To compile the program just type “Make” in the directory. The executable will be called “prog” and can be run using “./prog”.

Graphs have been saved as .eps, but .png copies are included in case there are any problems with .eps files.

<u>Question (a):</u>

See file *metropolis.c*.
The function *demo()* will show my code running for cos(x) and x^2.

Question (b) gives $\langle\cos(x)\rangle = 0.7788007831$ and $\langle x^2\rangle = 0.5$.
I tested my implementation by comparing my results to these values.
The closest I was able to get to these given values was $\langle\cos(x)\rangle = 0.778848$ and $\langle x^2\rangle = 0.499957$ using the following parameters:
Delta = 2.4
Starting x point = 0
Number of iterations = 10000000
Thermalisation discard = 1000

I tried other values for delta but using 2.4 gave me the best results.

<u>Question (b):</u>

See the *cosx-history.eps*, *xsqr-history.eps* and *x-history.eps* files in the *graph/* directory.
The raw data for these graphs can found as *cosx-*.txt*, *xsqr-*.txt* and *x-history-*.txt* in the *data/* directory.
The script used to create these graphs was *history_graph.sh*
All of these graphs have lines for delta = 1.5, 15 and 150. The cos(x) and x^2 histories also have the expected value plotted. The title in the top right gives the colour key for each.
See the function *create_history_data()* (commented out) for how this data was generated.

By looking at *cosx-history.eps* and *xsqr-history.eps* you can clearly see the need for thermalisation as it takes some time for the values to stabilise. You can also see the effect that changing delta will have. When delta is large it leads to large errors for $f(x) = \cos(x)$ and $f(x) = x^2$.

In addition I also graphed the history of x itself with three different delta values, with a starting point of 10. This is the *x-history.eps* graph.

It’s obvious from looking at this graph that as *delta* becomes larger the acceptance rate decreases and the sampled values don’t change as often.

Question (c):

See the *delta_vs_acceptance.eps* graph in the *graph/* directory
The data used to generate this graph can be found as *data/delta_vs_acceptance_rate.txt*.
The script used to create this graph was *acceptance_graph.sh*.
See function *create_delta_vs_acceptance_data()* (commented out) for how this data was created.

The above-mentioned graph shows how the acceptance rate changes as delta changes. I noticed that the graph looks quite similar to a *y = 1 / x* graph, so I checked how accurate the function *acceptance rate = 1 / delta* is, and the results are below.

Delta	**Actual acceptance rate**	**Acceptance rate = 1 / delta**
0.5	0.8607	2
1	0.7291	1
1.5	0.6129	0.6666
2	0.5129	0.5
5	0.2251	0.2
10	0.113	0.1
25	0.0451	0.04
50	0.022	0.02
100	0.0113	0.01
150	0.00735	0.0066

While not precise I think for this choice distribution and with delta > 1.5 *acceptance rate = 1 / delta* gives an ok estimate of the acceptance rate as delta changes.

According to these[1][2] documents, the ideal acceptance rate for a one dimensional gaussian distribution (which is what g(x) is for this assignment) is around 40-45%.

[1] https://arxiv.org/pdf/1011.6217.pdf page 10
[2] https://web.as.uky.edu/statistics/users/pbreheny/701/S13/notes/2-28.pdf slide 16

As noted above for question (a), I obtained the best results using delta = 2.4. This gives an acceptance rate of around 44.8%, which is within the ideal range described in the previous paragraph.

<u>Question (d):</u>

See code in *variance.c*.

See graph *bin_size_vs_variance.eps* in the *graph/* directory.
The raw data for this graph can be found in *data/bin_size_vs_bin_variance.txt*.
The script used to create the graph is *bin_graph.sh*.
See function *variance_calulcations()* (commented out) for how this data was created.

I wrote code that follows the approach shown in the lectures, but I don't think I was able to do it correctly.

My code splits the data into bins (ignoring the part discarded due to thermalisation) and then calculates an average for each of these bins. The binned variance is then measured by:

- summing the square of each bin's variance minus the mean
- dividing this sum by (num_bins * (num_bins - 1))

I was unable to find an obvious plateau from my data. For cos(x) the binned variance seemed to stay in the range $2\text{x}10^{-8}$ to $3\text{x}10^{-8}$ regardless of the bin size.

Using my naive variance and my estimated bin variance with bin size = 100000 I obtained an integrated autocorrelation time of $2.144\text{x}10^{6}$ for cos(x) and $3.0367\text{x}10^{6}$ for x^2.

<u>Question (e):</u>

See file *gaussian_rv.c*.
See function *do_gaussian_rv()* (commented out).

Using gaussian RVs with a mean of 0 and a standard deviation of 0.70785 I was able to obtain the following two estimates:
cos(x): 0.778767
x^2: 0.500003

The values used for the standard deviation and mean of the distribution were found through experimentation. I found that with the above mean and standard deviation, the number of samples needed to be at least 10’000 to achieve a reasonably accurate result.

The Box-Muller transform was used to transform a uniform RV in range [0, 1] (generated by RANLUX) into a RV with a Gaussian distribution and the provided mean and standard deviation.