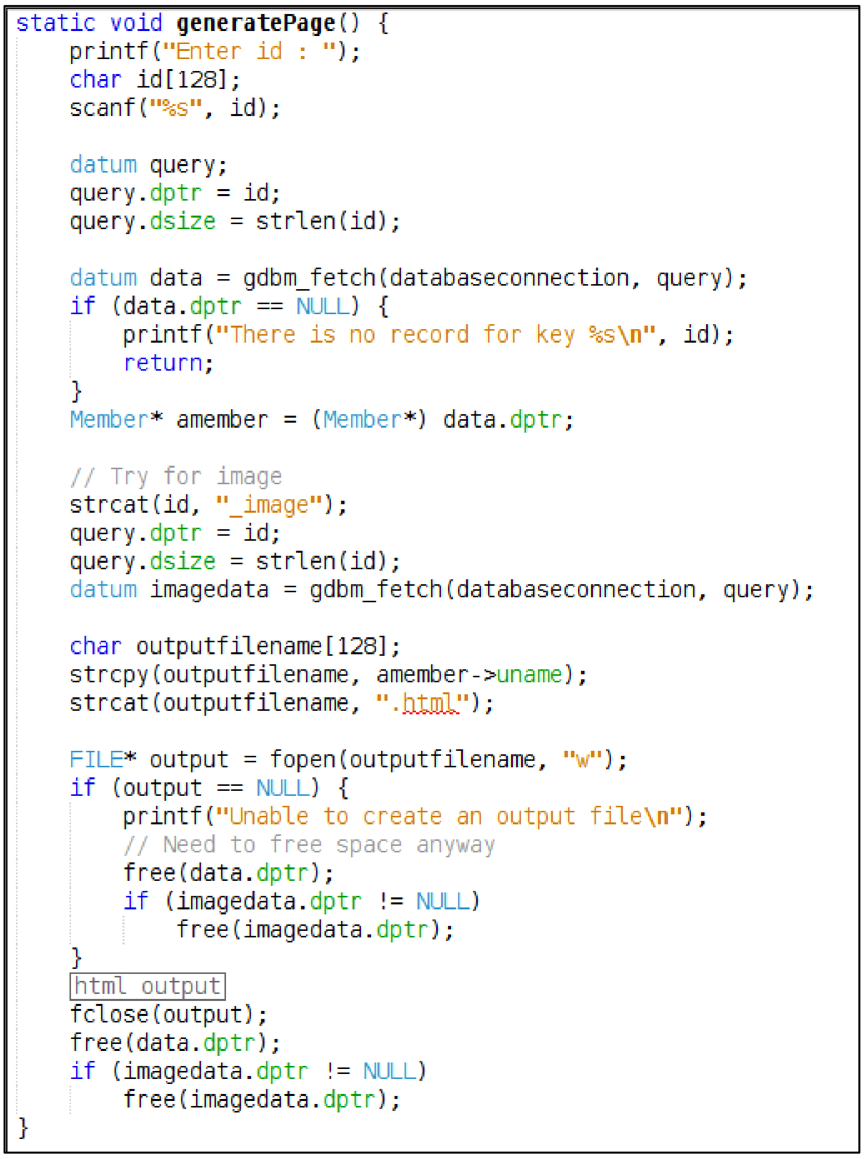
```
static void printBits(FILE* output,int value, char** names) {...14 lines }

static void printRolesAndCompetencies(FILE* output, Member* mptr) {...12 lines }

static void generatePage() {...56 lines }
```

generatePage

The generatePage function constructs a datum for the key based on the entered user-id, and uses this to retrieve a member record (returning with no output if there is no record for that key). It then tries to load an associated image record.

The output file is opened.

The HTML is generated by the code that is hidden in the editor-fold in the following view:



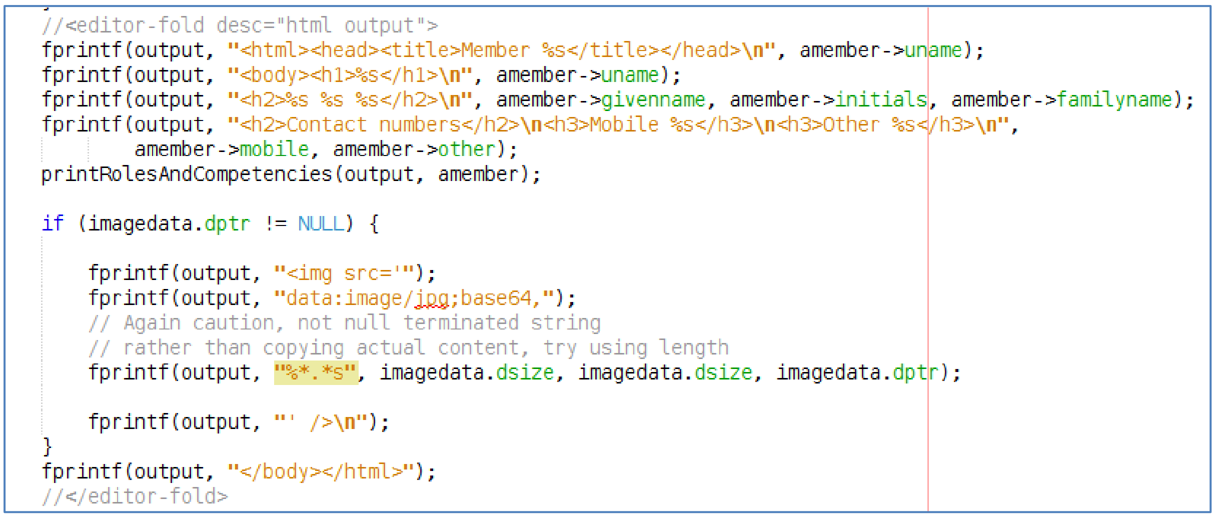
```c
static void generatePage() {
    printf("Enter id : ");
    char id[128];
    scanf("%s", id);

    datum query;
    query.dptr = id;
    query.dsize = strlen(id);

    datum data = gdbm_fetch(databaseconnection, query);
    if (data.dptr == NULL) {
        printf("There is no record for key %s\n", id);
        return;
    }
    Member* amember = (Member*) data.dptr;

    // Try for image
    strcat(id, "_image");
    query.dptr = id;
    query.dsize = strlen(id);
    datum imagedata = gdbm_fetch(databaseconnection, query);

    char outputfilename[128];
    strcpy(outputfilename, amember->uname);
    strcat(outputfilename, ".html");

    FILE* output = fopen(outputfilename, "w");
    if (output == NULL) {
        printf("Unable to create an output file\n");
        // Need to free space anyway
        free(data.dptr);
        if (imagedata.dptr != NULL)
            free(imagedata.dptr);
    }
    html output
    fclose(output);
    free(data.dptr);
    if (imagedata.dptr != NULL)
        free(imagedata.dptr);
}
```

The HTML code is largely fixed - <head> section, <body> etc. Details from the member record are slotted into <h1>…</h1> fields etc.

The roles and competencies will be <ul> lists; the code is shown later.

Commonly, an image in a HTML page is encoded as an <img …> tag with a src attribute that is the URL of the image (or sometimes is the URL of a program that will generate an image).

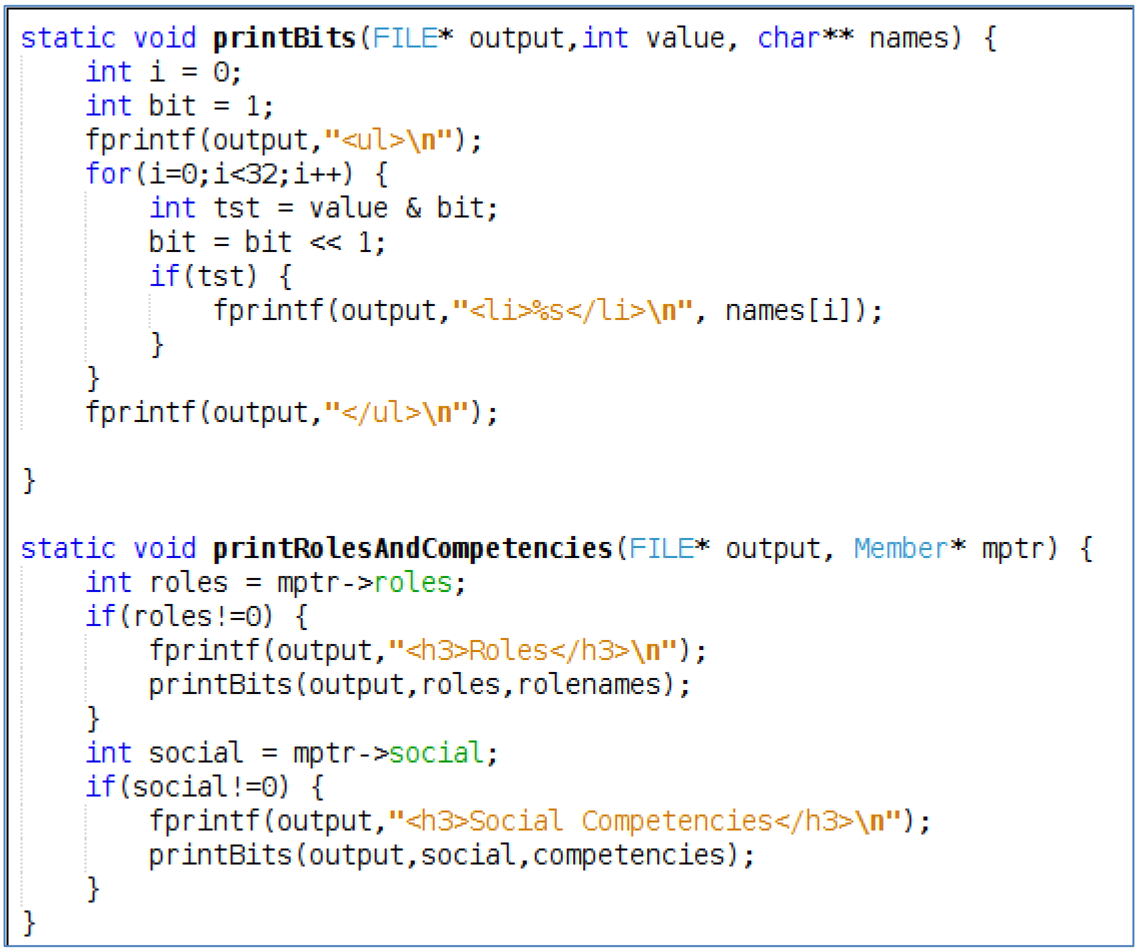
But if an image is small, it is better to have the HTML code contain the actual image data (this saves on HTTP traffic). Images can be defined as data strings that are given directly in an <img …> tag. This is done here with the base-64 encoded image data from the gdbm database. (Again, it is necessary to deal with a string that is not nul terminated; an unusual printf format string is used to select and print the correct number of characters in the base-64 string.)



```
//<editor-fold desc="html output">
fprintf(output, "<html><head><title>Member %s</title></head>\n", amember->uname);
fprintf(output, "<body><h1>%s</h1>\n", amember->uname);
fprintf(output, "<h2>%s %s %s</h2>\n", amember->givenname, amember->initials, amember->familyname);
fprintf(output, "<h2>Contact numbers</h2>\n<h3>Mobile %s</h3>\n<h3>Other %s</h3>\n",
        amember->mobile, amember->other);
printRolesAndCompetencies(output, amember);

if (imagedata.dptr != NULL) {

    fprintf(output, "<img src='");
    fprintf(output, "data:image/jpg;base64,");
    // Again caution, not null terminated string
    // rather than copying actual content, try using length
    fprintf(output, "%*.*s", imagedata.dsize, imagedata.dsize, imagedata.dptr);

    fprintf(output, "' />\n");
}
fprintf(output, "</body></html>");
//</editor-fold>
```

printBits and printRolesAndCompetencies

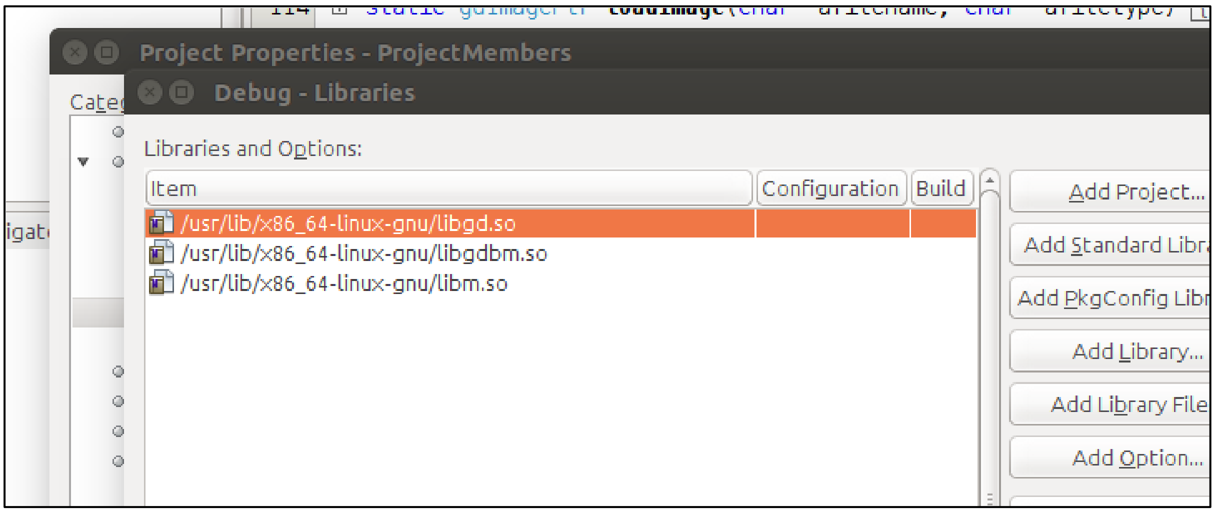
These functions are used to convert data in the bit-map integers into HTML <ul> lists.



```
static void printBits(FILE* output,int value, char** names) {
    int i = 0;
    int bit = 1;
    fprintf(output,"<ul>\n");
    for(i=0;i<32;i++) {
        int tst = value & bit;
        bit = bit << 1;
        if(tst) {
            fprintf(output,"<li>%s</li>\n", names[i]);
        }
    }
    fprintf(output,"</ul>\n");

}

static void printRolesAndCompetencies(FILE* output, Member* mptr) {
    int roles = mptr->roles;
    if(roles!=0) {
        fprintf(output,"<h3>Roles</h3>\n");
        printBits(output,roles,rolenames);
    }
    int social = mptr->social;
    if(social!=0) {
        fprintf(output,"<h3>Social Competencies</h3>\n");
        printBits(output,social,competencies);
    }
}
```

Linking

Both gdbm and gd should be in /usr/lib. But the linker must still be told to search these libraries. In your NetBeans project, you should use the Project/Properties/Linker dialog to specify the libraries ("Add Library File").


Project Properties - ProjectMembers

Debug - Libraries

Libraries and Options:

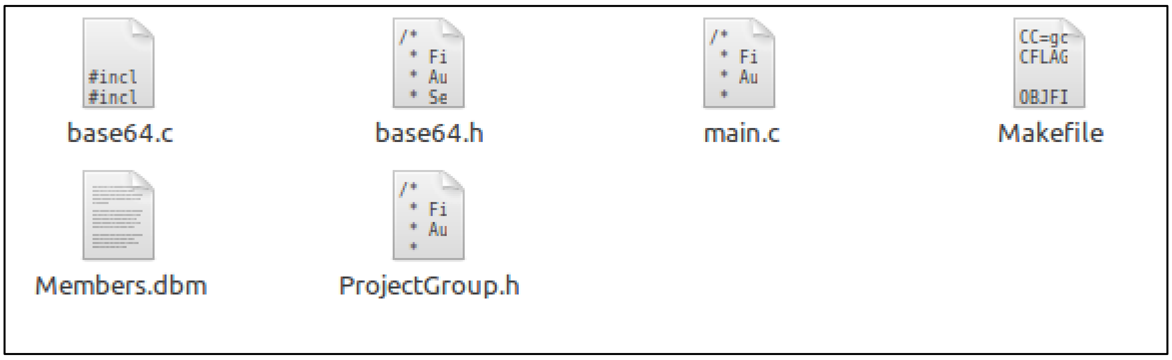
Item	Configuration	Build
/usr/lib/x86_64-linux-gnu/libgd.so		
/usr/lib/x86_64-linux-gnu/libgdbm.so		
/usr/lib/x86_64-linux-gnu/libm.so		

The libraries will be in the 64-bit subdirectory of /usr/lib.

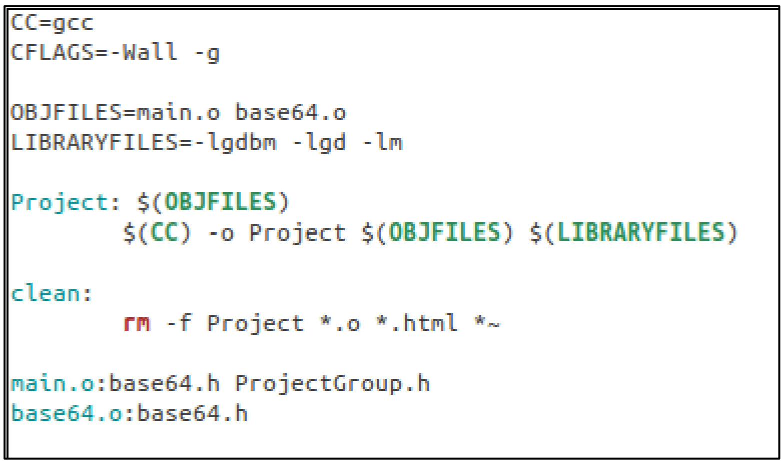
No IDE

When you have got the program to run correctly within NetBeans, you should create a copy that will be built using your own makefile.

Create a new subdirectory, and copy into this directory the files main.c, ProjectGroup.h, base64.h, base64.c along with any Members.dbm file that you have generated.



makefile



```
CC=gcc
CFLAGS=-Wall -g

OBJFILES=main.o base64.o
LIBRARYFILES=-lgdbm -lgd -lm

Project: $(OBJFILES)
	$(CC) -o Project $(OBJFILES) $(LIBRARYFILES)

clean:
	rm -f Project *.o *.html *~

main.o:base64.h ProjectGroup.h
base64.o:base64.h
```

CC=gcc

Use the gcc compiler

CFLAGS=-Wall -g

Make the compiler warn about all possible errors, and add in debugging information.

OBJFILES=main.o base64.o

The generated code files needed to build the program are main.o, and base64.o. These will be generated in accord with default rules from main.c and base64.c

LIBRARYFILES=-lgdm –lgd –lm

Build with the appropriate versions of these libraries. Here, the libraries are being specified using the conventional syntax with "-l". The linker will look for files libgdbm.so etc. (It will use configuration data that makes it check for 64-bit versions of the code.)

(NetBeans uses absolute path names for library files, but that isn't conventional for Unix programmers.)

There are two targets – Project, and clean (technically, clean is a pseudo target).

```
Project: $(OBJFILES)

		$(CC) –o Project $(OBJFILES) $(LIBRARYFILES)
```

The target Project should be built if the file “Project” does not exist or if it is older than any of the files identified in the file list OBJFILES (or files that these depend on).

Use the build toolkit specified by value of $(CC) to compile and link the code with output to “Project”.

```
clean:

		rm –f Project *.o *.html *~
```

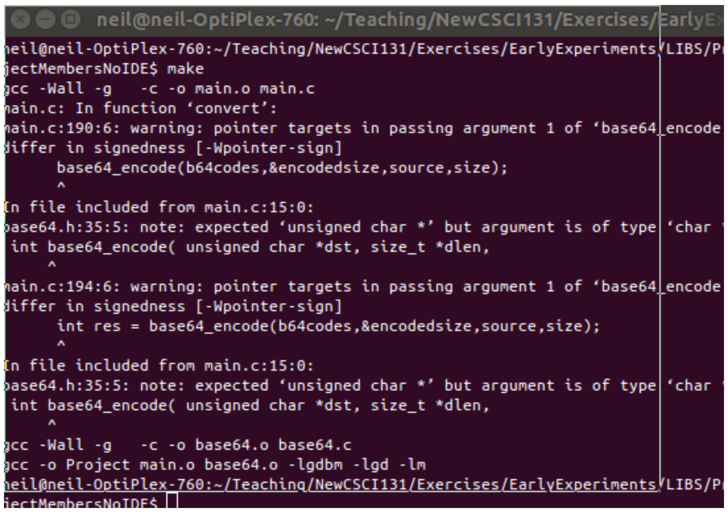
Target clean doesn’t have any dependencies. It will always run when asked for.

It removes the listed files – Project, any files with type .o, any .html files (outputs from earlier tests) and any files ending in ~ (discarded files from editors). You could also specify “.dbm” if you wanted to clear out database files.

```
main.o:base64.h Project.h

base64.o:base64.h
```

These specify extra dependencies. If either header file is changed, main.c must be recompiled. (The dependency specifies main.o not main.c. The make program cannot build main.c! It has to recreate the compiled version.)

Try building with your makefile (the compiler should grumble about the mixing of char* and unsigned char*).



```
neil@neil-OptiPlex-760: ~/Teaching/NewCSCI131/Exercises/EarlyE
neil@neil-OptiPlex-760:~/Teaching/NewCSCI131/Exercises/EarlyExperiments/LIBS/P
jectMembersNoIDE$ make
gcc -Wall -g   -c -o main.o main.c
main.c: In function ‘convert’:
main.c:190:6: warning: pointer targets in passing argument 1 of ‘base64_encode
differ in signedness [-Wpointer-sign]
      base64_encode(b64codes,&encodedsize,source,size);
      ^
In file included from main.c:15:0:
base64.h:35:5: note: expected ‘unsigned char *’ but argument is of type ‘char 
 int base64_encode( unsigned char *dst, size_t *dlen,
     ^
main.c:194:6: warning: pointer targets in passing argument 1 of ‘base64_encode
differ in signedness [-Wpointer-sign]
      int res = base64_encode(b64codes,&encodedsize,source,size);
      ^
In file included from main.c:15:0:
base64.h:35:5: note: expected ‘unsigned char *’ but argument is of type ‘char 
 int base64_encode( unsigned char *dst, size_t *dlen,
     ^
gcc -Wall -g   -c -o base64.o base64.c
gcc -o Project main.o base64.o -lgdbm -lgd -lm
neil@neil-OptiPlex-760:~/Teaching/NewCSCI131/Exercises/EarlyExperiments/LIBS/P
jectMembersNoIDE$ 
```

The program should run.

Try make again. The make program should reply that it has nothing to do as everything is up to date.

Try “touch Project.h”. That should change the date on the Project.h file. Try make again. It should re-build the application.

Clean up with “make clean”.

Task 2: Redis client library: 3 marks

This exercise uses an unusual library – theC/C++ client library for the Redis database server.

"Redis is a data structure server. It is open-source, networked, in-memory, and stores keys with optional durability. The development of Redis has been sponsored by Pivotal Software since May 2013; before that, it was sponsored by VMware. According to the monthly ranking by DB-Engines.com, Redis is the most popular key-value store. The name Redis means REmote DIctionary Server." (So says Wikipedia)

*"Redis is written in ANSI C and works in most POSIX systems like Linux, *BSD, OS X without external dependencies. Linux and OSX are the two operating systems where Redis is developed and more tested, and we recommend using Linux for deploying. Redis may work in Solaris-derived systems like SmartOS, but the support is best effort. There is no official support for Windows builds, but Microsoft develops and maintains a Win-64 port of Redis." (So says Redis's own documentation.)*

Redis is one of the "No-SQL" database systems that has emerged post 2005. Generally, these database systems provide more limited functionality than a relational database system such as MySQL, emphasising instead either much faster performance or greater scalability.

Redis keeps its data in memory (which is why it is fast), with regular backups to file. (Redis files are typically stored in /var/lib/redis.)

In the laboratory, the Redis server runs as a standard Linux service; when Linux starts, one of the processes that it creates is the Redis server. The Redis server is accessed via TCP/IP. A client process can run on the same machine (accessing its "localhost" at address 127.0.0.1) or can run on another machine on the network.

Redis has a simple command line client that can be used to practice the storage and retrieval of data. Client libraries exist for a number of programming languages allowing the Redis server to be accessed from applications. The C client code has been installed in the laboratory in /usr/unusual ; there are header files, and linkable library files. (There are several C clients; the one we use is hiredis.) (If you are trying to do the exercises on your own computer, you will have to install the Redis server and command line client, and configure Redis to run as a service. You will also need to install the hiredis C client code).

What is a "data structure server"?

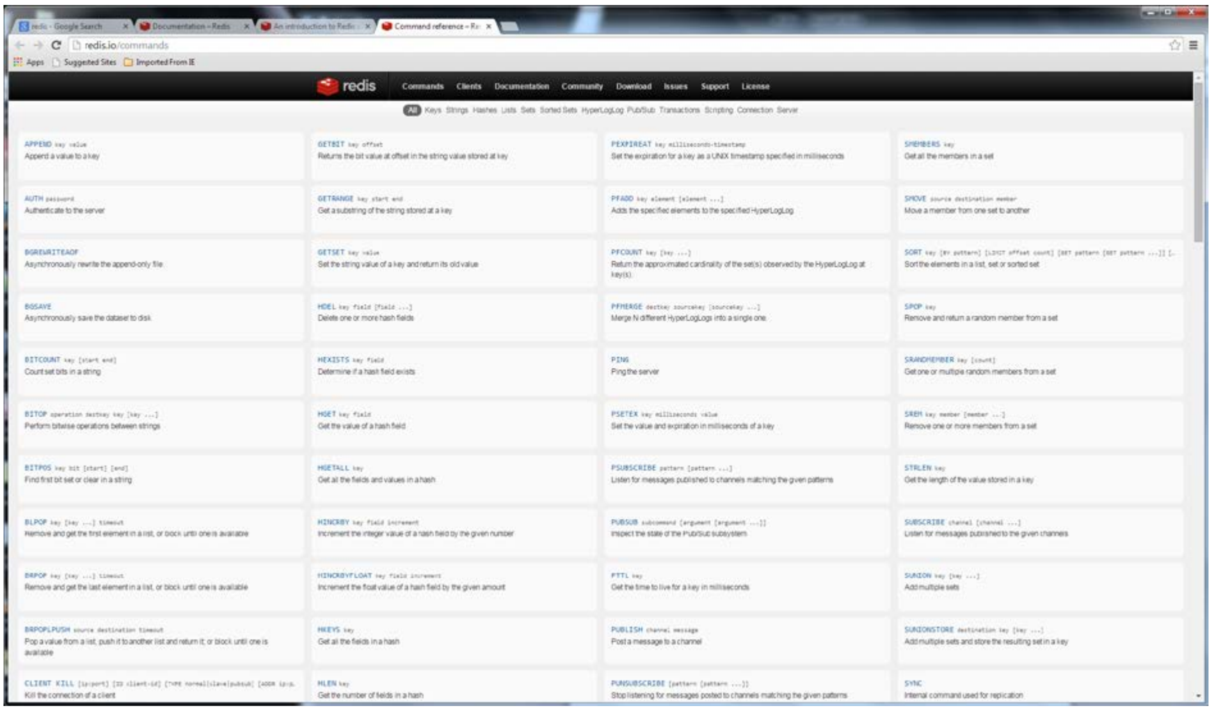
With gdbm, the data stored for a given key are simple a block of uninterpreted bytes. But with Redis the values associated with the keys can be:

- Integers (you can use these as counters that get incremented);
- Strings
- Binary data (unintepreted bytes as in gdbm)
- Lists!
  A Redis list value is a collection of strings sorted by insertion order.
- Sets!
  A Redis set is a collection of unique strings.

- Hashes!
  The Redis key will point to a structure that is itself a collection of string key/value pairs.
- (There are additional more exotic structure types).

When using an SQL database, e.g. Oracle or MySQL, from a command line client (e.g. Oracle's sqlplus client) or from within an application you compose strings with SQL commands that are sent to the database server. Work with Redis is similar. You compose strings containing the data manipulation commands that you wish the Redis server to execute. When working with the Redis command line client you simply type these commands; in an application, you compose the string with the command and use a library function to send the command to the server and return a data structure with the results.

The full set of commands, with illustrations of their use, is on the web at redis.io/commands:



APPEND key value
Append a value to a key

AUTH password
Authenticate to the server

BGREWRITEAOF
Asynchronously rewrite the append-only file

BGSAVE
Asynchronously save the dataset to disk

BITCOUNT key [start end]
Count set bits in a string

BITOP operation destkey key [key ...]
Perform bitwise operations between strings

BITPOS key bit [start] [end]
Find first bit set or clear in a string

BLPOP key [key ...] timeout
Remove and get the first element in a list, or block until one is available

BRPOP key [key ...] timeout
Remove and get the last element in a list, or block until one is available

BRPOPLPUSH source destination timeout
Pop a value from a list, push it to another list and return it; or block until one is available

CLIENT KILL [ip:port] [ID client-id] [TYPE normal|slave|pubsub] [ADDR ip:p
Kill the connection of a client

GETBIT key offset
Returns the bit value at offset in the string value stored at key

GETRANGE key start end
Get a substring of the string stored at a key

GETSET key value
Set the string value of a key and return its old value

HDEL key field [field ...]
Delete one or more hash fields

HEXISTS key field
Determine if a hash field exists

HGET key field
Get the value of a hash field

HGETALL key
Get all the fields and values in a hash

HINCRBY key field increment
Increment the integer value of a hash field by the given number

HINCRBYFLOAT key field increment
Increment the float value of a hash field by the given amount

HKEYS key
Get all the fields in a hash

HLEN key
Get the number of fields in a hash

PEXPIREAT key milliseconds-timestamp
Set the expiration for a key as a UNIX timestamp specified in milliseconds

PFADD key element [element ...]
Adds the specified elements to the specified HyperLogLog

PFCOUNT key [key ...]
Return the approximated cardinality of the set(s) observed by the HyperLogLog at key(s)

PFMERGE destkey sourcekey [sourcekey ...]
Merge N different HyperLogLogs into a single one

PING
Ping the server

PSETEX key milliseconds value
Set the value and expiration in milliseconds of a key

PSUBSCRIBE pattern [pattern ...]
Listen for messages published to channels matching the given patterns

PUBSUB subcommand [argument [argument ...]]
Inspect the state of the Pub/Sub subsystem

PTTL key
Get the time to live for a key in milliseconds

PUBLISH channel message
Post a message to a channel

PUNSUBSCRIBE [pattern [pattern ...]]
Stop listening for messages posted to channels matching the given patterns

SMEMBERS key
Get all the members in a set

SMOVE source destination member
Move a member from one set to another

SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [...
Sort the elements in a list, set or sorted set

SPOP key
Remove and return a random member from a set

SRANDMEMBER key [count]
Get one or multiple random members from a set

SREM key member [member ...]
Remove one or more members from a set

STRLEN key
Get the length of the value stored in a key

SUBSCRIBE channel [channel ...]
Listen for messages published to the given channels

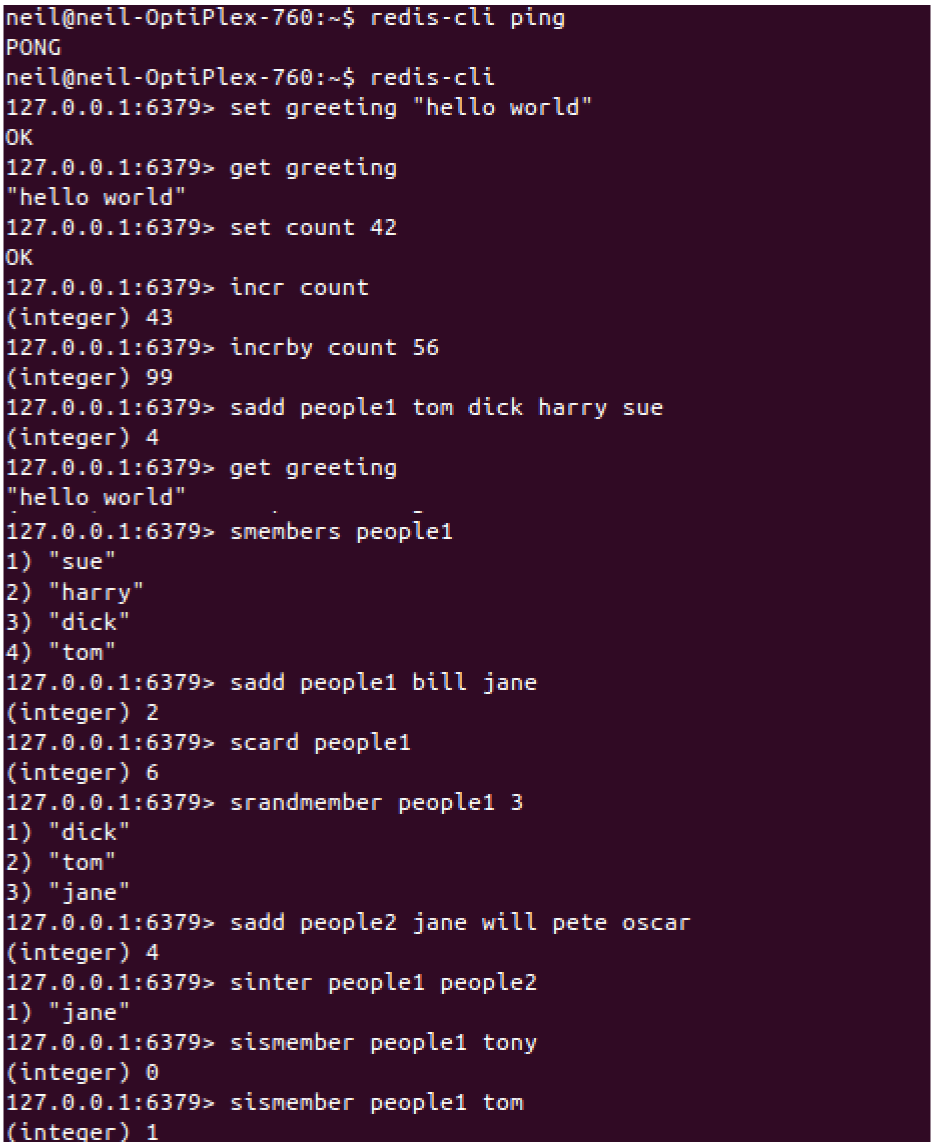
SUNION key [key ...]
Add multiple sets

SUNIONSTORE destination key [key ...]
Add multiple sets and store the resulting set in a key

SYNC
Internal command used for replication

There are several tutorials available on the web that illustrate Redis's commands.

The following recording shows use of the Linux redis-cli command line client being used to work with simple string, integer, and set data:



```
neil@neil-OptiPlex-760:~$ redis-cli ping
PONG
neil@neil-OptiPlex-760:~$ redis-cli
127.0.0.1:6379> set greeting "hello world"
OK
127.0.0.1:6379> get greeting
"hello world"
127.0.0.1:6379> set count 42
OK
127.0.0.1:6379> incr count
(integer) 43
127.0.0.1:6379> incrby count 56
(integer) 99
127.0.0.1:6379> sadd people1 tom dick harry sue
(integer) 4
127.0.0.1:6379> get greeting
"hello world"
127.0.0.1:6379> smembers people1
1) "sue"
2) "harry"
3) "dick"
4) "tom"
127.0.0.1:6379> sadd people1 bill jane
(integer) 2
127.0.0.1:6379> scard people1
(integer) 6
127.0.0.1:6379> srandmember people1 3
1) "dick"
2) "tom"
3) "jane"
127.0.0.1:6379> sadd people2 jane will pete oscar
(integer) 4
127.0.0.1:6379> sinter people1 people2
1) "jane"
127.0.0.1:6379> sismember people1 tony
(integer) 0
127.0.0.1:6379> sismember people1 tom
(integer) 1
```

The command:

```
redis-cli ping
```

serves to check whether the redis server is running. (If it's running, it will reply pong).

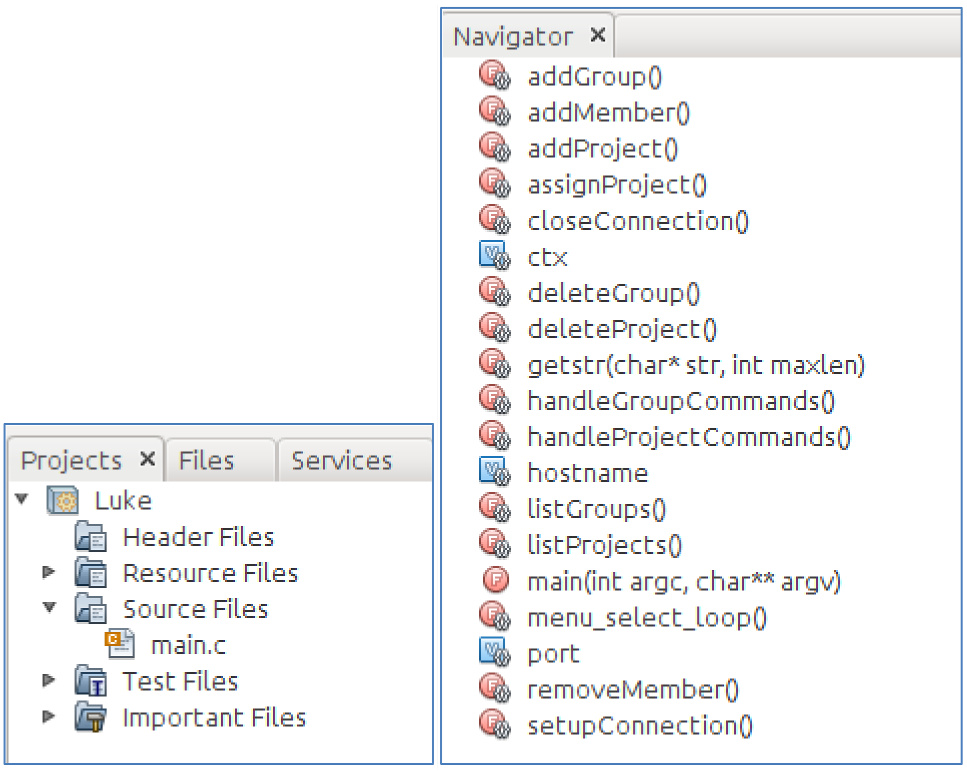
The `redis-cli` command started a client session. The key 'greeting' is assigned the string value "hello world"; the key count is given the value 42. The value of count is incremented in different ways. Then a "set" is created; the set is named 'people1', its initial members are the strings 'tom', 'dick', 'harry', and 'sue'. Different functions are used to retrieve simple data elements, sets, and hashes. There are utility functions like scard which gives the number of elements in a set; srandmember which selects some number of elements from a set chosen at random; sinter which intersects sets; and sismember that tests whether an element is currently in a set.

The command line client can be useful when debugging an application that uses Redis. You can use it to visualise the data that have been added. There is a debugging command "keys *" that lists all the keys; you can then use get, or smembers, or hgetall to look at the content of a simple element, a set, or a hash. (The keys command is not supposed to be used in application code as it is very costly at run-time; it is helpful for debugging).

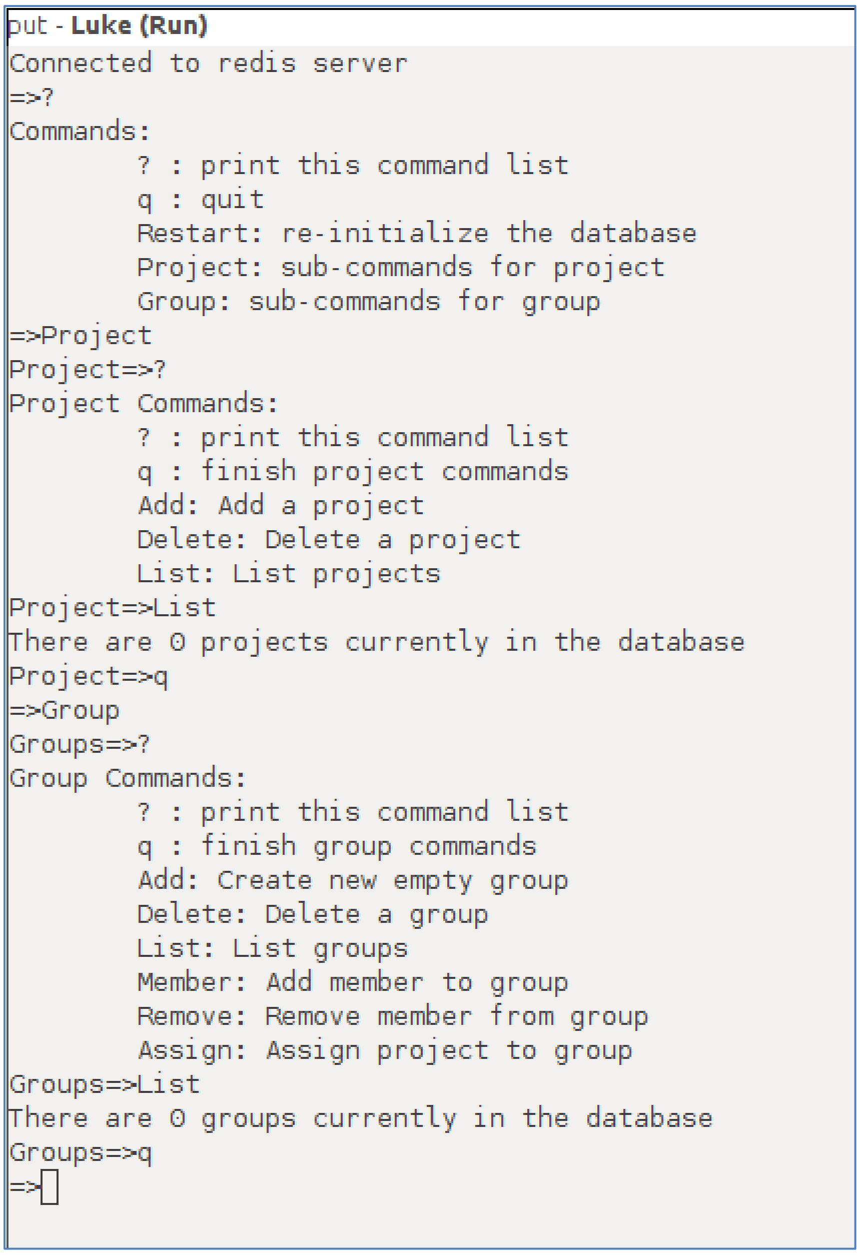
Application

The contrived application is a simple program that might be useful for the administrator of CSCI321 projects. It is to maintain records of available projects, student groups, and the allocation of projects to groups.

It's built as a NetBeans C application:



Once again, it is a "menu-select" program with commands for dealing with the projects and the student groups:



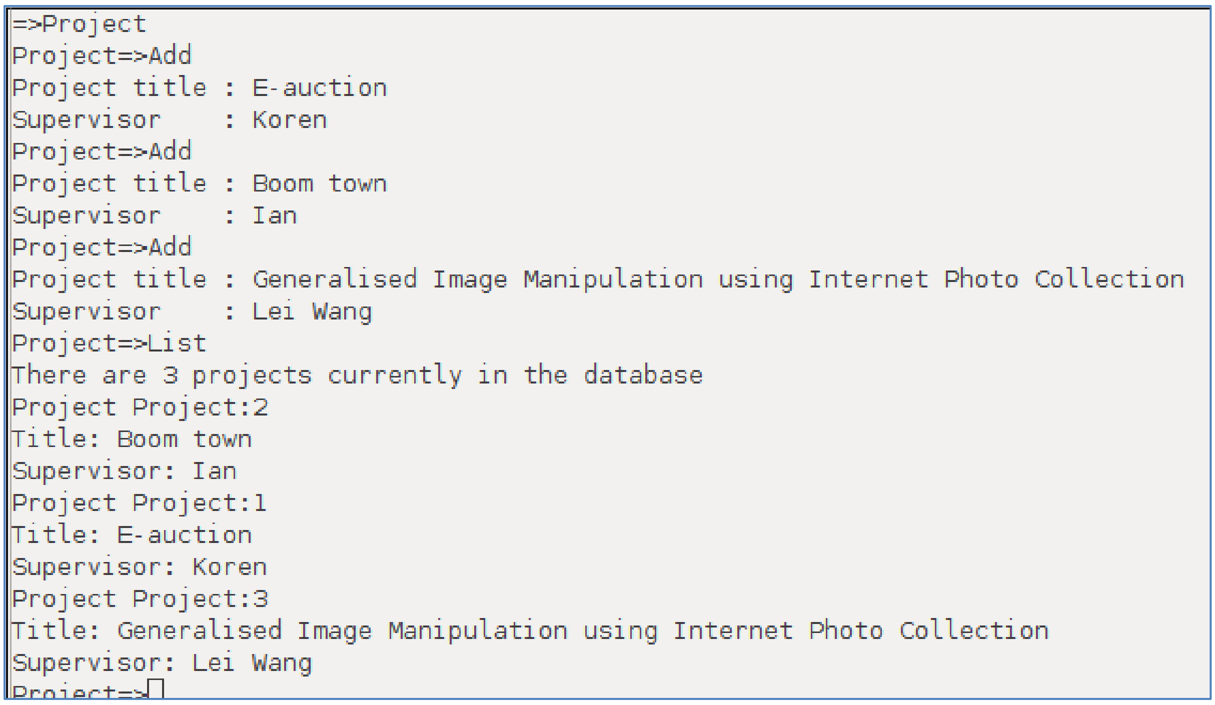
```
put - Luke (Run)
Connected to redis server
=>?
Commands:
        ? : print this command list
        q : quit
        Restart: re-initialize the database
        Project: sub-commands for project
        Group: sub-commands for group
=>Project
Project=>?
Project Commands:
        ? : print this command list
        q : finish project commands
        Add: Add a project
        Delete: Delete a project
        List: List projects
Project=>List
There are 0 projects currently in the database
Project=>q
=>Group
Groups=>?
Group Commands:
        ? : print this command list
        q : finish group commands
        Add: Create new empty group
        Delete: Delete a group
        List: List groups
        Member: Add member to group
        Remove: Remove member from group
        Assign: Assign project to group
Groups=>List
There are 0 groups currently in the database
Groups=>q
=>
```

The application maintains a collection of projects, with unique application defined identifiers. Each project has a collection of attributes – initially just title, and supervisor.

These data will be represented in the Redis database:

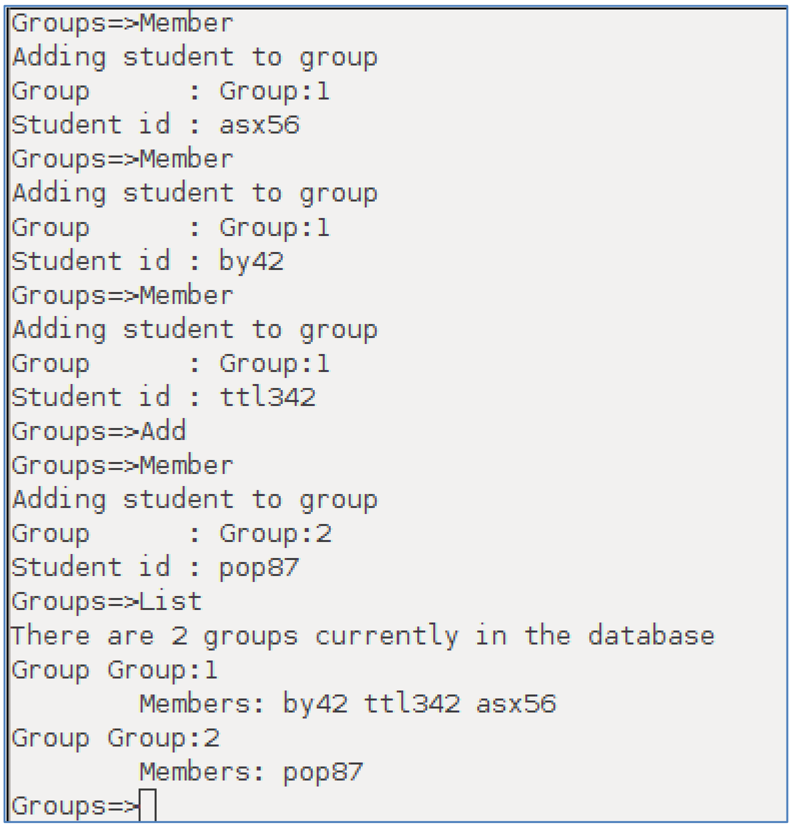
- A counter used to created distinct project identifiers.
- A set with strings that are the project names.
- For each project, there will be a 'hash'; its key is a project name, it will initially have values for two attributes (title, supervisor).

The following record shows some projects being added to the Redis database:



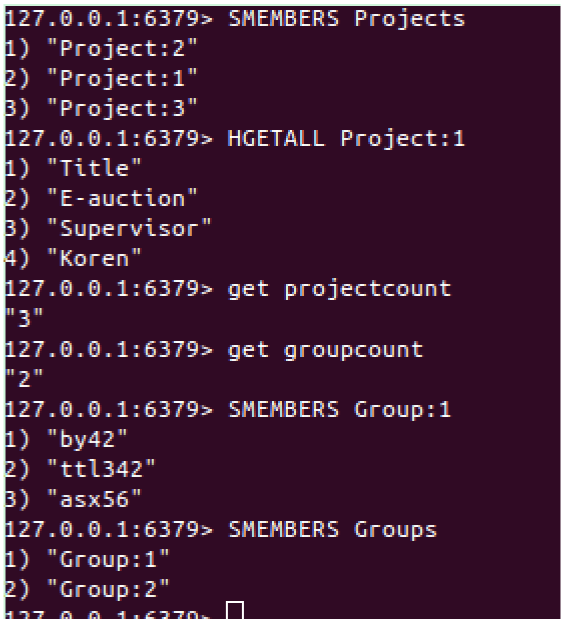
```
=>Project
Project=>Add
Project title : E-auction
Supervisor    : Koren
Project=>Add
Project title : Boom town
Supervisor    : Ian
Project=>Add
Project title : Generalised Image Manipulation using Internet Photo Collection
Supervisor    : Lei Wang
Project=>List
There are 3 projects currently in the database
Project Project:2
Title: Boom town
Supervisor: Ian
Project Project:1
Title: E-auction
Supervisor: Koren
Project Project:3
Title: Generalised Image Manipulation using Internet Photo Collection
Supervisor: Lei Wang
Project=>
```

The application will also keep track of groups and there members. The application allocates unique group identifiers; maintains a set with the names of the groups; each group will have a set of members.



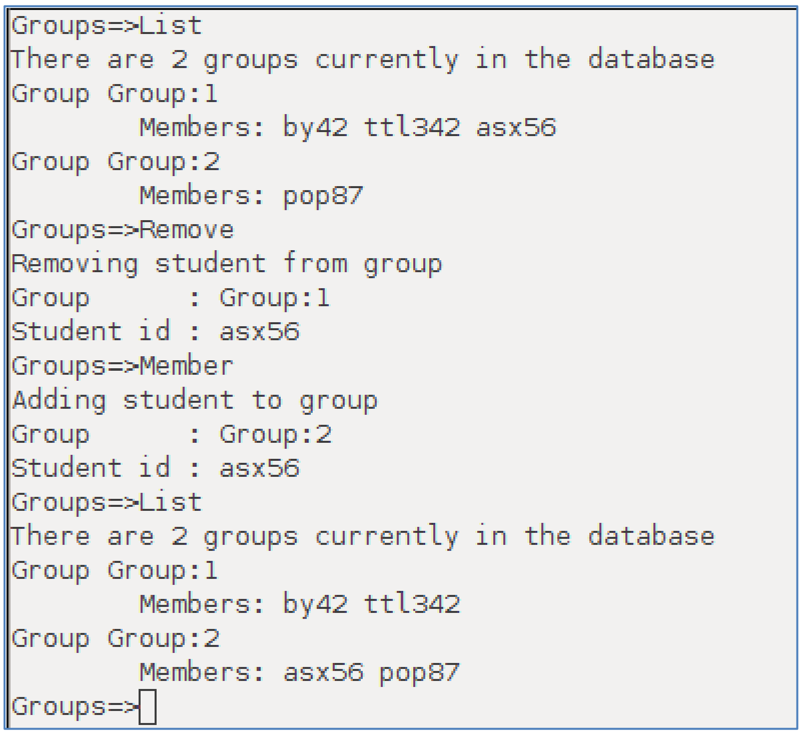
```
Groups=>Member
Adding student to group
Group      : Group:1
Student id : asx56
Groups=>Member
Adding student to group
Group      : Group:1
Student id : by42
Groups=>Member
Adding student to group
Group      : Group:1
Student id : ttl342
Groups=>Add
Groups=>Member
Adding student to group
Group      : Group:2
Student id : pop87
Groups=>List
There are 2 groups currently in the database
Group Group:1
        Members: by42 ttl342 asx56
Group Group:2
        Members: pop87
Groups=>
```

One can use the redis-cli client to check these data:

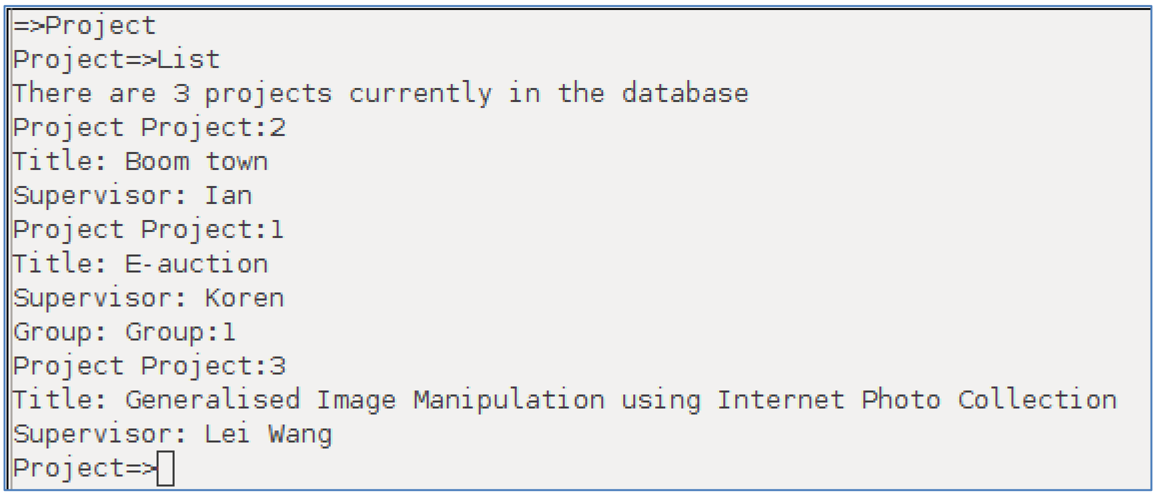


```
127.0.0.1:6379> SMEMBERS Projects
1) "Project:2"
2) "Project:1"
3) "Project:3"
127.0.0.1:6379> HGETALL Project:1
1) "Title"
2) "E-auction"
3) "Supervisor"
4) "Koren"
127.0.0.1:6379> get projectcount
"3"
127.0.0.1:6379> get groupcount
"2"
127.0.0.1:6379> SMEMBERS Group:1
1) "by42"
2) "ttl342"
3) "asx56"
127.0.0.1:6379> SMEMBERS Groups
1) "Group:1"
2) "Group:2"
```

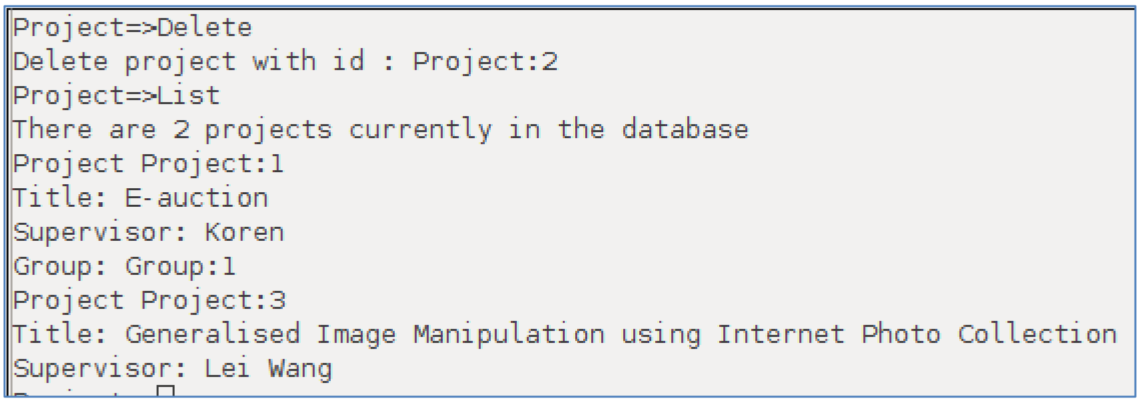
Other manipulations can be performed – allocate projects to groups, add and remove members for a group, delete projects:



```
Groups=>List
There are 2 groups currently in the database
Group Group:1
        Members: by42 ttl342 asx56
Group Group:2
        Members: pop87
Groups=>Remove
Removing student from group
Group      : Group:1
Student id : asx56
Groups=>Member
Adding student to group
Group      : Group:2
Student id : asx56
Groups=>List
There are 2 groups currently in the database
Group Group:1
        Members: by42 ttl342
Group Group:2
        Members: asx56 pop87
Groups=>
```



```
=>Project
Project=>List
There are 3 projects currently in the database
Project Project:2
Title: Boom town
Supervisor: Ian
Project Project:1
Title: E-auction
Supervisor: Koren
Group: Group:1
Project Project:3
Title: Generalised Image Manipulation using Internet Photo Collection
Supervisor: Lei Wang
Project=>
```



```
Project=>Delete
Delete project with id : Project:2
Project=>List
There are 2 projects currently in the database
Project Project:1
Title: E-auction
Supervisor: Koren
Group: Group:1
Project Project:3
Title: Generalised Image Manipulation using Internet Photo Collection
Supervisor: Lei Wang
```

The Code

It is a typical menu-select program – some initialization, the command loop, and some final tidying up before termination. The commands are at two levels – top level commands selecting whether processing groups or projects, and detailed commands for these two cases.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <hiredis.h>

// Data for the connection to a redis server on the local machine
static char* hostname = "127.0.0.1";
static redisContext *ctx;
static int port = 6379;

static void setupConnection() {...14 lines }

static void closeConnection() {...4 lines }

static void getstr(char* str, int maxlen) {...9 lines }

static void addProject() {...26 lines }

static void deleteProject() {...24 lines }

static void listProjects() {...32 lines }

static void handleProjectCommands() {...23 lines }

static void addGroup() {...14 lines }

static void deleteGroup() {...23 lines }

static void listGroups() {...32 lines }

static void addMember() {...20 lines }

static void removeMember() {...20 lines }

static void assignProject() {...40 lines }

static void handleGroupCommands() {...32 lines }

static void menu_select_loop() {...28 lines }

int main(int argc, char** argv) {
    setupConnection();
    menu_select_loop();
    closeConnection();

    return (EXIT_SUCCESS);
}
```

Functions to establish and later close a connection to the Redis server are standard:

```
// Data for the connection to a redis server on the local machine
static char* hostname = "127.0.0.1";
static redisContext *ctx;
static int port = 6379;

static void setupConnection() {
    struct timeval timeout = {1, 500000}; // 1.5 seconds
    ctx = redisConnectWithTimeout(hostname, port, timeout);
    if (ctx == NULL || ctx->err) {
        if (ctx) {
            printf("Connection error: %s\n", ctx->errstr);
            redisFree(ctx);
        } else {
            printf("Connection error: can't allocate redis context\n");
        }
        exit(1);
    }
    printf("Connected to redis server\n");
}

static void closeConnection() {
    redisFree(ctx);
    printf("Disconnected from redis server\n");
}
```

The top-level menu select loop is:

```
static void menu_select_loop() {
    for (;;) {
        printf("=>");
        char command[128];
        getstr(command, 128);
        if (command[0] == 'q') break;
        if (command[0] == '?') {
            printf("Commands:\n");
            printf("\t? : print this command list\n");
            printf("\tq : quit\n");
            printf("\tRestart: re-initialize the database\n");
            printf("\tProject: sub-commands for project\n");
            printf("\tGroup: sub-commands for group\n");
        }
        if (0 == strcmp(command, "Restart")) {
            redisReply *reply;
            // FLUSHDB, or if really want to clear everything from
            // Redis FLUSHALL
            reply = redisCommand(ctx, "FLUSHDB");
            freeReplyObject(reply);
        } else
            if (0 == strcmp(command, "Project"))
            handleProjectCommands();
        else
            if (0 == strcmp(command, "Group"))
            handleGroupCommands();
    }
}
```

Group commands

Group commands are handled by a subordinate menu-select loop:

```
static void handleGroupCommands() {
    for (;;) {
        printf("Groups=>");
        char command[128];
        getstr(command, 128);
        if (command[0] == 'q') break;
        if (command[0] == '?') {
            printf("Group Commands:\n");
            printf("\t? : print this command list\n");
            printf("\tq : finish group commands\n");
            printf("\tAdd: Create new empty group\n");
            printf("\tDelete: Delete a group\n");
            printf("\tList: List groups\n");
            printf("\tMember: Add member to group\n");
            printf("\tRemove: Remove member from group\n");
            printf("\tAssign: Assign project to group\n");

        }
        if (0 == strcmp(command, "Add")) addGroup();
        else
            if (0 == strcmp(command, "Delete")) deleteGroup();
        else
            if (0 == strcmp(command, "List")) listGroups();
        else
            if (0 == strcmp(command, "Member")) addMember();
        else
            if (0 == strcmp(command, "Remove")) removeMember();
        else
            if (0 == strcmp(command, "Assign")) assignProject();

    }
}
```

When a group is added, the counter in the Redis database must be incremented, and its updated value used to generate a new group identifier. This group identifier must be added to the set of group identifiers as maintained by Redis. This code provides a first view of the hiredis library API for C clients to Redis.

The function redisCommand() is used to create and submit commands. The redisCommand() function has been made to work a bit like fprintf. The fprintf function took a FILE* connection, a format string, and an arbitrary number of additional arguments; it worked with the low-level I/O system to send text and returned a count of elements successfully sent. The redisCommand() function takes a redisContext argument (connection to Redis), a format string, and an arbitrary number of additional arguments. It returns a data-structure whose fields hold result data. (This structure will have been allocated on the heap, along with any required subordinate structures such a strings also allocated on the heap. These data must be freed after use using the freeReplyObject function.)

In this code, the first command is simple – increment the specified 'groupcount' element in the Redis database. (If 'groupcount' didn't exist, it would be created with value 0.)

The second command is a bit more complex, it adds a member to the set "GROUPS". The value added is a string (%s in the format string). It will be the name generated for the new group.

```
static void addGroup() {
    redisReply *reply;
    reply = redisCommand(ctx, "INCR groupcount");
    int acount = (int) reply->integer;
    freeReplyObject(reply);

    char projid[128];
    memset(projid, 0, 128);
    sprintf(projid, "Group:%d", acount);

    // Add the project id to the set of all projects
    reply = redisCommand(ctx, "SADD Groups %s", projid);
    freeReplyObject(reply);
}
```

A group may have to be deleted. The code here checks whether the name as entered is a defined member of the set (this is just to catch typing errors made by the user); this code uses the SISMEMBER Redis operation. This should return an integer value, so the appropriate field is checked in the reply object. If a group with the name existed, it is first removed from the set, and then all data associated with that name are removed (if members had been allocated to the group, the Redis database would have contained a set with the groupname as key).

```
static void deleteGroup() {
    char gname[128];

    printf("Delete group with id : ");
    getstr(gname, 128);

    redisReply *reply;
    reply = redisCommand(ctx, "SISMEMBER Groups %s", gname);
    int res = (0 == reply->integer);
    freeReplyObject(reply);

    if (res) {
        printf("There was no group with that id\n");
    } else {
        // Remove id from set
        reply = redisCommand(ctx, "SREM Groups %s", gname);
        // Not checking confirmation, just assume it works
        freeReplyObject(reply);
        // Now remove the group details
        reply = redisCommand(ctx, "DEL %s", gname);
        freeReplyObject(reply);
    }
}
```

The user would need to be able to list groups with their members. Note how here we get an array of string data elements for the identifiers of the members:

```
static void listGroups() {
    redisReply *reply;
    reply = redisCommand(ctx, "SCARD Groups");

    int count = (int) reply->integer;
    freeReplyObject(reply);
    printf("There are %d groups currently in the database\n",
            count);
    char** groupnames = malloc(count * sizeof (char*));
    reply = redisCommand(ctx, "SMEMBERS Groups");
    int j;
    for (j = 0; j < reply->elements; j++) {
        groupnames[j] = strdup(reply->element[j]->str);
    }
    freeReplyObject(reply);

    for (j = 0; j < count; j++) {
        reply = redisCommand(ctx, "SMEMBERS %s", groupnames[j]);
        printf("Group %s\n\tMembers: ", groupnames[j]);
        int k;
        for (k = 0; k < reply->elements; k++) {
            printf("%s ",
                    reply->element[k]->str);
        }
        printf("\n");
        freeReplyObject(reply);
    }

    for (j = 0; j < count; j++)
        free(groupnames[j]);
    free(groupnames);
}
```

Of course, we have to be able to add members to a group (checking that the group name is known before trying to store more data):

```
static void addMember() {
    char groupname[128];
    char membername[128];
    printf("Adding student to group\n");
    printf("Group      : ");
    getstr(groupname, 128);
    printf("Student id : ");
    getstr(membername, 128);
    redisReply *reply;
    reply = redisCommand(ctx, "SISMEMBER Groups %s", groupname);
    int res = (0 == reply->integer);
    freeReplyObject(reply);

    if (res) {
        printf("There was no group with that id\n");
    } else {
        reply = redisCommand(ctx, "SADD %s %s", groupname, membername);
        freeReplyObject(reply);
    }
}
```

May need to remove members:

```
static void removeMember() {
    char groupname[128];
    char membername[128];
    printf("Removing student from group\n");
    printf("Group      : ");
    getstr(groupname, 128);
    printf("Student id : ");
    getstr(membername, 128);
    redisReply *reply;
    reply = redisCommand(ctx, "SISMEMBER Groups %s", groupname);
    int res = (0 == reply->integer);
    freeReplyObject(reply);

    if (res) {
        printf("There was no group with that id\n");
    } else {
        reply = redisCommand(ctx, "SREM %s %s", groupname, membername);
        freeReplyObject(reply);
    }
}
```

Project commands

The project commands are handled through another menu-select loop:

```
static void handleProjectCommands() {
    for (;;) {
        printf("Project=>");
        char command[128];
        getstr(command, 128);
        if (command[0] == 'q') break;
        if (command[0] == '?') {
            printf("Project Commands:\n");
            printf("\t? : print this command list\n");
            printf("\tq : finish project commands\n");
            printf("\tAdd: Add a project\n");
            printf("\tDelete: Delete a project\n");
            printf("\tList: List projects\n");
        }
        if (0 == strcmp(command, "Add")) addProject();
        else
            if (0 == strcmp(command, "Delete")) deleteProject();
        else
            if (0 == strcmp(command, "List")) listProjects();


    }
}
```

When a project is added, the title and name of the supervisor must be entered.

Titles will typically have multiple words separated by spaces, so we want to read the entire line. The gets() function in C is a pain with its handling of newline characters etc; it is better to use fgets which provides more control. Here, a little auxiliary function is used to read strings with fgets, and to remove the newline character from any string that is read.

```
static void getstr(char* str, int maxlen) {
    memset(str, 0, maxlen);
    fgets(str, maxlen, stdin);
    // fgets includes the terminating newline character for a string
    // will usually want this removed
    int len = strlen(str);
    len--;
    if (str[len] == '\n') str[len] = '\0';
}
```

The code for addProject is:

```
static void addProject() {
    printf("Project title : ");
    char title[128];
    getstr(title, 128);

    printf("Supervisor    : ");
    char supervisor[128];
    getstr(supervisor, 128);

    redisReply *reply;
    reply = redisCommand(ctx, "INCR projectcount");
    int acount = (int) reply->integer; // It is a long long, but we won't have that many projects
    freeReplyObject(reply);

    char projid[128];
    memset(projid, 0, 128);
    sprintf(projid, "Project:%d", acount);

    // Add the project id to the set of all projects
    reply = redisCommand(ctx, "SADD Projects %s", projid);
    freeReplyObject(reply);
    // Create a hash named by project id with values for title and supervisor
    reply = redisCommand(ctx, "HMSET  %s Title %s Supervisor %s",
            projid, title, supervisor);
    freeReplyObject(reply);
}
```

The Redis counter variable projectcount is incremented and used when creating a project id. This id is added to the set of project identifiers maintained in Redis. A simple hash table structure is then created as the value of the project record in Redis.

Projects may get deleted:

```c
static void deleteProject() {
    char pname[128];

    printf("Delete project with id : ");
    getstr(pname, 128);

    redisReply *reply;
    reply = redisCommand(ctx, "SISMEMBER Projects %s", pname);
    int res = (0 == reply->integer);
    freeReplyObject(reply);

    if (res) {
        printf("There was no project with that id\n");
    } else {
        // Remove id from set
        reply = redisCommand(ctx, "SREM Projects %s", pname);
        // Not checking confirmation, just assume it works
        freeReplyObject(reply);
        // Now remove the project details
        reply = redisCommand(ctx, "DEL %s", pname);
        freeReplyObject(reply);
    }

}
```

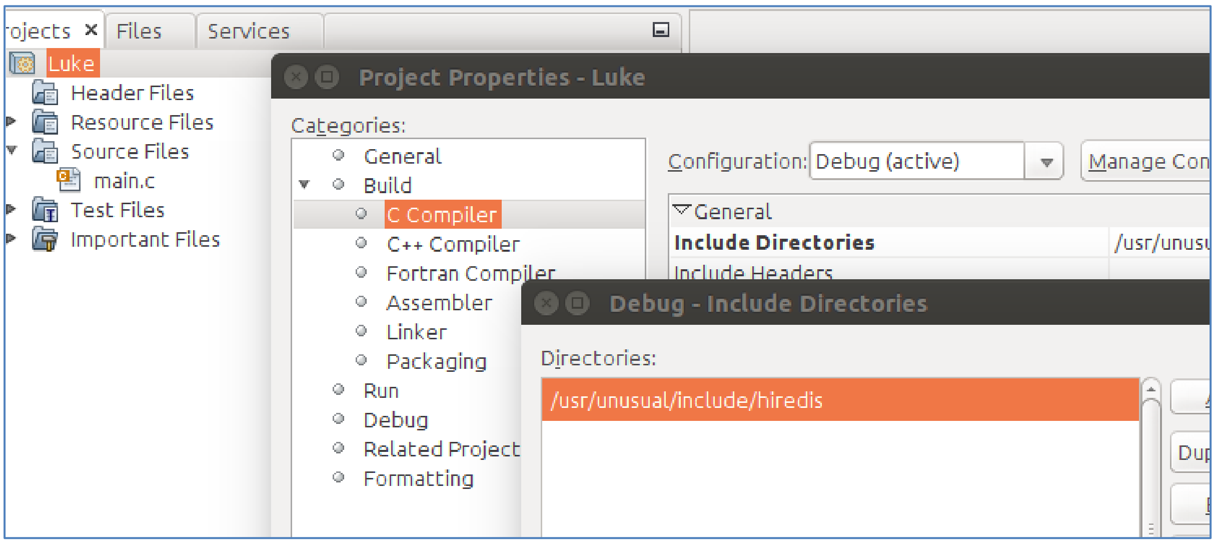
They get listed:

```
static void listProjects() {
    redisReply *reply;
    reply = redisCommand(ctx, "SCARD Projects");

    int count = (int) reply->integer;
    freeReplyObject(reply);
    printf("There are %d projects currently in the database\n",
            count);
    char** projnames = malloc(count * sizeof (char*));
    reply = redisCommand(ctx, "SMEMBERS Projects");
    int j;
    for (j = 0; j < reply->elements; j++) {
        projnames[j] = strdup(reply->element[j]->str);
    }
    freeReplyObject(reply);

    for (j = 0; j < count; j++) {
        reply = redisCommand(ctx, "HGETALL %s", projnames[j]);
        printf("Project %s\n", projnames[j]);
        int k;
        for (k = 0; k < reply->elements; k += 2) {
            printf("%s: %s\n",
                    reply->element[k]->str,
                    reply->element[k + 1]->str);
        }
        freeReplyObject(reply);
    }

    for (j = 0; j < count; j++)
        free(projnames[j]);
    free(projnames);
}
```

Building the project in NetBeans

The project properties will have to be edited in NetBeans. As 'hiredis' is a non-standard library in a non-standard location, the compiler and linker must both be given details of where to find header files and link files.

The compiler will need extra include directories:



Project Properties - Luke

Categories:
- General
- Build
  - C Compiler
  - C++ Compiler
  - Fortran Compiler
  - Assembler
  - Linker
  - Packaging
- Run
- Debug
- Related Project
- Formatting

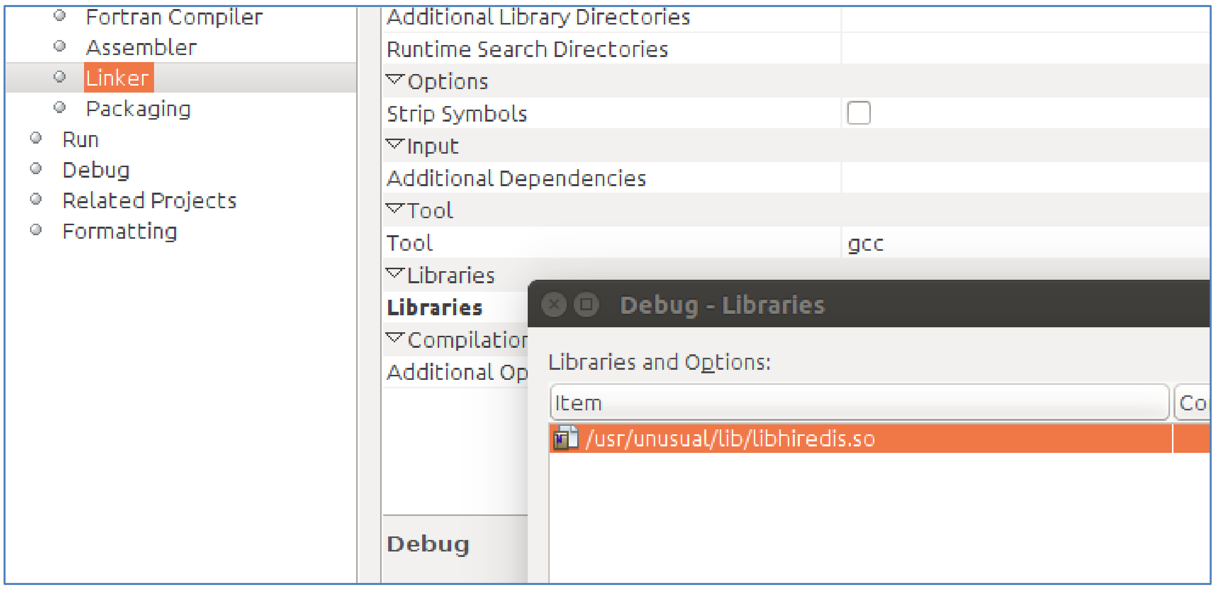
Configuration: Debug (active)

General
Include Directories /usr/unusu
Include Headers

Debug - Include Directories

Directories:
/usr/unusual/include/hiredis

The linker needs the library. This build is using the .so shared object library; so the linking process is really just checking that functions are defined there in the library. A complete executable is not actually built:



Additional Library Directories
Runtime Search Directories
Options
Strip Symbols ☐
Input
Additional Dependencies
Tool
Tool gcc
Libraries
Libraries
Compilation
Additional Op

Debug

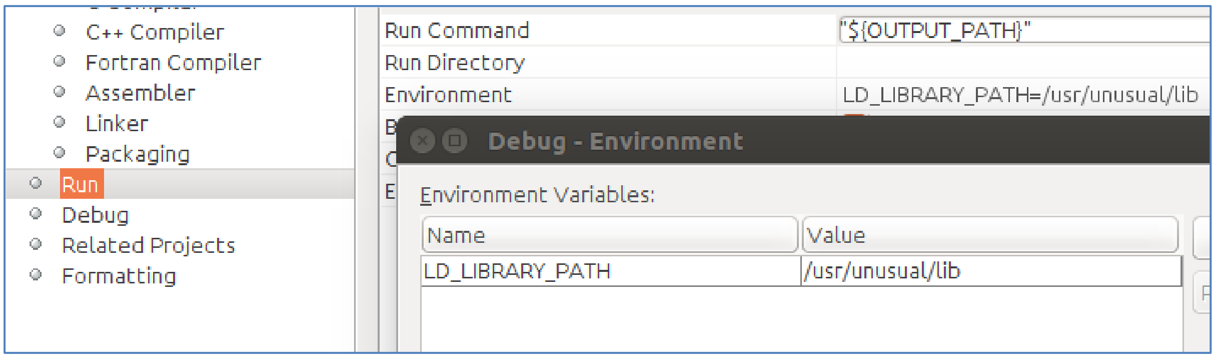
Debug - Libraries

Libraries and Options:

Item	Co
/usr/unusual/lib/libhiredis.so	

If you simply tried to run the application as built, it would fail at run-time as it wouldn't find the libhiredis.so library.

In a case like this, it is necessary to set the LD_LIBRARY_PATH so that at run-time the libhiredis.so library can again be found. In NetBeans this is achieved by setting a run-time environment variable.



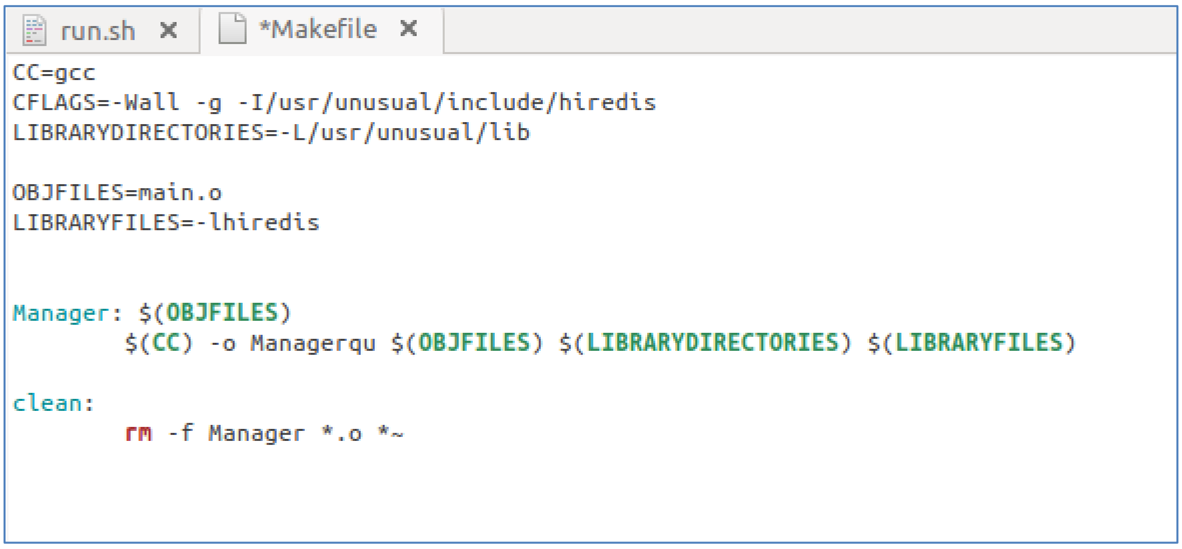
Run Command "${OUTPUT_PATH}"
Run Directory
Environment LD_LIBRARY_PATH=/usr/unusual/lib

Name	Value
LD_LIBRARY_PATH	/usr/unusual/lib

You should be able to build and run the application from inside NetBeans.

Building the project without NetBeans

Once you have got the program to run correctly via NetBeans, you should create a stand-alone copy using your own makefile.



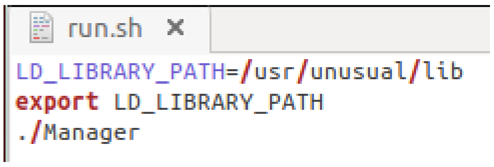
```
CC=gcc
CFLAGS=-Wall -g -I/usr/unusual/include/hiredis
LIBRARYDIRECTORIES=-L/usr/unusual/lib

OBJFILES=main.o
LIBRARYFILES=-lhiredis


Manager: $(OBJFILES)
	$(CC) -o Managerqu $(OBJFILES) $(LIBRARYDIRECTORIES) $(LIBRARYFILES)

clean:
	rm -f Manager *.o *~
```

The value of CFLAGS now has a –I element – this is adding the location where the compiler can find the <hiredis.h> file that is included by the program. The LIBRARYFILES and LIBRARYDIRECTORIES elements identify the location and library name for use by the linker. This script builds the application as Manager.

If you attempt to run this application, it will fail at run-time. The dynamic linker will not find the libhiredis.so file. You must set the environment variable LD_LIBRARY_PATH, and 'export' the value once set. As noted in the lectures, this tends to be error prone.

One trick that is commonly used for programs that require unusual values for environment variables is to create a shell script that sets those values and then runs the program. The application is then always launched via this shell script.

For this example, the script would be:



```
LD_LIBRARY_PATH=/usr/unusual/lib
export LD_LIBRARY_PATH
./Manager
```

(Create the script, and assign appropriate execute permissions.) You should be able to get the application to run at the command line.

Exercises complete

Show the tutor that you have completed all the parts.

(That was 6 very easy marks wasn't it – just copy my code. Hopefully, you learnt a little more C, have become more aware of the libraries that are standard, and are more confident in your use of make and makefiles.)

Assignment

The assignment: build an application exploiting several common, and less common, libraries.