-- Project Overview --

Your **final grade** in IB Computer Science comes from several individual grades:

Standard Level		**Higher Level**	
Paper 1 (Core Syllabus sec.1-4)	**45%**	**Paper 1 (Core Syllabus + HL ext.)**	**40%**
Paper 2 (Web Science Option)	**25%**	**Paper 2 (Web Science Option)**	**20%**
Project (Internal Assessment)	**30%**	**Project (Internal Assessment)**	**20%**
		Paper 3 (Case Study)	**20%**

The Internal Assessment Project consists of about **50 hours** work (maybe more). This involves analyzing a real problem, planning a solution, writing and debugging a Web page with Javascript automation to solve the problem, and documenting the entire process in writing and a short video.

-- Project Outline --

The Computer Science Internal Assessment project contains a Web-page(s), which solves a **real problem** or **realistic problem**. But the project requires **more** than just writing Java, and includes 5 stages of development and documentation:

- **Starting - Choose a Problem and a Client, Write Visual Prototype** ***Apr - June***
- **A = Analysis** - investigating and analyzing the problem - ***September***
  a detailed **analysis** of the **problem** leading to a **set of GOALS** that will guide the design and be used later (in section D) to **test** the success of the program
- **B = Design** - designing the solution and planning completion - ***October***
  a pre-programming **design** of a complete **solution** describing **data-structures, algorithms and modules** that will be included in the solution, as well as a timeline for completion
- **C = Development** - creating the solution, writing the Javascript - ***January***
  the documentation includes listings of important parts of the program(s) as well as screen-shots that show the results produced by the Web-page
- **D = Functionality and Extensibility of Solution** - ***February***
  - a video shows the proper functioning of the program
  - written documentation shows that the possibility to easily improve and extend the solution
- **E = Evaluation** - ***February***
  written evaluation of the success of the product, as well as suggestions for future improvements

The **finished project** is due on **Monday 2 March, 2018**. It is **all** submitted electronically in a single .zip archive, organized as prescribed by the IBO. Between **5 March and 16 March**, each student will be scheduled for a **20 minute interview**, to run and discuss the finished program.

-- Project Requirements -

The student must: select a **problem** and a **client (intended user)**; **analyze** the problem; **design** a solution**;** create a **Java program (solution)**; and **document** the solution fully. The solution must solve the problem sensibly, to the satisfaction of the Client, as well as demonstrating COMPLEX (ingenious) Java programming. Some **problems** are **not appropriate** (animated video games) and some **solutions** are **not appropriate** (for example avoiding loops, data-structures and methods) .

-- Examples --

IBO has provided sample Projects in the Teacher Support Material. Two of these samples are worth looking at - that does not mean they are "perfect", but the general concepts are useful:

Teacher Support Material IA Overview Example 8 : Computer Science for Kids in Java

Example 6 : Stock Control System in Java and MS Access

-- Choosing a Topic --

The most important considerations when choosing a topic are:

- the student **understands** the problem (don't make a chess program unless you play chess)
- the student **identifies** an intended **end-user** (this must be a real person - it could be another student, but is preferably an adult or at least a sophisticated student)
- there is sufficient **potential** for using complex programming techniques in Javascript
- the problem can be adequately solved using the **student's programming skills** and the **available hardware and software**

Keep in mind that you are **solving a problem -** you are not **building a system.** For example, you would not create a word-processor, which is a huge text-management system, but you might create a tool that takes a text file as input and analyzes the readability of the text by counting words and the sizes of words. Text-readability is a single function in a word-processor. That said, your problem must provide enough scope for some **sophisticated and complex** programming.

-- Things to Avoid --

The following are forbidden and/or discouraged:

- **collaboration** - students must do the work alone - especially writing the program - "teamwork" is **not permitted**
- **animated graphics** - although not actually forbidden, animation cannot be documented on paper so it causes problems and is usually much more difficult than anticipated. Thus, a video game is not a sensible topic.
- **copying Javascript code -** if Javascript code is copied, this must be **clearly documented** in the program listing, and the student will receive **no credit** for that part of the program. For example, a copied method that sorts an array would **not** be given credit for demonstrating **complexity**. If standard solutions are downloaded, they should be used for **clearly separate functions** than the parts the student is programming.
- **writing the program first -** although a **prototype** is a required part of the analysis and design stages, thorough PLANNING must be completed before starting the final programming stage
- **writing for yourself as the only intended user** - the program might be useful to the author, but there must still be **a separate end-user** who is involved in the analysis and testing stages. This is not actually forbidden by IBO regulations, but it is a very bad idea and will probably lead to a poor result.

Project Topic Areas - The Good, the Bad and the Painful

- **Databases** (boring and time-consuming, but straightforward)
  Library circulation, Contacts (telephone numbers and addresses), Inventory, Video rentals, Sports, personal calendar, colleges DB, CAS activities DB - many possibilities around school. ** Note that database problems require data to be stored in files, which is not a topic required in the 2014 syllabus - but you still need to do it.**

- **Graphics Displays** (usually unrealistic, results too simple, programming too difficult)
  Banner ads, slide-shows, photo effects
  ** Note that graphics commands are not part of the IB syllabus, so only choose a graphics oriented program if you are willing to learn these commands on your own **

- **Games** (fun but challenging)
  Board-games like tic-tac-toe, Minesweeper, checkers Mastermind, Battleship, 4 in a row, Sudoku. Gambing games. Some ideas at: http://www.mazeworks.com/home.htm

- **Edutainment** (reasonably easy) e.g. http://www.myvocabulary.com/
  on-screen quizzes, interactive puzzles, educational question banks, etc

- **Text-File Processing** (tricky but interesting)
  Mail-merge, Spell-check, File format conversion (HTML to text), Language translator, Index creator, Web-page builder, extended search/replace, templates, macro text-insertion

- **Simulations** (tricky but interesting)
  Queuing simulation, Gambling simulation, Physics experiments (pendulum, linear motion and momentum), pond ecology

- **Mathematics** (rather difficult especially if you don't like math)
  Calculator, Drawing graphs (difficult), Geometry, Statistics, Charts, Financial calculations, Mortgages, Calculating prices for pizzas with various toppings (too simple?)

- **Tools/Utilities** (generally too difficult)
  Compiler/Interpreter, Compression, Encryption, Printer control and/or layout

- **Network-based** (tricky and non-standard, disaster-prone)
  chat, e-mail, web-site crawler (searcher), online multi-player games

- **Resource Management** (generally quite difficult - most students don't understand these)
  Train-route planner, PERT/CPM analysis, Scheduling classes, Balance an airplane's freight load, Distribute seats in an airplane

- **Multimedia** (probably **too** difficult and/or inappropriate**)
  music-track editor, animation builder, video editor, video game

- **Real-Time** (probably **too** difficult and/or inappropriate**)
  Robot control, Sensors and switches (temperature sensors and light switches), Traffic lights, Elevators, Burglar alarms

Many other ideas are possible and acceptable. But check with the teacher first, so you can avoid wasting time on a problem that will be too difficult or unsuitable.

Forbidden - any problem that was presented as a sample project by the teacher or the IBO

Finding a Suitable Problem

Candidates must ensure that the problem provides scope for fulfilling the expectations and requirements. If you are unsure, **ASK THE TEACHER** before starting to work on the problem.

A **data-base** or **text-parsing** problem generally provides the most straightforward way to meet the IA requirements and criteria. There is an obvious need for sorting and searching, as well as storing data in objects and arrays. However, many successful dossiers have been based on other types of problems.

If a non-database problem is chosen, teachers can help the students find sensible, appropriate uses for loading and/or saving data. A few ideas:

- **Saving results** - A simulation or math program can store results in a data-file for later analysis. This is especially sensible if the program creates large quantities of results, such as a simulation producing output every minute for a 24 hour period, or a graphing program generating hundreds of coordinates.
- **Keyword Validation** - Many input operations require a word to be typed which is in a list of "valid" entries - for example, names of months, names of geometric figures, etc. These could be stored in a data-file, along with associated information. In the case of geometric figures, there may be accompanying information such as a definition, formula for the area, etc. The validation routine might permit on-line additions to the keywords, like a spell-check which allows additions to the dictionary.
- **Conversion/Translation** - A program could be "internationalized" by allowing constants and instructions to appear in a variety of languages. Other programs may benefit from conversions of scientific units, money units, etc, where conversions are stored in a file.
- **On-Line Help/Instructions** - Most programs can benefit from some on-line help. This could be stored in a data-file, with help/instruction text stored together with key-words to identify the topic(s).
- **Log-File** - In a tutorial/on-line quiz situation, store results, best times, best scores, etc. Print error-messages and warnings into a log-file instead of on the screen. Store passwords and ID numbers in a file. In each of these, the resulting data-file must be manipulated and used in a meaningful way - e.g. a students' progress in a tutorial system is monitored and difficulty level raised or lowered to match their achievement level.

At HL, a similar concept (supplementary use) applies to linked-lists and/or trees. Teachers can help the students identify suitable, sensible uses for dynamic data structures. The candidates are not required to select problems or solutions that use dynamic data structures, but this is a straightforward way to demonstrate **complexity** an it reinforces some syllabus topics.

Further Suggestions for Avoiding Difficulties

Sample Data - Wherever data-files or data-lists are used, documentation should contain complete, annotated **listings of the contents** of some sample data-files, so the examiner (and/or teacher) can see the effects of various operations, and see that the operations have functioned correctly. Some sample data-files might be quite large - e.g. hundreds of records or more. These need not be printed in their entirety, but they do make it easier to understand why some operations are sensible (e.g. why a binary search is better than a sequential search, or why a search operation saves time), as well as providing a more realistic testing environment.

Realistic Problems - Although not strictly required, a "real" or "realistic" problem provides more opportunities for candidates to do some investigation and design, and eases the task of working with the intended user. Realistic problems make many parts of the documentation process easier.

Familiar Topics - Candidates need to understand the problem that they are solving. For this reason, many candidates choose problems that exist in school or in areas with which they have considerable familiarity. Some examples:

- Library circulation and customer data-base
- On-line quizzes/tutorials for school subjects (math, science, English, etc) - edutainment
- Board games and language games
- Student grades, attendance, schedules
- School bus routes and rider data-base
- Travel/vacations - exchanging money, dates and times problems, airline reservations
- Retail stores - video lending, inventory, sales, ordering
- Entertainment - TV guide, movies, CD's
- Sports - statistics, planning tournaments
- Parents' business (varies with the student)

-- Interesting Problems (Danger!) --

Many "interesting" problems, such as E-mail, playing chess, video games, graphics displays can be unsuitable for the following reasons:

- The programming is actually too difficult (which is the same reason it is interesting)
- Interesting problems can be very difficult to limit and/or clearly define
- They may be quite difficult to document properly

Interesting problems certainly increase motivation. However, teachers should provide guidance when the students are selecting topics, to ensure that the topic has reasonable scope for complete coverage of the required elements, and that the problem is not too difficult. In many cases it is sufficient to help the student find a reasonable way to **limit the scope** of the solution.

-- Limited Solutions --

High school students **won't succeed** in making **general-purpose commercial software** similar to a professional word-processing program, as they have neither the time nor the necessary skills. It is better to address a **very specific** and **limited** problem. For example, rather than writing a word-processor, the student could create a program that analyzes essays by counting words and producing an **index** and a list of **overused** words. Even if this is only usable on text-files (not .doc files), it fills a need that might be missing from the professional word-processor, and is a problem of roughly the correct size and complexity for this project.

== Some Ideas for SL and/or HL ==

Database

The student is asked to write a data-base management system. This has traditionally been a very common choice for portfolios in the old syllabus, but is probably less sensible in the new course where file storage commands are not part of the course. Any of the following are appropriate topic areas: phone-book (or contact manager), recipes, dictionary, student courses and grades, teacher grade-book, school bus routes and riders, on-line multiple-choice quiz, inventory, payroll, etc.

Web-Page Creator

This can be a template-based, wizard driven web-page creator. It needn't provide full editing features. It is just a quick solution for somebody wanting a straightforward web-page. A good solution would meet the specific needs of the intended user. For example, schools have various events like concerts, presentations, and trips that need to be announced. The intended user might be an activity supervisor like the band director. Such a user will have a pretty clear set of standard information they want included. Simplicity is the goal here - not enormous flexibility.

Digital Camera Album Creator

This might assume the user has stored their photos in a single folder on the hard-disk. Then it takes all those photos and creates an HTML file presenting the photos in a simple layout, linked to an enlarged view when the user clicks. It probably wants to include the ability to add annotations to the photos.

Student and/or Faculty ID Cards

This is a data-base oriented problem which is a bit more exciting than most, as it can involve graphics (photographs) and passwords. (Photographs will only be appropriate if the programming language and the school's hardware support this feature.) It could also involve the students in the graphical layout of the ID card - solving the problem of squeezing the picture and all necessary information onto the card, as well as making it look nice by using interesting fonts and colors. A very good solution might allow the user to customize their own card with their own choice of fonts and colors. Students must not get carried away with the DTP aspects of the problem - they still need to completely demonstrate data-file and pointer skills.

E-mail List Manager

Manage various e-mail lists for various groups. This is especially useful in a school, club, or other large organization which sends regular announcements to various groups of people. This will be a much better solution if some automation is provided, such as actually **sending** e-mails. Good error-handling should prevent accidental mistakes like choosing many groups accidentally. Students should be cautioned that their tests should **not** generate actual spam and annoyance.

Calendar Publisher

Collects events in a database - this part can be quite simple and straightforward. A good solution would then produce outputs in various formats - weekly calendar, monthly calendar, screen or paper versions, alphabetically sorted. The solution will be better if it requires minimal input from users and produces maximum outputs.

== Specific Ideas for HL ==

Stock-Market Prices Analysis

Stock market investors want to analyze historical data (past few weeks or months) as a basis for choosing stocks to buy and sell. Typically the data would be captured from a nicely formatted text file, but that might not permit HL students to demonstrate mastery of file operations. Students are inclined to generate random sample data rather than using real data. However, they should be encouraged to find a source for real data and thus produce a more realistic, usable solution.

Board Games and Puzzles (not video action games)

The knight's tour, 8 queens, and other "traditional" board-game puzzles provide stimulus for trees, stacks, and linked-lists for an exhaustive search for a solution. The student must find an appropriate need for data-files - otherwise they will have difficulty achieving the 100% mastery factor. For example, chess playing programs store an "opening book" in a large data-file. A program which allows users to solve the 8 queens problem might store all the known solutions in a data-file, or collect successful solutions produced by users.

Simulations (Queuing, Physics Experiments, etc)

Students should simulate something with which they are already thoroughly familiar. For example, a billiard-ball simulation should only be attempted by a student with sufficient mathematics and physics knowledge. As with other non-data-base projects, it may be difficult to find a sensible need for data-files here. Students may tend to produce very inflexible solutions using many iterative calculations and techniques. They should be guided to use pointers to produce more flexible, robust solutions.

Mail-Merge

This problem requires 3 distinct modules - creating a form letter, creating a data-base, and then merging the two. A student should not attempt to create all three of these modules - typically a standard word-processor can be used for writing the form letter, so the student need not program a word-processor or text-editor. The data-base management functions are quite straightforward but students will probably want to construct a specific record type (e.g. Name,Phone,Address) rather than allowing a general, flexible, user-modifiable record structure. The merging process presents a good opportunity for using pointers. Teachers may need to guide students in limiting their solutions so that the problem does not become unmanageable. For example, merging into a text-file is sufficient - the student need not program a merge into a proprietary word-processing file format such as MSWord or WordPerfect. The project could be designed to use student and teacher addresses that are already available in a data-file somewhere, but the candidate still needs to program some data-file management features (deleting records, sorting, searching).

Math Calculator/Grapher

This is only appropriate for a student with high ability or high interest in mathematics. Plotting fractals, 2-D and 3-D object rotations and transformations, random walk, line of best fit, etc are problems which are not normally handled by standard calculators and software, and thus motivate the student's efforts. There is considerable scope for using pointers here. However, it is easy to get carried away in the mathematics and graphics and forget about the need to demonstrate mastery of data-file operations. Suggestions for sensible inclusion of data-files appear elsewhere.

Intended End-User

There must be a **specific intended end-user.** This must be a real person (or people) **other than the author** (Mulkey's requirement). This requirement causes some discomfort for many IB Computer Science students, so they try to avoid it. Students must overcome their discomfort early and engage in **productive discussions** with the intended user during the analysis and design stages. If done correctly, this makes the programming job easier. Avoid the following:

- Don't say "my program is for **everyone**." That makes the problem very difficult to define, and even more difficult to solve. Choose **one specific user** for discussions - you may wish to think about a group of users, but this should be a small group. For example, "all teachers" is a bad choice, but "several teachers in the math department" is a reasonable choice. "All students" is a bad choice, but "some of the IB Diploma candidates at my school" is a reasonable choice.

- Think about the **intended user** when **choosing the problem**. Making a "personal calendar" is different for a businessman than an elementary school student. The intended user has a significant effect on the problem definition and analysis, the design, and eventually the choice of the problem.

- Let the user **help** you - talk to them regularly. If you are in the middle of writing the program and trying to decide how a specific interface should look, a brief conversation with the user might reveal that they are happy with a very simple interface, or that they actually don't need that feature at all, thus saving time and energy for the programmer.

- **Take notes** every time you are talking to the user. Otherwise, you must keep everything in your head. More likely, you will simply forget and ignore some of the useful ideas that came from discussions with the user.

- Find a **sophisticated user** - one who knows something about computers and uses them often. Otherwise, they will either have lots of impossible ideas (e.g. "I want to use a microphone to talk to the computer"), or even worse they might have no ideas at all (e.g. "I don't care, anything's okay").

Concentrate on the Problem

During this project you should be **building a solution for a problem** - NOT **inventing a problem to match a solution.** Nevertheless, you must choose a problem that **CAN** be solved by writing a Web-page with Javascript. So you will probably start with an **idea** based on some Javascript page that you have already seen - that doesn't mean you are finished before you start. **Careful analysis** of the **problem** will reveal lots of issues you might not think of right away. In the end the success of your project will be assessed against the **GOALS** you set in the beginning. You should:

- **Investigate** the problem thoroughly and carefully

- **Be creative** when thinking about the **goals**

- **Discuss** the goals with the **user** and **reach agreement**

- Make the list of goals **as clear and concise as possible** without sacrificing functionality Concentrate on some clear **benefit** for the user - something that makes their life easier

- **Don't** add **extra (cool) features** if they are not actually needed – this only adds more work without improving the solution (and probably leads to lower assessment)

Some Clever Technical Ideas

Here are some suggestions that might keep your project from being boring.

Standard (boring?)	***More Interesting***
Text based pages	Use GUI controls
Printing results in text mode	Displaying formatted HTML
Your program runs all alone, and you must code everything (re-invent the wheel)	Your program executes other standard applications to use as tools, and you code only the really important algorithms
Typing in lots of data	Data-mining in the Web, CD-ROMS, or other existing data-bases
Type inputs	Select inputs from a Choice box, or by clicking a button, or selecting a menu item
The program is always the same	User can change options/settings/preferences and these are stored and used next time
Your program only speaks English	Prompts and error messages are retrieved from a text-file, so **internationalization** is possible
Pop-up help dialog	**HTML** help file with display button
Event-driven or menu-driven, so user decides what happens next	**Wizards** guide the user through the program, so it is easier (user-friendly)
User presses lots of buttons, types in lots of data, gets a few results	User presses few buttons, types a little data, and **automation** produces LOTS of results

You can see some of these programming techniques demonstrated in this example:
http://ibcomp.fis.edu/projects/mathtools/mathtools.html

Here are some programming ideas that might help you achieve better **complexity** marks.

Simple (poorer)	***Flexible (better)***
Program has exact values (numbers and Strings) written in the code	Values are flexible (changeable) and are saved to disk and reloaded next time
Single variables (A,B,C)	Arrays (num[0], num[1], num[2])
Arrays have a fixed length	Arrays with flexible length - ending with a sentinel or coupled with a SIZE variable
Parallel arrays	Arrays of objects
Parse a text-file made with Notepad	Parse an HTML file loaded from a web-site
Put everything in one single array	Use multiple arrays, encapsulate Data Structures in ADT objects
Methods/functions	Flexible, re-usable methods with **parameters**
Use each class only once	Re-use classes
Verify inputs with "are you sure?"	Validate inputs by range checks or format checks
Program crashes when user makes a mistake	Error-handling rejects bad-data and prevents crashes

== More Topic Suggestions ==

Here are some suggested topics for FIS students. Many of them are for me, so I would be the intended end-user. The descriptions were written for the old Dossier in the previous syllabus. They must probably be shortened or simplified to fit the Project in the new syllabus.

Please ignore any comments that refer to "mastery factors".

- **IB Math Calculator** - includes functions for various IB math topics, like standard-deviation, solving a quadratic, multiplying matrices, etc. The basic idea is similar to MatheAss (http://www.matheass.de/) but should focus on problems that are in the IB math syllabus. It need not be nearly as long as MatheAss.

- **IB (and other) Review Questions Database -** provide a database where teachers can deposit review and practice questions, and students can read and try the questions.

- **General Drill and Practice** - teachers can type drill & practice worksheets quickly. When students view the WS, they can choose to do a fill-in-the-blanks exercise, as demonstrated in this web-page: Life-Cycle . The program should also be able to produce printed or web-page versions of the worksheet.

Applications (DB = database oriented)

- **Web-Addresses DB** - rather than a list of links on a web-page, or the lists I have in my bookmarks, I'd like a nice searchable DB of web-sites, with http addresses, description of contents. Should include key-words. It would be nice if these could be extracted automatically from the web-page itself (or Google description) and then stored in the DB

- **Web-Site generator** - should be VERY EASY to use, for inexperienced and impatient teachers, or very young students. Needn't be flexible - should be template or wizard based, not using an HTML editor. Can start with one simple template (e.g. navigation left, title top, content right). Add other templates later (maybe). Should produce pages out of a data-base or file(s)
- **CAS projects DB** - stores list of students with areas of interest, plus list of projects with similar keywords, and tries to match students with projects. Also keeps a record of currently active projects, so the CAS coordinator can keep tabs on what students are doing.
- **Worksheets Manager** - A DB for teachers to keep track of and manage their worksheets and other teaching materials. Should be able to click on a file name and add it to the DB, together with keywords and description. Should enable teacher to combine questions from several worksheets and produce a new worksheet
- **Friends Tracker DB -** Keep track of friends' e-mail addresses, ICQ name, Handy number, and other contact info. Should be able to merge lists from several friends together, so they can share. This is probably an SL project.
- **Knowledge Exchange (Volunteers)** - a database of e-mail addresses (and possibly phone numbers) for students, teachers and parents. Each record includes a list of interests/hobbies/keywords identifying issues where this person is willing and able to answer questions. For example, a French speaker might volunteer to do short translations for other people. A math teacher might volunteer to help students with homework. A hobby cook might volunteer to share recipes. The application should be able to search for a topic (e.g. French) and provide a list of matching e-mail addresses that can easily be copied into an e-mail address field.

- **IT Software DB** - Our school has a large collection of software – both local applications and online apps. Links are stored but without any documentation. We have a few lists of required software for various installations - the labs are different from the library, and those are different from the teachers' machines. A good support database would have records of the names of software titles, where they are stored, notes about installation tricks, as well as lists of software to be installed for specific machine types.
- **IT Tips and Tricks** - a database of FAQs, hints, instructions, tips and tricks, etc. - for teachers, students, and parents. The database should include **keywords** for each record, and provide both keyword and full-text searching. It would also be nice if it could produce HTML pages as output.
- **Software Trainer (Tutorial MAKER)** - used by a teacher to create a set of instructional screens to lead someone through the use of a program. For example, a set of screens showing how to use Paint-Shop-Pro to make a transparent logo and save it in a GIF file. YOU (the programmer) are not supposed to make the lessons. Rather, you make a tool so that it is easy for a teacher (or student) to make and save lessons. This should be quick and easy to use so a non-expert can make lessons - e.g. a math teacher could make lessons about using a graphing program. Avoid fancy multi-media stuff - simply some text and a single screen-shot picture on each page is sufficient.

Data Mining

This is a general concept, referring to finding data sources on the web and "refining" these to produce more specific, useful information. For example, the CIA factbook has lots and lots of data about countries, but it is probably too much data for most purposes. A program could present smaller, more useful subsets of the data. It could also offer the ability to do clever searching and sorting. Some possible data-mining areas are:

- **CIA Factbook -** https://www.cia.gov/cia/publications/factbook/index.html
- **Dictionaries –** http://freedict.org/en/
- **US Census Bureau –** http://www.census.gov/
- **Social Studies Data -** http://www.icpsr.umich.edu/org/index.html
- **USDA Example - Farmers Computer Usage** - http://usda.mannlib.cornell.edu/usda/usda.html

There are many other possibilities. In some cases you will find data embedded in HTML files or structured text files. Then in addition to putting a nicer, more usable interface on to of the data, you also need to extract the data by **parsing** the file. Here are a few examples:

- **Project Gutenberg** (free online books) – http://www.gutenberg.org/
- **Forbes Lists** (mostly about money) – http://www.forbes.com/lists/
- **Wikipedia Lists** - http://en.wikipedia.org/wiki/List_of_reference_tables
- **Internet Usage Stats** - http://www.internetworldstats.com/stats.htm

Educational (for students)

- **Vocabulary Trainer** - maintains a list of vocabulary words. Presents them either randomly or according to some plan. Keeps track of missed words, so student can work on those again later. Student can add new words themselves, or merge an entire list from the teacher (or other students)
- **Drill and Practice (especially math)** - timed practice on math facts, including calculations. Questions can come from a long list of simple problems, or be generated randomly according to rules
- **Multiple Choice Quiz Generator** - stores a DB of questions. Allows teacher to generate a multiple choice quiz, in a variety of ways: printed, on-line, on-line random order. Would be nice if it could include 1 picture for each record, but that should simply be a text input telling the path and name of the file (could use file-open command to retrieve). Online needs to give a score, but NOT store this in a central file.
- **Nations Intro DB** - stores basic info about various nations, the kind of info a casual tourist might want before traveling there. For example, capital city, time-zone, tourist attractions, money exchange rate, etc. This should include some web-links for further info.

Games

- **Memory Game** (SL) - turn cards over and find matching cards. This could be based on vocabulary practice where a word must be matched with it's definition, or matching a word with a picture.
- **Hangman** (or other word-guessing) - Selects a random word from a list in a file and lets the user guess. Word-list maintenance is the significant part of the program, not the game. Could have various lists of words on different topics.
- **Text Adventure** - move from room to room, fight monsters, collect weapons, etc. Scenario (rooms and rules) must be stored in a file, so it is possible to change the game or to make various different games. Use pointer references (a web) to store connections
- **Checkers** (or other board game) - more appropriate for SL. Should include the ability to remember the sequence of moves and save these in a file and replay the game later.
- **Concentration** - a TV game show where a large picture is hidden behind cards. The cards are in pairs and must be turned over two at a time and matched (like memory). Players win prizes for the items they match, and a big prize figuring out the message in the hidden picture. The program should choose a picture at random and mix up the pairs of cards randomly.

Wizard Managed Projects

Some computer users need to do tasks using several pieces of software and various other resources. They might only do the task one time (or once a year), so they might not know what they need to do. They want a "wizard" to guide them through the process. Applying to colleges is a good example:

- **College Applications Wizard -** helps senior students manage their college applications. This will include web-links and advice from counselors, as well as connections to automatically start needed software - like a word-processor to type a personal essay, or running the UCAS online registration system. The important part about this application is that the **counselors** must be able to modify the information (instructions) easily, and create **profiles** for different types of students - e.g. applying to US colleges involves different steps than applying to the UK. The program will be a success if it makes it easier for the counselors and students to keep their work organized. It might also include check lists so the counselors can easily track students' progress. You definitely want to talk to a counselor before starting this - if the counselors are not interested, don't bother with the project.

Math Stuff

- **Math Problem Trainer** - this works by making a **template** math problem (e.g. some text with numbers, but the numbers are changeable). Then a teacher can make: (1) lots of similar problems; (2) making a template for solutions; (3) produce on-line quizzes.
- **High Level Math Calculator** - does complex mathematical calculations, like adding up sequences, solving for pieces of triangles, statistics (standard deviation, etc), solve simultaneous equations, matrix calculations, etc. This must be solving math **problems**, not doing simple calculations like sines and cosines - like the "fancy" stuff in your calculator, but make it a bit easier to use and produce some nicer output. Each type of calculation can be hard-coded (programmed), but must allow the user to input numbers. This project is only appropriate for an HL math student - but you could be doing calculations that would help a Math Studies or younger students. Have a look at this: http://www.matheass.de/matheass_en/index.htm, but make something simpler. Perhaps create collections of tools according to lessons in math classes (e.g. sine rule, cosine rule, bearings in one group).