Marks : 12

Due date: October 9th changed to 16th October

CSCI131

Spring Session 2015

Assignment 4
Libraries and the build process
6 marks

Complete the associated exercises before attempting the assignment

Aim

This assignment and its associated exercises provide experience in the use of libraries and more complex program build processes.

Objectives

On completion of this assignment and its associated exercise, you will be able to:

- Explain the differences between static and dynamic linking
- Link code that you have written with supplied libraries.
- Create makefiles

Task domain for example application – *generating a graphic CAPTCHA*

You will often have encountered CAPTCHAs when using the web. They are intended to block features of web-sites from automated bots, spiders, scanners, scripts etc. The web server displays a puzzle that the user must solve before they can advance to the controlled web resource. The puzzle is displayed in a web page, the user's "solution" is sent back to the server for checking; only if the solution is correct may the user advance to the controlled web resource.

"A CAPTCHA (an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge-response test used in computing to determine whether or not the user is human.

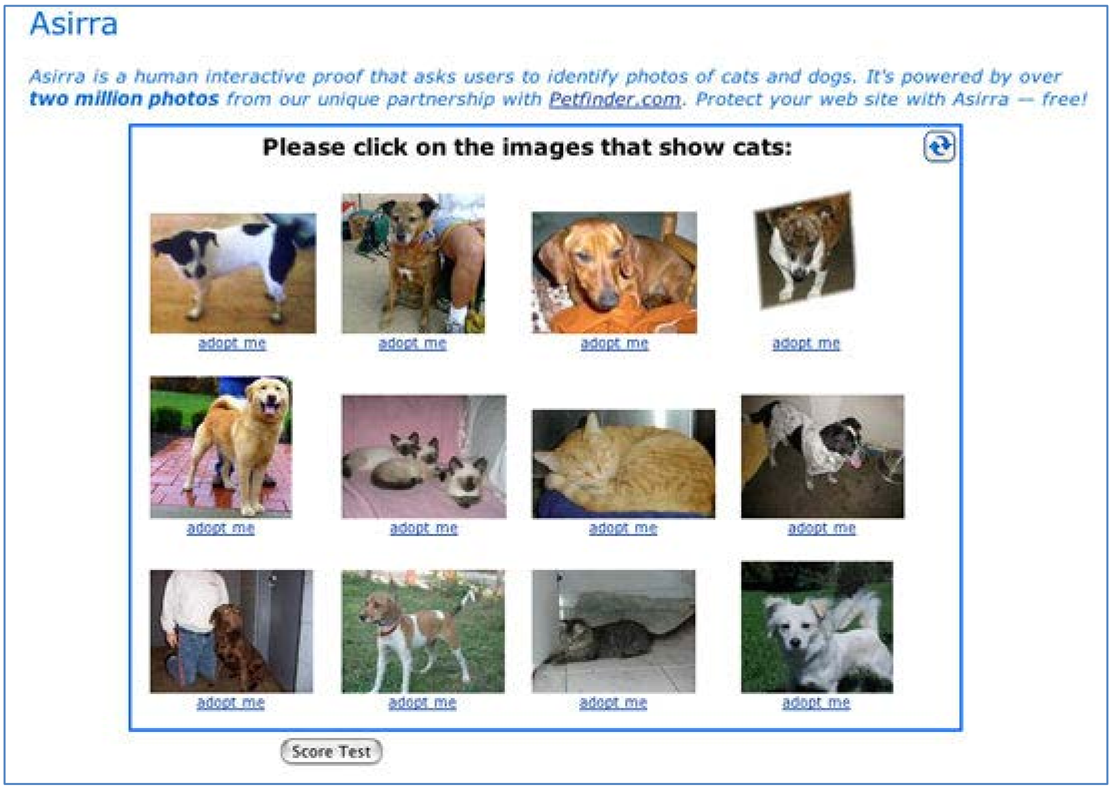
The term was coined in 2000 by Luis von Ahn, Manuel Blum, Nicholas J. Hopper of Carnegie Mellon University and John Langford of IBM. The most common type of CAPTCHA was first invented by Mark D. Lillibridge, Martin Abadi, Krishna Bharat and Andrei Z. Broder. This form of CAPTCHA requires that the user type the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen." (So says Wikipedia.)

Wikipedia adds: "*This user identification procedure has received many criticisms, especially from disabled people, but also from other people who feel that their everyday work is slowed down by distorted words that are illegible even for users with no disabilities at all."*

Apart from being a pain for humans solve, these distorted letter patterns are increasingly vulnerable.

"However, our research recently showed that today's Artificial Intelligence technology can solve even the most difficult variant of distorted text at 99.8% accuracy. Thus distorted text, on its own, is no longer a dependable test." http://googleonlinesecurity.blogspot.com.au/2014/12/are-you-robot-introducing-no-captcha.html

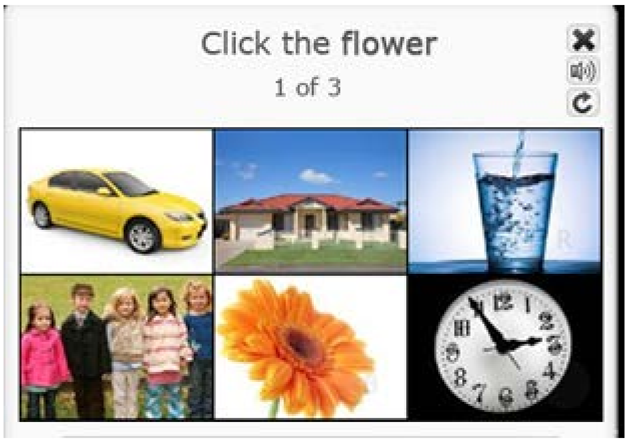
A number of organisations have created alternative "picture recognition" based CAPTCHAs. For example, Microsoft created the Asirra CAPTCHA:


Asirra
Asirra is a human interactive proof that asks users to identify photos of cats and dogs. It's powered by over two million photos from our unique partnership with Petfinder.com. Protect your web site with Asirra — free!
Please click on the images that show cats:
adopt me
Score Test

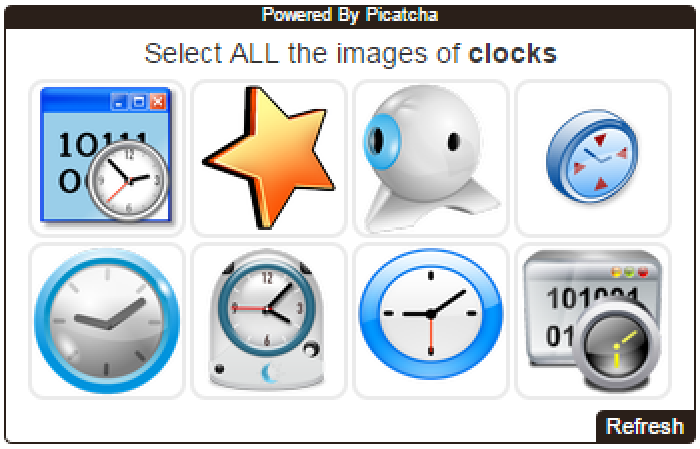
"Asirra is easy for users; it can be solved by humans 99.6% of the time in under 30 seconds. Anecdotally, users seemed to find the experience of using Asirra much more enjoyable than a text-based CAPTCHA."

There are a number of similar projects –

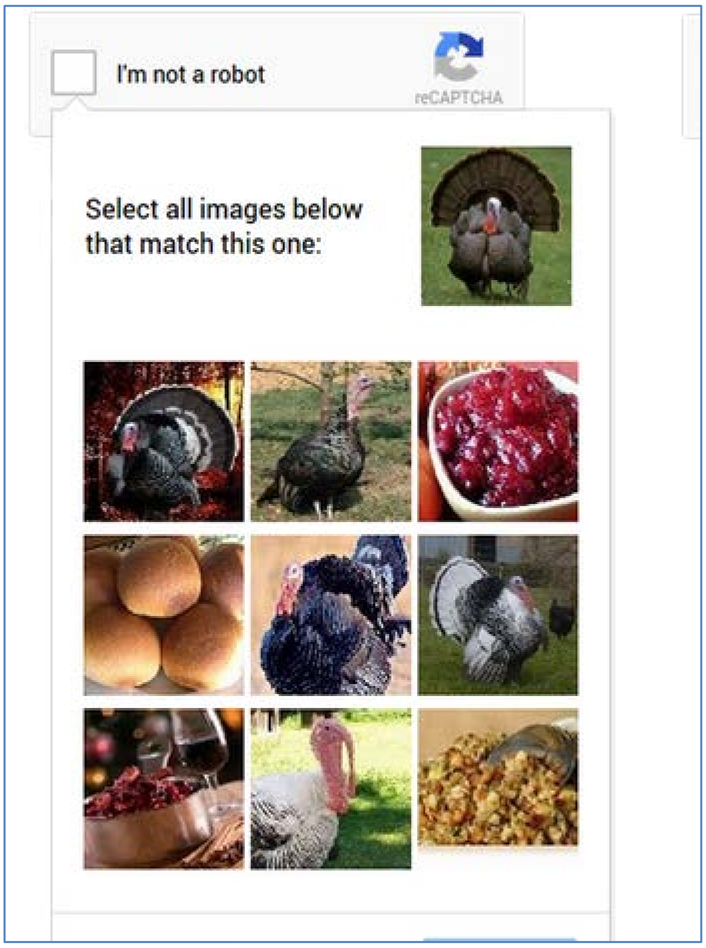
Confidentcaptcha.com



Piccaptcha.com



Google's own version:



While currently more secure than corrupted text, these picture based approaches are still vulnerable.

That collection of 2 million dog and cat photos – *only 2 million*, hackers have infinite time and can it seems map out the space of picture files onto dog | cat. Other attacks may also be possible - https://www.linkedin.com/pulse/20140417144957-237781962-cracking-ms-asirra-captchas-with-google-repost-from-blog .

And these picture based approaches are an excellent area for PhD students to create thesis projects for recognising images - http://epub.uni-regensburg.de/16872/1/trustbus_1.pdf .

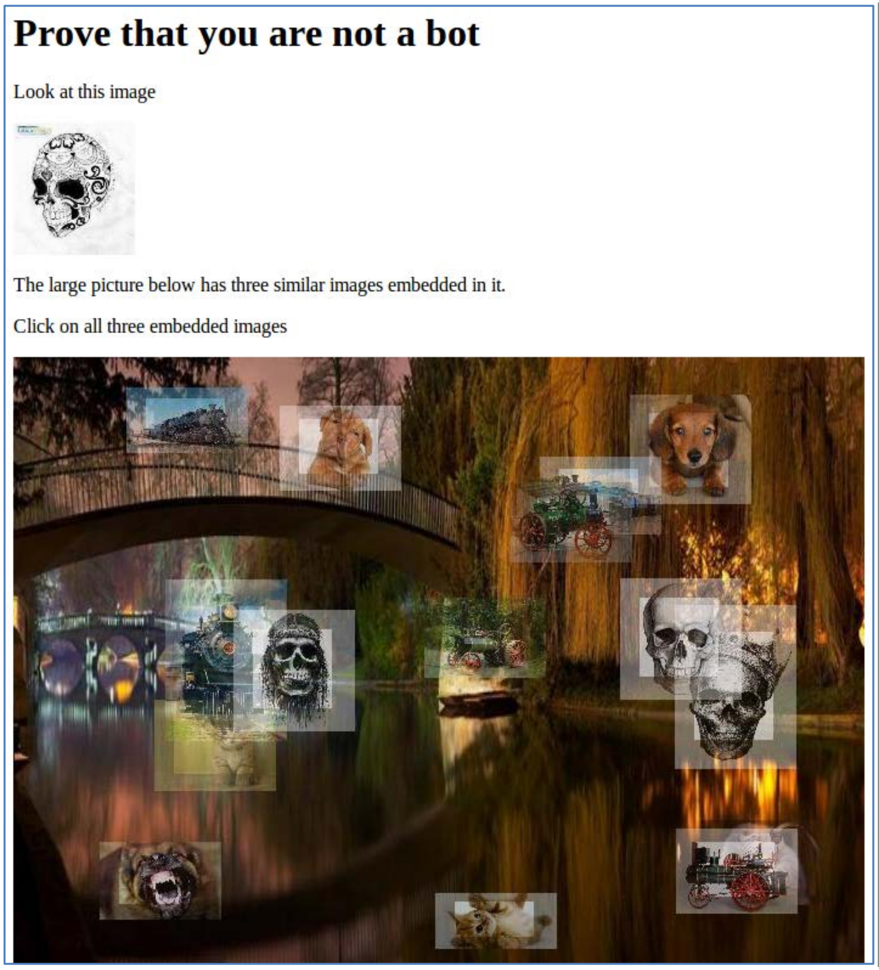
So what you will be doing in this exercise is creating a component for a slightly more challenging variant!

This variant works as follows:

- For every CAPTCHA test the server web server will generate a complex and quite unique image that has embedded in it a set of sub-images of a given type, it records the position of these sub-images for use when verifying the users response. (This version uses a fixed number of sub-images, but could easily be generalised.)
  It sends this unique generated image to the client's browser.

- The page sent to the browser incorporates the image and some Javascript code.
  The CAPTCHA test requires the user to click on the embedded sub-images; the positions of the clicks are captured by Javascript and sent in a verification request using AJAX.
- The server receives the user input and checks that the user clicked within the areas of the sub-images. If the user input is valid, the server creates session data "*not a bot*" that will allow the user to reach controlled web resources.

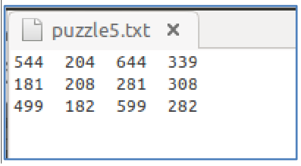
As shown in the following examples, the overall image is comprised of a background photo (or abstract patterned image) and a large number of embedded partially transparent sub-images. These sub-images are taken from several different collections. One group of sub-images constitutes the target for the user – as identified by a different sub-image of similar type.


Prove that you are not a bot

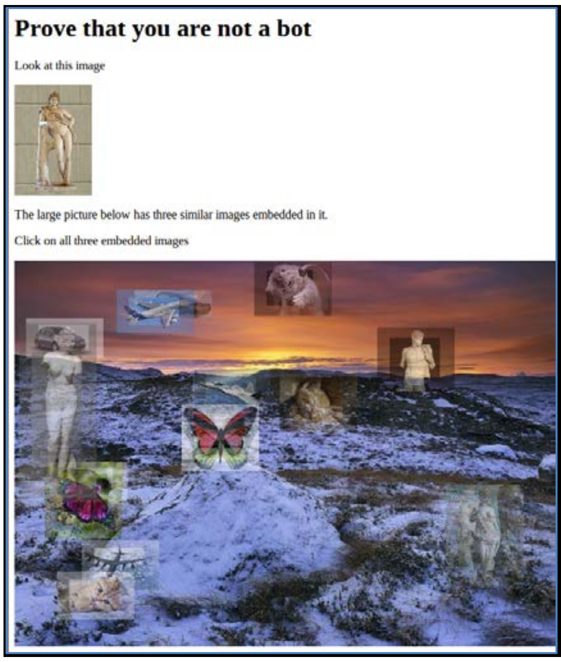
Look at this image

The large picture below has three similar images embedded in it.

Click on all three embedded images



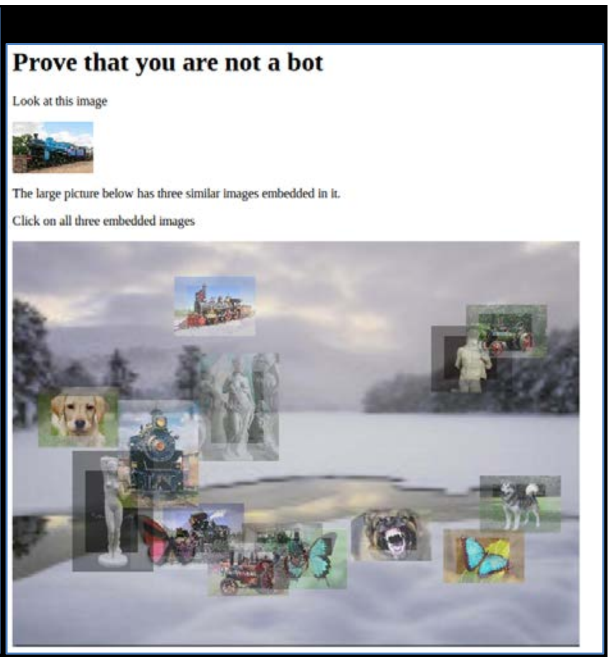
```
544  204  644  339
181  208  281  308
499  182  599  282
```


Prove that you are not a bot

Look at this image

The large picture below has three similar images embedded in it.

Click on all three embedded images


Prove that you are not a bot

Look at this image

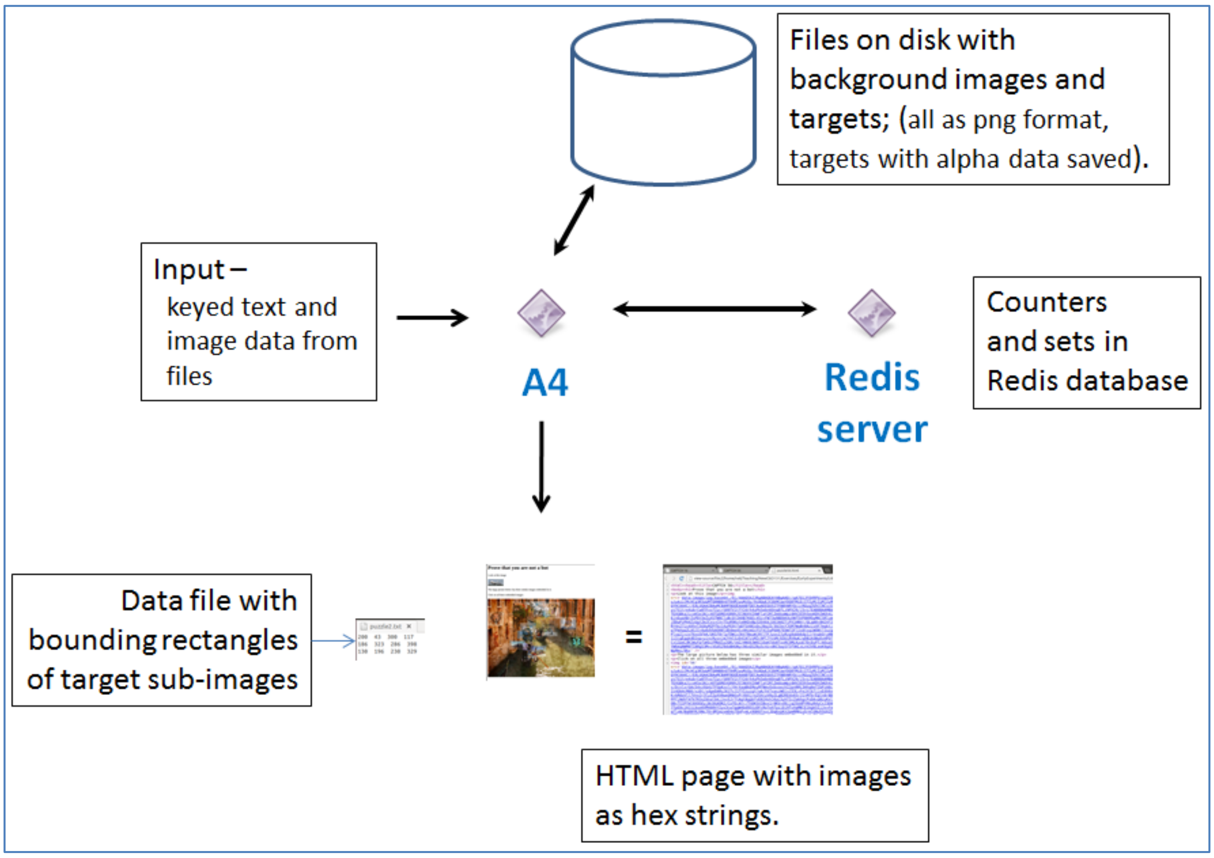
The large picture below has three similar images embedded in it.

Click on all three embedded images

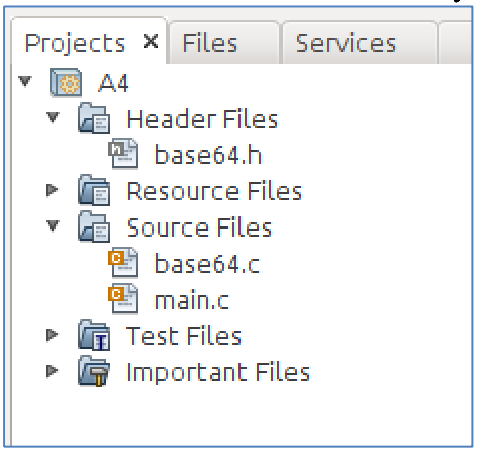
The code that you write for this exercise is the C code for firstly creating an image collection, and then generating HTML pages with images along with the files containing the associated data that define the position of the embedded sub-images that the user is to identify. (Your HTML pages don't include the Javascript that would be required; this code would be added via a HTML <script > link.)

Task

The application


Files on disk with background images and targets; (all as png format, targets with alpha data saved).
Input – keyed text and image data from files
A4
Redis server
Counters and sets in Redis database
Data file with bounding rectangles of target sub-images
=
HTML page with images as hex strings.

You are to build the application firstly as a NetBeans project, and then when it works you are to create a standalone version with your own makefile.



Another menu-select program!

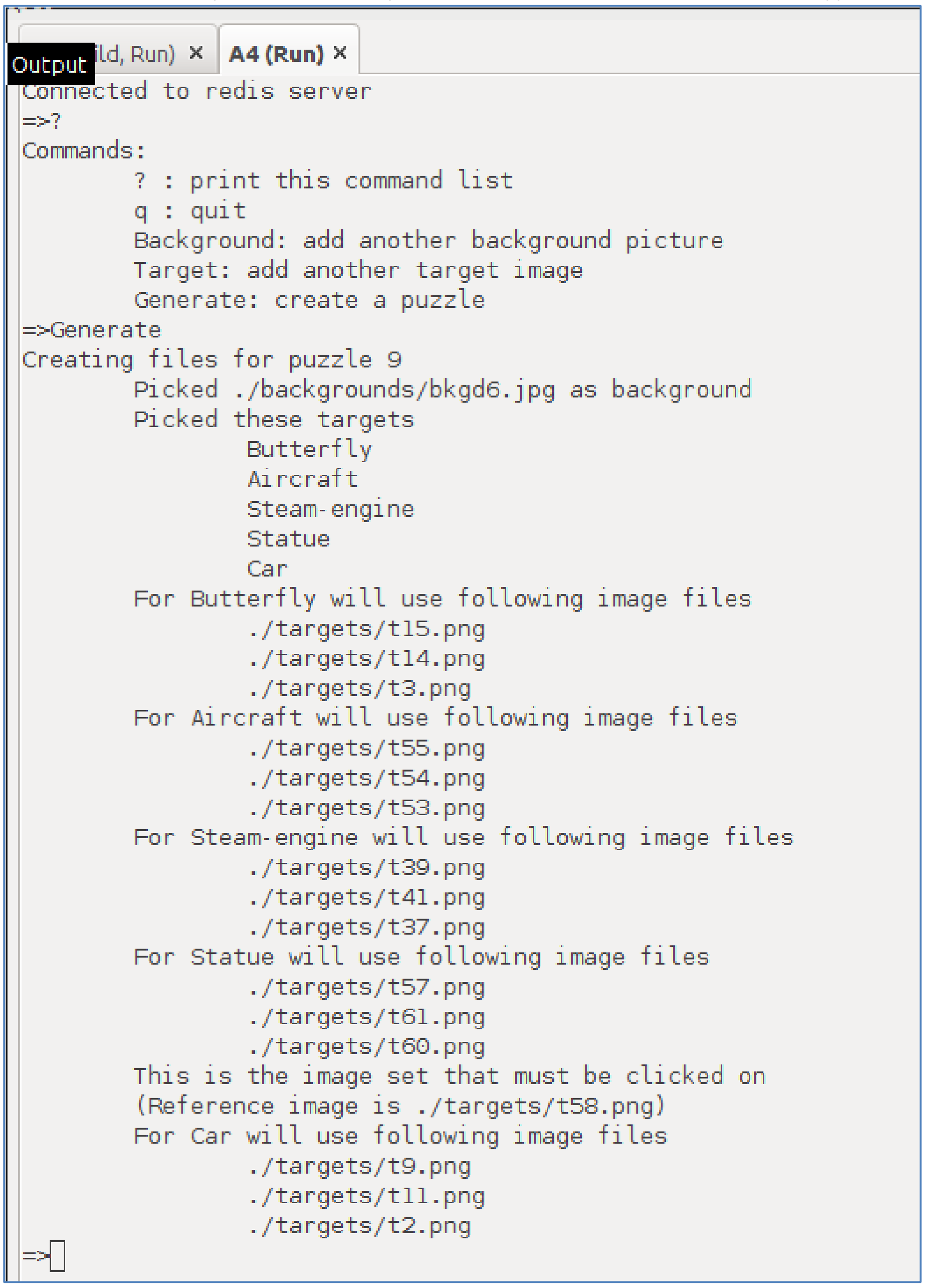
This version of the program is simply an exercise and incorporates both the code to build up the image collection and the code to generate puzzles. (A realistic implementation would split these aspects into different applications.)

```c
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("\tBackground: add another background picture\n");
            printf("\tTarget: add another target image\n");
            printf("\tGenerate: create a puzzle\n");
        }
        if (0 == strcmp(command, "Background")) addBackground();
        else
            if (0 == strcmp(command, "Target")) addTarget();
        else
            if (0 == strcmp(command, "Generate")) generatePuzzle();
    }
}

int main(int argc, char** argv) {
    setupConnection();
    // The directories ./backgrounds and ./targets are required
    // create them if they do not exist
    createDirectory(backgrounddir);
    createDirectory(targetsdir);

    menu_select_loop();
    closeConnection();
    return (EXIT_SUCCESS);
}
```

Example Use

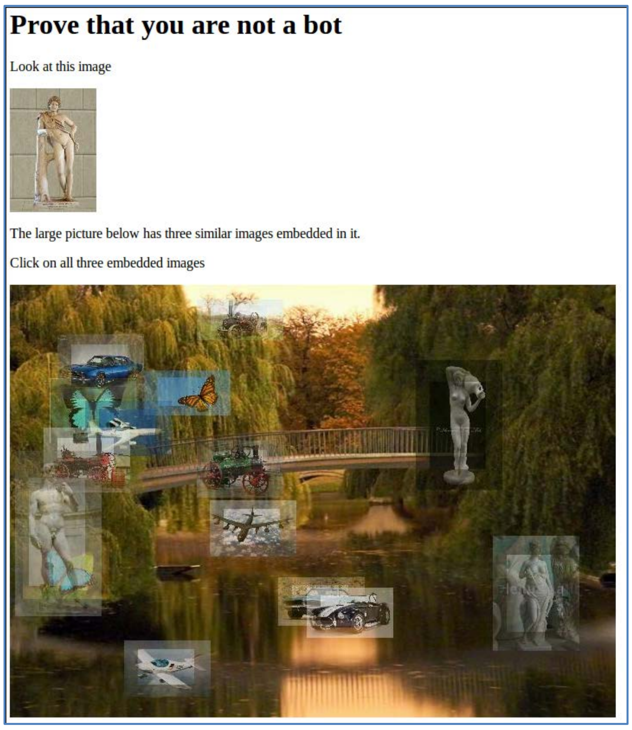
Generate a puzzle:

- This version of the program is to generate a log that specifies the image selected as a background, the types of sub-image to embed (at least 5 different types should be used in each generated puzzle image), the specific sub-images selected (at least 3 from each different image type), and also identifies the sub-images that the user must select.
- In this case, the background bkgd6.jpg was used (backgrounds can be jpg, but .png must be used for sub-images as these require transparency data).
- The target sets were Butterfly, Aircraft, Steam-engine, Statue, and Car; 3 pictures were picked from each set. "Statue" was the set selected, and an additional statue sub-image is used. (Target sets are picked randomly from the set of all possible target types. Image files are picked randomly from the set associated with chosen type.)



```
Connected to redis server
=>?
Commands:
        ? : print this command list
        q : quit
        Background: add another background picture
        Target: add another target image
        Generate: create a puzzle
=>Generate
Creating files for puzzle 9
        Picked ./backgrounds/bkgd6.jpg as background
        Picked these targets
                Butterfly
                Aircraft
                Steam-engine
                Statue
                Car
        For Butterfly will use following image files
                ./targets/t15.png
                ./targets/t14.png
                ./targets/t3.png
        For Aircraft will use following image files
                ./targets/t55.png
                ./targets/t54.png
                ./targets/t53.png
        For Steam-engine will use following image files
                ./targets/t39.png
                ./targets/t41.png
                ./targets/t37.png
        For Statue will use following image files
                ./targets/t57.png
                ./targets/t61.png
                ./targets/t60.png
        This is the image set that must be clicked on
        (Reference image is ./targets/t58.png)
        For Car will use following image files
                ./targets/t9.png
                ./targets/t11.png
                ./targets/t2.png

=>
```

This generated the HTML page:



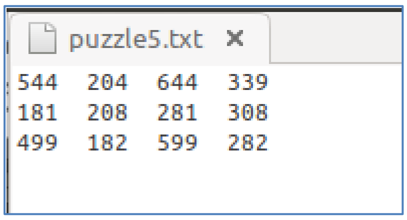
Prove that you are not a bot

Look at this image

The large picture below has three similar images embedded in it.

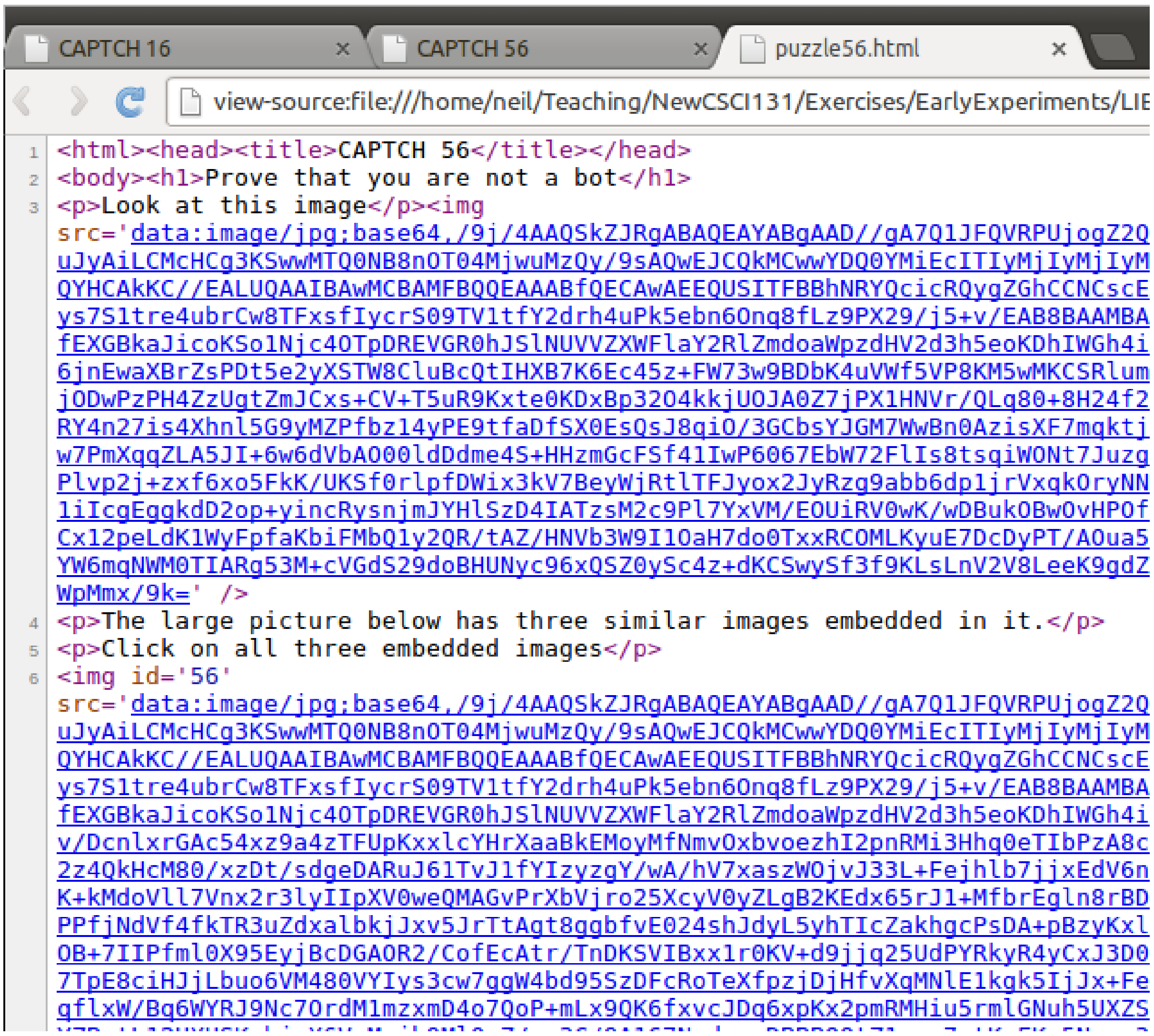
Click on all three embedded images

along with a file with the coordinates for the bounding rectangles for the target sub-images, something similar to the following:



```
544  204  644  339
181  208  281  308
499  182  599  282
```

The generated HTML file contains the base-64 encoded version of the images:



CAPTCH 16 | CAPTCH 56 | puzzle56.html

view-source:file:///home/neil/Teaching/NewCSCI131/Exercises/EarlyExperiments/LIE

```
<html><head><title>CAPTCH 56</title></head>
<body><h1>Prove that you are not a bot</h1>
<p>Look at this image</p><img
src='data:image/jpg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD//gA7Q1JFQVRPUjogZ2Q
```

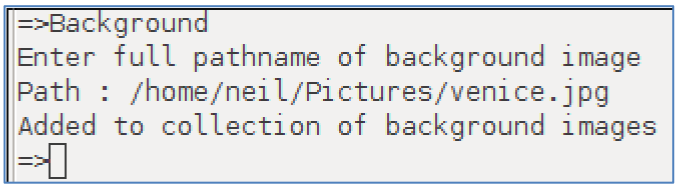
uJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyM
QYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscE
ys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBA
fEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4i
6jnEwaXBrZsPDt5e2yXSTW8CluBcQtIHXB7K6Ec45z+FW73w9BDbK4uVWf5VP8KM5wMKCSRlum
jODwPzPH4ZzUgtZmJCxs+CV+T5uR9Kxte0KDxBp32O4kkjUOJA0Z7jPX1HNVr/QLq80+8H24f2
RY4n27is4Xhnl5G9yMZPfbz14yPE9tfaDfSX0EsQsJ8qiO/3GCbsYJGM7WwBn0AzisXF7mqktj
w7PmXqqZLA5JI+6w6dVbAO00ldDdme4S+HHzmGcFSf41IwP6067EbW72FlIs8tsqiWONt7Juzg
Plvp2j+zxf6xo5FkK/UKSf0rlpfDWix3kV7BeyWjRtlTFJyox2JyRzg9abb6dp1jrVxqkOryNN
1iIcgEggkdD2op+yincRysnjmJYHlSzD4IATzsM2c9Pl7YxVM/EOUiRV0wK/wDBukOBwOvHPOf
Cx12peLdK1WyFpfaKbiFMbQ1y2QR/tAZ/HNVb3W9I1OaH7do0TxxRCOMLKyuE7DcDyPT/AOua5
YW6mqNWM0TIARg53M+cVGdS29doBHUNyc96xQSZ0ySc4z+dKCSwySf3f9KLsLnV2V8LeeK9gdZ
WpMmx/9k=' />

```
<p>The large picture below has three similar images embedded in it.</p>
<p>Click on all three embedded images</p>
<img id='56'
src='data:image/jpg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD//gA7Q1JFQVRPUjogZ2Q
```

uJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyM
QYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscE
ys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBA
fEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4i
v/DcnlxrGAc54xz9a4zTFUpKxxlcYHrXaaBkEMoyMfNmvOxbvoezhI2pnRMi3Hhq0eTIbPzA8c
2z4QkHcM80/xzDt/sdgeDARuJ61TvJ1fYIzyzgY/wA/hV7xaszWOjvJ33L+Fejhlb7jjxEdV6n
K+kMdoVll7Vnx2r3lyIIpXV0weQMAGvPrXbVjro25XcyV0yZLgB2KEdx65rJ1+MfbrEgln8rBD
PPfjNdVf4fkTR3uZdxalbkjJxv5JrTtAgt8ggbfvE024shJdyL5yhTIcZakhgcPsDA+pBzyKxl
OB+7IIPfml0X95EyjBcDGAOR2/CofEcAtr/TnDKSVIBxx1r0KV+d9jjq25R4yCxJ3D0
7TpE8ciHJjLbuo6VM480VYIys3cw7ggW4bd95SzDFcRoTeXfpzjDjHfvXqMNlE1kgk5IjJx+Fe
qflxW/Bq6WYRJ9Nc70rdM1mzxmD4o7QoP+mLx9QK6fxvcJDq6xpKx2pmRMHiu5rmlGNuh5UXZS

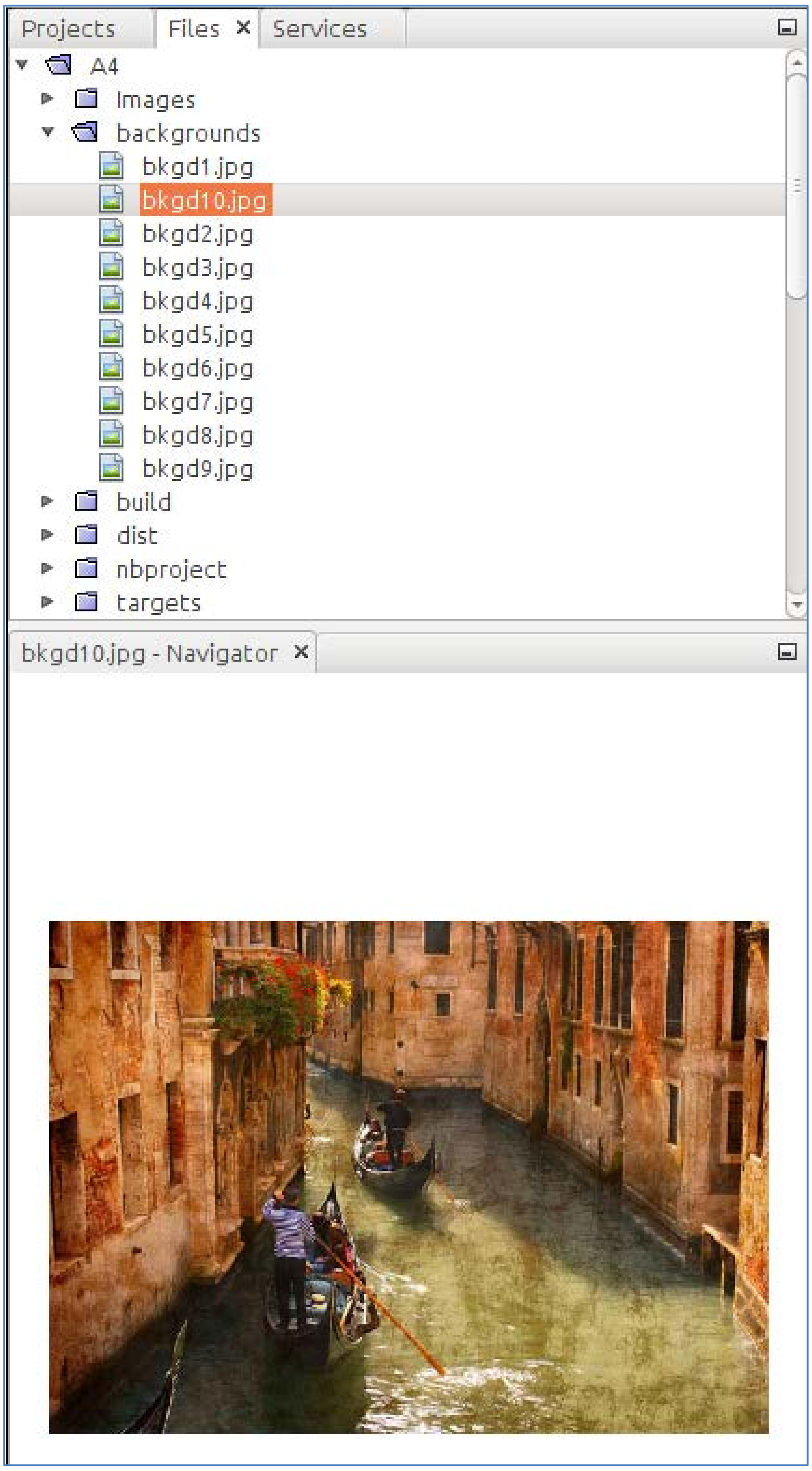
<span style="color:red">Add another background image:</span>

The program prompts for the filename, reads in the image file, scales the image to a standard width (about 700px), and saves the background as a .jpg file in a “backgrounds” directory. The Redis database is also updated. The Redis database has a counter for background images (used to generate a name for the background image file in the backgrounds directory), and a set “bkgrdimgs” whose members are the names of the files with backgrounds.



```
=>Background
Enter full pathname of background image
Path : /home/neil/Pictures/venice.jpg
Added to collection of background images
=>
```

The scaled picture of Venice became the 10th image in the backgrounds collection. (I used photos as backgrounds, but in practice it might be better to use abstract images composed of multiple overlayed figures and lines in many different colours.)



Projects | Files | Services

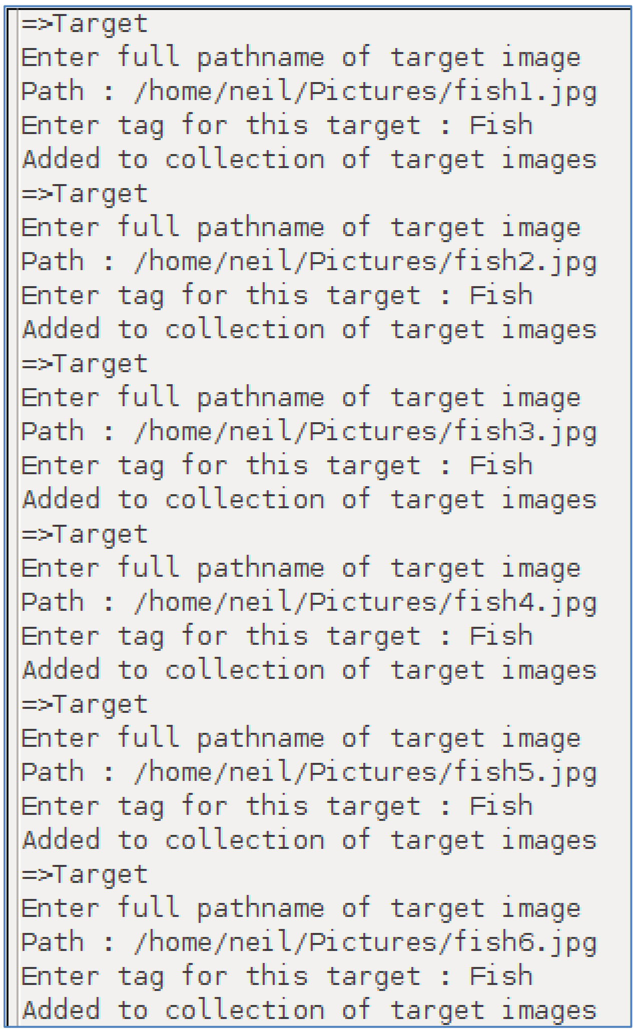
- A4
  - Images
  - backgrounds
    - bkgd1.jpg
    - bkgd10.jpg
    - bkgd2.jpg
    - bkgd3.jpg
    - bkgd4.jpg
    - bkgd5.jpg
    - bkgd6.jpg
    - bkgd7.jpg
    - bkgd8.jpg
    - bkgd9.jpg
  - build
  - dist
  - nbproject
  - targets

bkgd10.jpg - Navigator

Add another “target”:

In my implementation, sub-images are referred to as “targets”. The program is given the name of an image file with the additional sub-image. The image is read. It is then scaled to a fixed 100px width. It is then made partially transparent by adjusting the alpha values for each pixel; the transparency makes the targets merge into the background rather than existing as recognizably distinct areas. The scaled, partially transparent image is then saved to a file in a “targets” directory. Here, .png format must be used so as to preserve alpha channel data.

The program also gets the user to assign a “tag” for the image. This tag is used to group similar images.

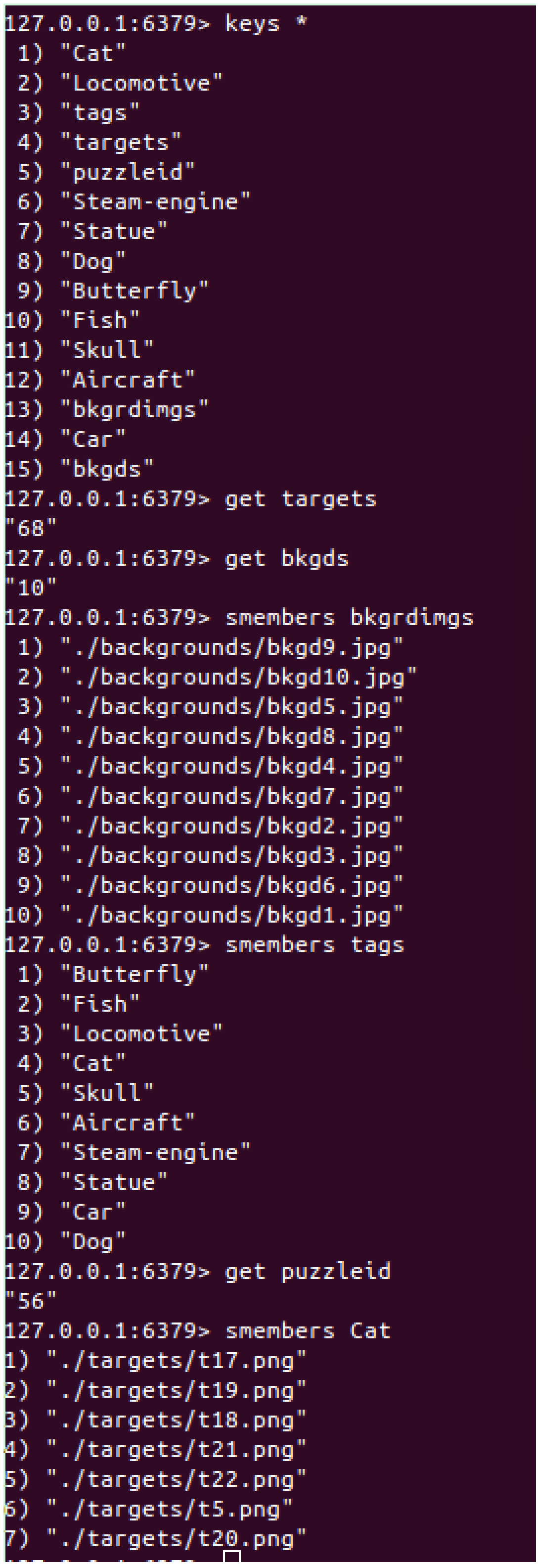


```
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish1.jpg
Enter tag for this target : Fish
Added to collection of target images
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish2.jpg
Enter tag for this target : Fish
Added to collection of target images
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish3.jpg
Enter tag for this target : Fish
Added to collection of target images
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish4.jpg
Enter tag for this target : Fish
Added to collection of target images
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish5.jpg
Enter tag for this target : Fish
Added to collection of target images
=>Target
Enter full pathname of target image
Path : /home/neil/Pictures/fish6.jpg
Enter tag for this target : Fish
Added to collection of target images
```

As targets are added, records in the Redis database are updated. There is a target counter; this is used to generate unique names for the .png files created in the ‘targets’ directory. There is a set that contains the names for all distinct tags – “Fish”, “Car”, “Locomotive”, “Statue”, “Butterfly”, …

Each tag is associated with a set – the identifiers of the target files given that tag.

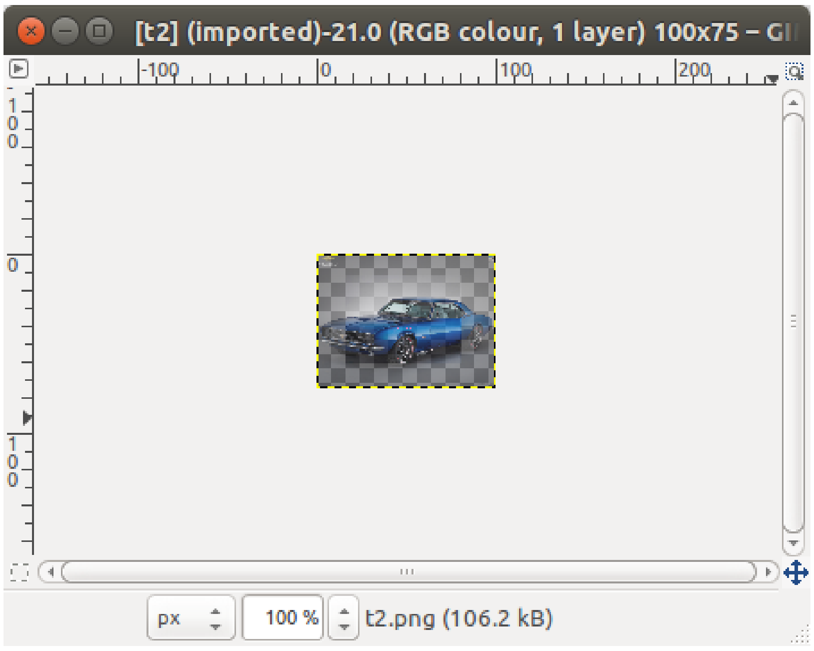
The following view of the Redis database shows that there were ten backgrounds and 68 different target images. The” tags” set contains the names of the different target groups. Each group has a set of associated images; the contents of the set “Cat” is shown. The puzzleid gets incorporated in the generated HTML page and is also used to name the file that is used to hold the bounding rectangles of the sub-image targets.



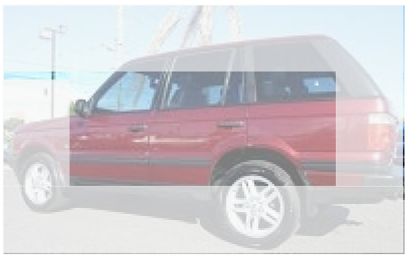
```
127.0.0.1:6379> keys *
 1) "Cat"
 2) "Locomotive"
 3) "tags"
 4) "targets"
 5) "puzzleid"
 6) "Steam-engine"
 7) "Statue"
 8) "Dog"
 9) "Butterfly"
10) "Fish"
11) "Skull"
12) "Aircraft"
13) "bkgrdimgs"
14) "Car"
15) "bkgds"
127.0.0.1:6379> get targets
"68"
127.0.0.1:6379> get bkgds
"10"
127.0.0.1:6379> smembers bkgrdimgs
 1) "./backgrounds/bkgd9.jpg"
 2) "./backgrounds/bkgd10.jpg"
 3) "./backgrounds/bkgd5.jpg"
 4) "./backgrounds/bkgd8.jpg"
 5) "./backgrounds/bkgd4.jpg"
 6) "./backgrounds/bkgd7.jpg"
 7) "./backgrounds/bkgd2.jpg"
 8) "./backgrounds/bkgd3.jpg"
 9) "./backgrounds/bkgd6.jpg"
10) "./backgrounds/bkgd1.jpg"
127.0.0.1:6379> smembers tags
 1) "Butterfly"
 2) "Fish"
 3) "Locomotive"
 4) "Cat"
 5) "Skull"
 6) "Aircraft"
 7) "Steam-engine"
 8) "Statue"
 9) "Car"
10) "Dog"
127.0.0.1:6379> get puzzleid
"56"
127.0.0.1:6379> smembers Cat
1) "./targets/t17.png"
2) "./targets/t19.png"
3) "./targets/t18.png"
4) "./targets/t21.png"
5) "./targets/t22.png"
6) "./targets/t5.png"
7) "./targets/t20.png"
```

Partial transparency

The target images have alpha channel transparency values defined.



The transparency is handled somewhat naively by adjusting each pixel in the image. (There may be a better way to do this, but the gd documentation was not helpful). The outer region is made more transparent than the centre:



The code is along the following lines –

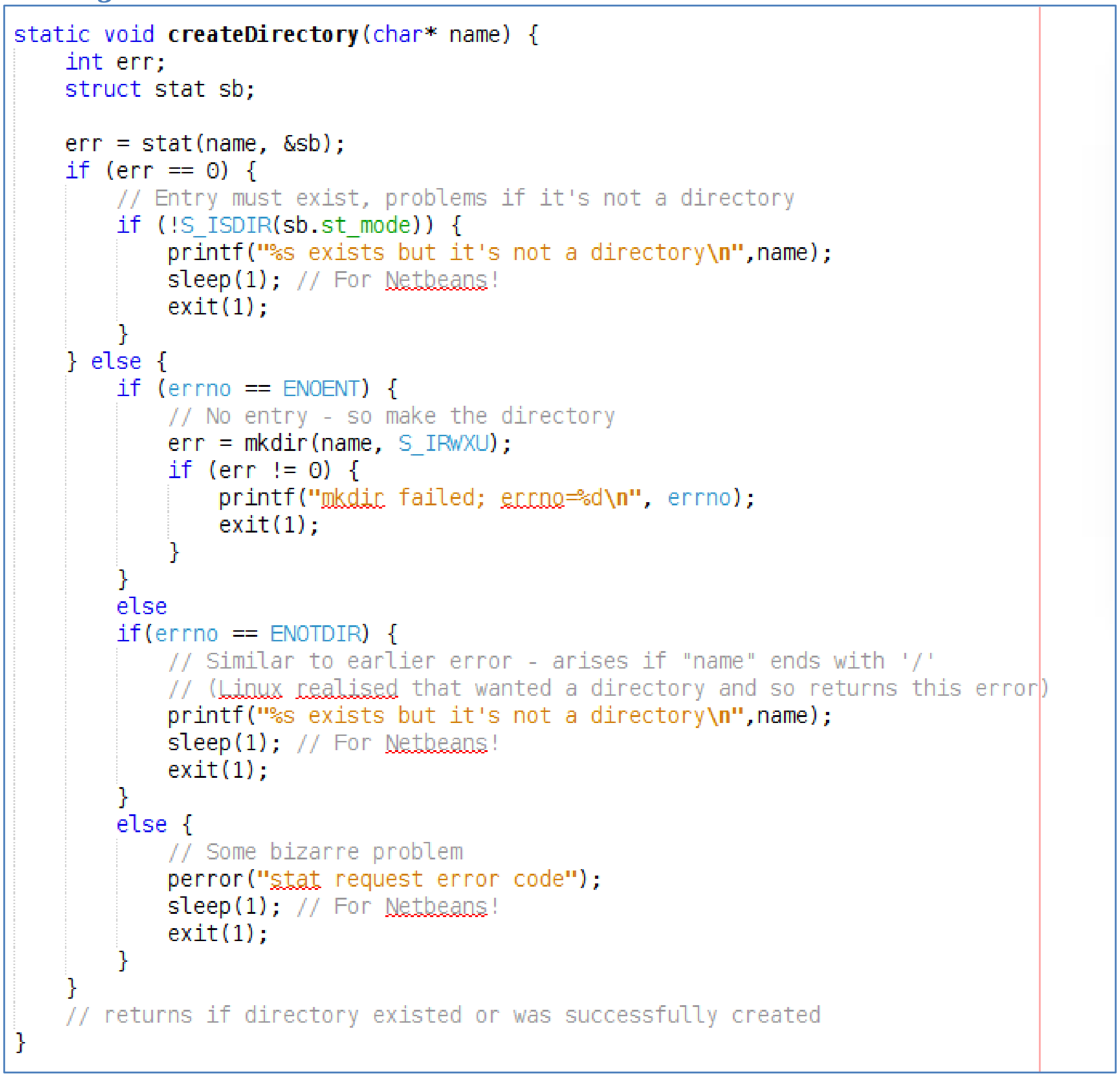
```
// Add transparency
// Rim more transparent than centre
gdImageAlphaBlending(ims, 0);
int w = 100;
int h = nh;
int h1 = h / 6;
int h2 = 5 * h1;
int l1 = w / 6;
int l2 = 5 * l1;
int row, col;

for (row = 0; row < h; row++) {
    for (col = 0; col < w; col++) {
        int aval = 40;
        if ((row < h1) || (row > h2))
            aval = 80;
        else
            if ((col < l1) || (col > l2))
            aval = 80;

        int c = gdImageGetPixel(ims, col, row); // Current colour
        int r = gdImageRed(ims, c);
        int g = gdImageGreen(ims, c);
        int b = gdImageBlue(ims, c);
        int c1 = gdImageColorAllocateAlpha(ims, r, g, b, aval);
        gdImageSetPixel(ims, col, row, c1);
    }
}

gdImageSaveAlpha(ims, 1);
```

Embedding images in background

Load the sub-image from the targets directory, use gdImageCopy() to position sub-image at some random position within the main puzzleImage.

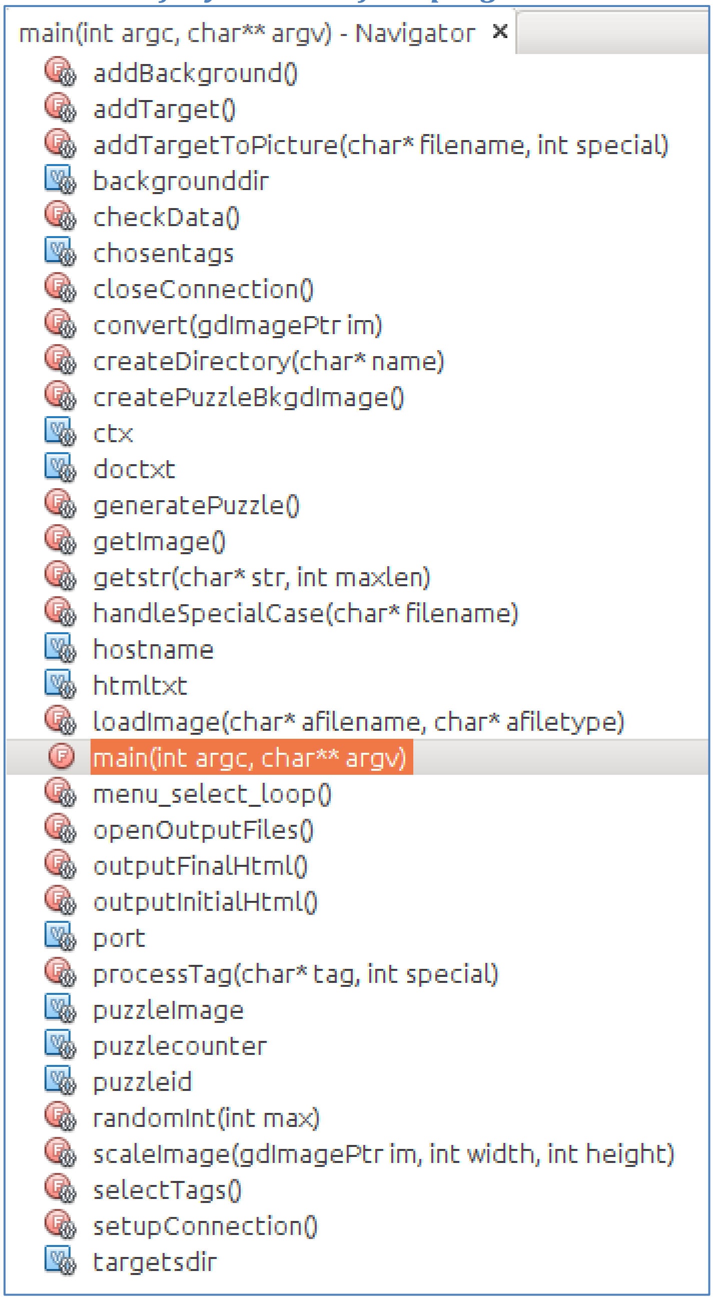
```
FILE* input = fopen(filename, "r");

gdImagePtr im = gdImageCreateFromPng(input);
fclose(input);

int height = gdImageSY(im);

int top = randomInt(490 - height);
int left = randomInt(590);

gdImageCopy(puzzleImage, im, left, top, 0, 0, 100, height);

gdImageDestroy(im);
```

Creating directories



```c
static void createDirectory(char* name) {
    int err;
    struct stat sb;

    err = stat(name, &sb);
    if (err == 0) {
        // Entry must exist, problems if it's not a directory
        if (!S_ISDIR(sb.st_mode)) {
            printf("%s exists but it's not a directory\n",name);
            sleep(1); // For Netbeans!
            exit(1);
        }
    } else {
        if (errno == ENOENT) {
            // No entry - so make the directory
            err = mkdir(name, S_IRWXU);
            if (err != 0) {
                printf("mkdir failed; errno=%d\n", errno);
                exit(1);
            }
        }
        else
        if(errno == ENOTDIR) {
            // Similar to earlier error - arises if "name" ends with '/'
            // (Linux realised that wanted a directory and so returns this error)
            printf("%s exists but it's not a directory\n",name);
            sleep(1); // For Netbeans!
            exit(1);
        }
        else {
            // Some bizarre problem
            perror("stat request error code");
            sleep(1); // For Netbeans!
            exit(1);
        }
    }
    // returns if directory existed or was successfully created
}
```

Structure of my version of the program



main(int argc, char** argv) - Navigator ×

```
addBackground()
addTarget()
addTargetToPicture(char* filename, int special)
backgrounddir
checkData()
chosentags
closeConnection()
convert(gdImagePtr im)
createDirectory(char* name)
createPuzzleBkgdImage()
ctx
doctxt
generatePuzzle()
getImage()
getstr(char* str, int maxlen)
handleSpecialCase(char* filename)
hostname
htmltxt
loadImage(char* afilename, char* afiletype)
main(int argc, char** argv)
menu_select_loop()
openOutputFiles()
outputFinalHtml()
outputInitialHtml()
port
processTag(char* tag, int special)
puzzleImage
puzzlecounter
puzzleid
randomInt(int max)
scaleImage(gdImagePtr im, int width, int height)
selectTags()
setupConnection()
targetsdir
```

Number of images

You will need a minimum of 3 different background images, and five different target sets (tag entries). For each target set, you will need at least five scaled, partially transparent images. Your program should refuse to generate HTML pages if there are too few images available.

Submission

Prepare your report and convert to PDF format as the file A4.pdf.

Submission is done electronically via a program called `turnin` that runs on "banshee" – the main CS undergraduate machine. You must first transfer your A4.pdf file to your home directory on banshee (this is different from the home directory that you access on the Linux machines). You can transfer the file using a SSH file-transfer program. The Ubuntu OS allows you to open a file-browser connected to your banshee home directory – and you can simply drag your A4.pdf file across using the visual file browser.

For CSCI131, assignments are submitted electronically via the `turnin` system. For this assignment you submit your assignment via the command:

```
turnin -c csci131 -a 4 A4.pdf
```

Late submissions would be submitted as:

```
turnin -c csci131 -a 4late A4.pdf
```

The program `turnin` only works when you are logged in to the main banshee undergraduate server machine. From an Ubuntu workstation in the lab, you must open a terminal session on the local machine, and then login to banshee via ssh and run the `turnin` program.

Marking

The assignment is worth 6 marks total.

- Appearance and structure of report: 1 mark
- Evidence for correct operation: 1 mark
- Code and explanations of your implementation (and also your makefile for your stand-alone version): 4 marks total