Project

This project is intended to improve your ability to solve a larger problem that requires more integration, planning and design than assignments have required to this point. This project must be a culmination of ideas across the entire course.

Larger projects require additional documentation in order to help guide the programmer in the implementation. For this project, a small Waterfall lifecycle will be used with five parts:

- Requirements and plan (what is being built)
- Design and small test cases (how the code is built)
- Implementation (most of the actual coding)
- Integration (combining individual components together) and Test (making sure it works as planned or how you know what was built is what is supposed to be built)
- Postmortem

The project is out of a possible 40 marks, with the following marking scheme:

1. Requirements & Planning Document (5 marks)
   - 1 – 2 page MS Word document
   - Describes what your project hopes to accomplish
   - Breaks down your project into activities with an approximate schedule and due dates
2. Design Document (5 marks)
   - 1 – 2 page MS Word document
   - Divides the project into an approximate grouping of files, functions and/or major data structures.
3. Testing Document (5 marks)
   - 1 – 2 page MS Word document
   - Indicates tests you performed on your code and results from your program
   - If you did not get your program to successfully run, this document becomes theoretical — that is, what you would have tested and potential results
   - Higher marks for this document are a reflection of more complete code coverage, ensuring that you test as many code paths and end conditions within your project as possible
4. Postmortem Document (5 marks)
   - 1 page MS Word document
   - Describe what went well, what got in the way, and what could have been improved
   - If you worked with a partner, evaluate your performance and your partner's performance. This is the only document where each partner submits a separate document.
5. Weekly Status Reports (5 marks)
   - A summary update **each week** that answers the questions:
     - What was done?
     - What got in the way?
     - What is planned for the following week?
   - Point form notes are fine.
6. Software Implementation (15 marks)
   - The source files and a project file (Codeblocks) for organization.
   - The 15 marks are divided up as follows
     a. 2 marks for structure and style
     b. 2 marks for documentation and comments
     c. 1 mark for successful completion of the project
     d. 10 marks for program execution, output and quality of the solution

Notice that the actual coding portion of this project is only worth 15 out of 40 marks (only 38%). In most software projects, the documentation is as important or even more important than the actual code, and this marking scheme reflects that.

Software Estimates and Project Scope

This project is expected to take approximately five weeks or 25 days. If the design and documentation takes 3 to 5 days and testing takes 5 days, this leaves a total of 15 days for implementation. Most programmers are able to write 20 to 50 lines of code each day. Assuming that you write 30 lines of code per day, this suggests a

project that would take approximately 30 * 15 = 450 lines of code. Ordinarily, software developers are wildly optimistic in the amount of code they can write. You may want to scale these numbers up or down as appropriate when considering estimates for your project scope.

During the planning phase, estimate your project based on the number of lines of code for each section, as that will give you a reasonable estimate of the amount of work required to complete the project.

Projects marked with an asterisk (*) indicate a very challenging project for any high school student.

Possible projects:

1. Image processing – start with read/write BMP or JPG files, then process the images in other ways – this is often a necessary sub-project for other projects
2. Animate a series of images to create a flipbook/simple movie renderer
3. Create a mosaic of images from a set of images
4. Image stenography (hiding messages inside images)
5. Model a physical system – inside of a cell, diffusion, flow, evolution, three-body problem, traffic systems, blood flow
6. Chaos in physical systems
7. DNA pattern matching
8. Simulate the Travelling Salesman Problem
9. Simulate a computer within a computer (core war game)
10. Algorithmic game mechanics – see http://www.squidi.net/three/
11. PDF generator from text and/or images
12. Swarms/Agents – nano-like creatures/habitats
13. Predator/prey systems and ecosystem modelling
14. Image matching / face recognition
15. Load balancing for messaging systems
16. Text compression – Huffman coding, LZW coding, DEFLATE
17. Image compression – fractals, wavelets, signal processing
18. Content generation
19. Create a neural net or fuzzy logic library – this needs to be tested within an application
20. Fractals – visualization for example, http://www.skytopia.com/project/fractal/mandelbulb.html
21. Maze generator – see for example www.cgl.uwaterloo.ca/~csk/projects/mazes where different shapes are used to outline the mazes
22. Biology or tree creation – L-systems or www.speedtree.com
23. Landscape creation – see www.planetside.co.uk/terragen/
24. Societal generation
25. Create an OpenGL ray tracer *
26. Numerically analyze a mathematical number theory problem such as the Collatz problem or multiple bases persistence
27. Create a compiler
28. Encrypt and decrypt a message using the RSA encoding scheme or other standard scheme – Elliptic Curve Cryptography, DES, AES, Vigner, etc. The encryption method should cover large number arithmetic.
29. BEDMAS parser of large numbers using trees or tokens
30. Create an artificial programming language such as a reduced version of C, C++ or a unique language created for this project *

31. Implement a standard game optimization algorithm (such as minmax) for Chess, checkers, go, 2048 or other game *
32. Pathfinding algorithm *
33. Minecraft clone or voxel world *
34. Implement a translator for a standard protocol such as HTML or XML *
35. Design and implement a simple phone app – since this is C++, limited to Android/Blackberry
36. Simulate Rubik's Cube solver
37. Create a scheduler (for student/teachers classes, messages, computer processes)
38. Text Analysis – such as determining the language of the text
39. Handwriting recognition
40. Tools to demonstrate program test completion
41. Creating/analyzing filesystems
42. Visualization tools – for example music, or datasets using geographic, business/economic or scientific data
43. Implementing Turing machines
44. Graph or tree data structure visualization
45. Voronoi Diagrams
46. Bin packing