CAB402 Programming Paradigms
Assignment 1

Due: 18th April 2016
Worth: 30%

According to the theory of Evolution, all living organisms on earth evolved (through a process of mutation and natural selection) from a single first living organism[1]. This assignment is based on a heavily simplified computer simulation of that evolutionary process.

You are to submit three separate implementations of the same application:

1. Implemented using F# (in a strictly pure functional style).
2. Implemented using F# (using mutable state to create a more efficient implementation).
3. Implemented using C# in a strict object-oriented style.

You will also write a **1-2 page report** discussing your experience and comparing three approaches. Your comparison should include an evaluation of each of the following:

(i) the efficiency and effectiveness of the three approaches,
(ii) which of the approaches was easier to program,
(iii) which code is easier to read, understand and maintain,
(iv) which of the resulting programs is more concise, and
(v) which produces more efficient code

Where possible, you should support your arguments with quantitative data.

Your application will take as input a (provided) text file that contains a trace/log of a sequence of evolutionary events that you are to model. Each line of the file contains a separate evolutionary event and these events must be applied in order from first/top line to last/bottom line. Each species is identified by an integer id, with the first species given a species id of 1. Each species has a distinct genome - the genetic material or DNA associated with the organism. Your application must track the genome for each species as evolution proceeds. For the sake of this exercise, we are only interested in the sections of DNA that directly control the form and function of the organisms. We call these sections of DNA *genes*. Each gene consists of a sequence of DNA (or nucleotides), which we represent computationally as character strings over an alphabet of size 4: A for Adenine, C for Cytosine, G for Guanine and T for Thymine. For our purposes, each species therefore has a genome that consists of a set of genes, with each gene represented as a string of characters. Each gene of a species is given an integer id to identify it. This gene id is only unique within a given species; i.e. gene 1 of species 1 is different from gene 1 of species 2 and so on.

[11] For those interested, see Wikipedia, or alternatively this compendious site at Berkeley: http://evolution.berkeley.edu/evolibrary/article/evo_01

The simulation always begins with a world containing only a single species (with a species id of 1) and the genome of this species initially has no genes. The first few lines of the trace/log file specify the initial set of genes for species 1. e.g.:

```
create,1,1,ACTAAGCAGGAAAGCCCTTACACGCATAACCGTTTGAGGTCCGGTCTTACC
create,1,2,CTTGATACAATAAGCCTATAGCCTGGTTGAACAACACATG
create,1,3,GAAGCTAAGTAATGCGACCACCAATAAAGCCTCGGATCCGTCCCCTGCGCTTGGGCGTCGA
```

Each line contains a comma separated list of values. The first value is always the name of the event; in this example, there are 3 creation events. For a creation event, the second value is the species id, the third value is the gene id and the fourth value is the DNA for that new gene.
After applying those 3 events, we would have a world, still containing only 1 species, but that species would now have 3 genes (with gene Ids 1, 2 and 3).

The other types of events are all different kinds of mutations that alter existing genes in various ways:

A single nucleotide polymorphism (abbreviated SNP, pronounced snip) is where a single nucleotide (character) within an existing gene is replaced by a different nucleotide. E.g.:

`snip,1,2,19,C,A`

*before SE1_G2: CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
*after SE1_G2: CTTGATACAATAAGCCTAT**A**GCCTGGTTGAACAACACATG*

In gene 2 of species 1, change the 19th nucleotide from Cytosine (C) to Adenine (A).
(Note: when referring to positions within genes, the first nucleotide is considered position 0, though this is a computational convention rather than a biological custom.)

An insertion is where a new sequence of DNA is inserted within an existing gene. E.g.:

`insert,7,77,5,ACTG`

*before SE7_G77: CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
*after SE7_G77: CTTGA**ACTG**TACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*

In gene 77 of species 7, insert the DNA sequence ACTG at position 5.

A deletion is where a section of DNA is removed from an existing gene. E.g.:

`delete,5,129,29,7`

*before SE5_G129: CTTGATACAATAAGCCTAT**C**GCCTGGTTG**AACAACA**CATG*
*after SE5_G129: CTTGATACAATAAGCCTAT**C**GCCTGGTTGCATG*

In gene 129 of species 5, delete the section of DNA at position 29 of length 7.

Duplication is where a new gene is added to an existing species that is an exact copy of another gene within that species. E.g.:

duplicate,1,12,7

*before SE1_G7 : CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
before SE1_G12: non-existent
*after SE1_G7 : CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
*after SE1_G12: CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*

Add a new gene with id 12 to species 1 by duplicating gene 7 (of species 1).

Loss is where an existing gene is lost/removed from an existing species. E.g.:

loss,1,6

*before SE1_G6: CTTGATACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
after SE1_G6: non-existent

Remove gene 6 from species 1.

Fission is where two genes are created by splitting an existing gene into 2 parts. E.g.:

fission,1,167,120,6

*before SE1_G120: CTTGAT|ACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*
before SE1_G167: non-existent
after SE1_G120: CTTGAT
*after SE1_G167: ACAATAAGCCTAT**C**GCCTGGTTGAACAACACATG*

Gene 120 of species 1 is split at position 6 to create two new genes. The first part of the old gene (up to position 6) replaces the old gene with id 120 and the second part of the old gene (after position 6) becomes a new gene with id 167.

Fusion is where a new gene is created by fusing (or concatenating) together two existing genes. E.g.:

fusion,9,146,113

before SE9_G146: AAACCATAGTGACCGCGGAGAGCAACA
before SE9_G113: TGAATACCGTAGATAAAACAGT
after SE9_G146: AAACCATAGTGACCGCGGAGAGCAACATGAATACCGTAGATAAAACAGT
after SE9_G113: non-existent

A new Gene is formed by fusing gene 146 followed by gene 113 of species 9. This new gene is given the old gene id 146 and the old gene with id 113 is removed from the species.

Speciation is a special event that occurs when so many mutations have occurred that the new organism is so different that it can no longer be regarded as belonging to the same species. It is the mechanism by which new species are created. For the sake of modelling in this application, the speciation event will simply create a new species that initially will have the exact same set of genes as an existing species. Such a speciation event will typically be followed by a sequence of mutation events that then render the new species different from the original species. E.g.:

speciation,2,1

before SE1_G1: ACTAAGCAGGAAAGCCCTTACACGCATAACCGTTTGAGGTCCGGTCTT
before SE1_G2: CTTGATACAATAAGCCTATAGCCTGGTTGAACAACACATG
before SE1_G3: GAAGCTAAGTAATGCGACCACCAATAAAGCCTCGGATCCGTCCC
after SE1_G1: ACTAAGCAGGAAAGCCCTTACACGCATAACCGTTTGAGGTCCGGTCTT
after SE1_G2: CTTGATACAATAAGCCTATAGCCTGGTTGAACAACACATG
after SE1_G3: GAAGCTAAGTAATGCGACCACCAATAAAGCCTCGGATCCGTCCC
after SE2_G1: ACTAAGCAGGAAAGCCCTTACACGCATAACCGTTTGAGGTCCGGTCTT
after SE2_G2: CTTGATACAATAAGCCTATAGCCTGGTTGAACAACACATG
after SE2_G3: GAAGCTAAGTAATGCGACCACCAATAAAGCCTCGGATCCGTCCC

A new species (with id 2) is created by copying all of the existing genes of species 1.

Output

Once all of the events listed in the trace/log file have been applied, you must save all of the resulting genes (of all species) to a specially formatted text file (called FASTA file format):
E.g. output file: test10.fa:

```
>SE1_G4
AAACCATAGTGACCGCGGAGAGCAACA
>SE1_G79
AAAACTGATGGACTGCCCAGCATGAC
>SE1_G114
TGAATACCGTAGATAAAACAGT
>SE2_G4
TCCGACACGGGTGCCAAGATATT
>SE2_G16
GATTGGCCGCCGAGAGTCACG
...
```

Two lines are used to output each gene. The first line which must start with a > character contains the name of the gene. E.g. SE1_G4 is the name of gene 4 of species 1. The second line contains the characters representing the DNA of that gene. The output should be ordered by species id and within each species, ordered by gene id.

Homologous Genes (Challenge Exercise)

In order to get top marks, in addition to tracking the current DNA sequence of each gene, you must also track where each segment of that gene originated from. Two genes are referred to as *Homologous* if at least part of those genes evolved from a common section of DNA. New/novel sections of DNA are injected into the system in only two types of evolutionary events: in creation events when the world is first formed and in insertion events at later times. Note: snip events are not regarded as introducing new DNA. All other mutation events simply rearrange and combine those novel sections of DNA in various ways. Every segment of every gene has an *origin* that can therefore be traced back to exactly one of these novel sections of DNA. Different segments of a gene may have different origins, so in general, a gene may have multiple origins. In order for two genes to be regarded as homologous, it is not sufficient for them to have a shared origin – there must also be some overlap in the exact segment of that origin from which they have evolved. If two genes share an overlapping section of novel DNA then they are regarded as homologous. Note, that if the exact same sequence of DNA nucleotides is introduced (by mere chance) via two different creation or insertion events, then they are not considered to have the same origin.

Appendix 1 shows an example of how the origin of each segment of each gene can be tracked. There are only 3 novel sections of DNA introduced in this example (the two creation events and the one insertion event). If we consider for example SE2_G4, we see that its evolutionary history involved all three of those novel sections of DNA, but in its final form, all of the DNA that remains in the final form of SE2_G4 came only from the light green segment of the DNA introduced in **`create,1,2`** event. So, SE2_G4 would be regarded as homologous to SE1_G3 because a part of both derived from that same section of DNA, however, SE2_G2 would not be regarded as homologous to SE2_G1 (see table in Appendix 1).

Second Output File

In addition to outputting the genes in a FASTA file format, you must also save the homologous relationships in a separate file:
E.g. output file: test10.homologs:

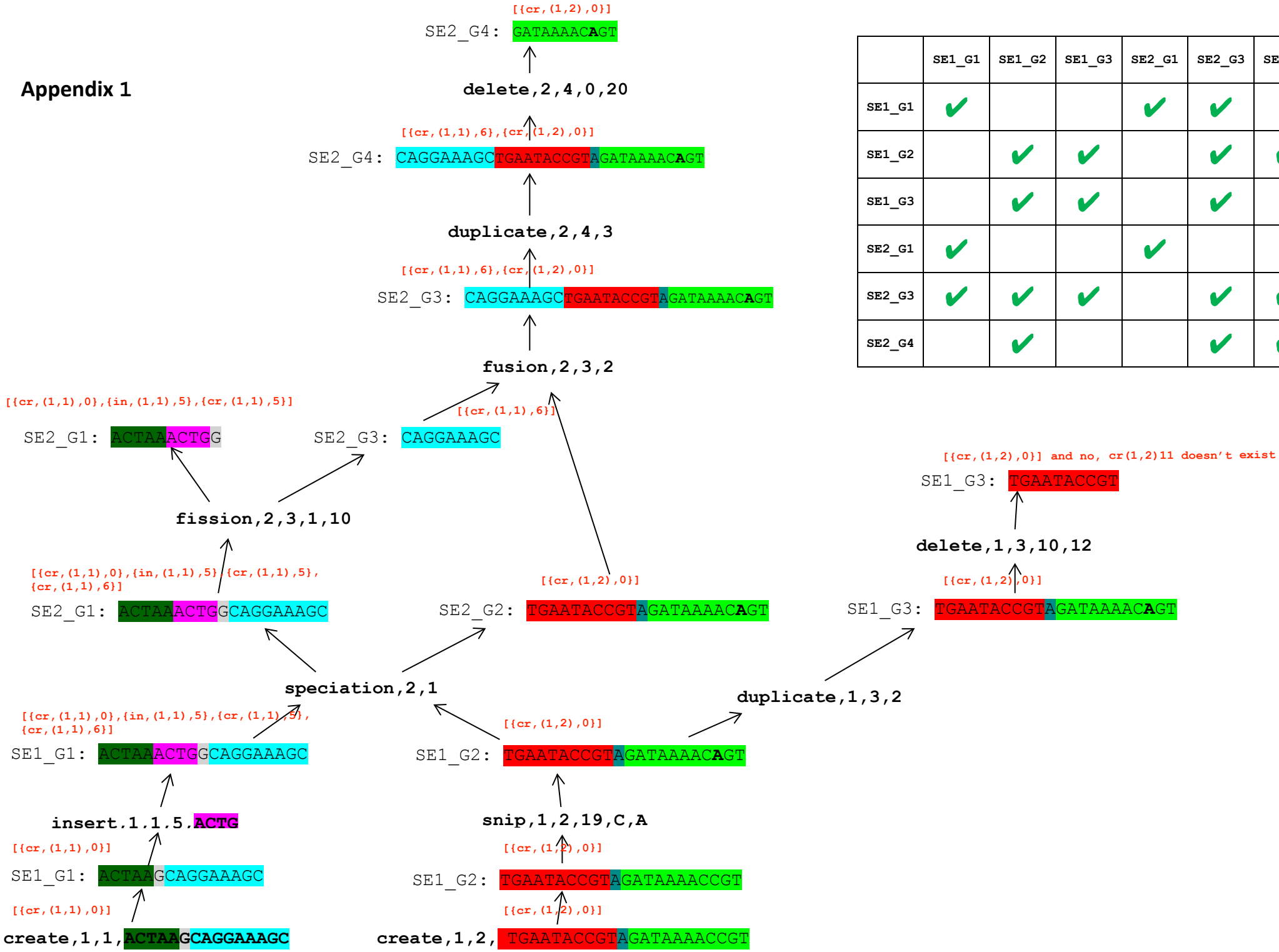
```
SE1_G4: SE1_G4 SE1_G79 SE1_G143 SE3_G167
SE1_G5: SE1_G5
SE1_G22: SE1_G22 SE1_G34 SE1_G52 SE1_G60 SE1_G117 SE1_G153 SE3_G140 SE7_G47 SE8_G113
SE1_G34: SE1_G22 SE1_G34 SE1_G52 SE1_G60 SE3_G140 SE7_G47 SE8_G113 SE9_G146
SE1_G52: SE1_G22 SE1_G34 SE1_G52 SE1_G60 SE1_G117 SE1_G153 SE3_G140 SE7_G47 SE8_G113
SE1_G60: SE1_G22 SE1_G34 SE1_G52 SE1_G60 SE1_G117 SE1_G153 SE3_G140 SE7_G47 SE8_G113
SE1_G79: SE1_G4 SE1_G79 SE1_G167
SE1_G114: SE1_G114 SE1_G143
...
```

This tells us that gene 4 of species 1 is homologous to itself (SE1_G4) and to gene 79 of species 1 and gene 143 of species 1 and gene 167 of species 3 …

Coding Hints for F# Implementation

- **`Map.add`** method can be used in strictly pure functional implementation because it doesn't really add a new item to an existing map, it creates a new Map which is the same as the old map except that it contains a new key/value.
- Use **`fold`** function to apply a sequence of operations while maintaining the state of the changing world in an "accumulator".
- Use **`assert`** to test expected preconditions for your functions

Appendix 1



	SE1_G1	SE1_G2	SE1_G3	SE2_G1	SE2_G3	SE2_G4
SE1_G1	✔			✔	✔	
SE1_G2		✔	✔		✔	✔
SE1_G3		✔	✔		✔	
SE2_G1	✔			✔		
SE2_G3	✔	✔	✔		✔	✔
SE2_G4		✔			✔	✔