```
#!/usr/bin/perl -w
use strict;

=head1 storableDemonstration.pl

=head2 Distribution Terms

This version written by Michael J. Moorhouse, Erasmus MC and distributed under the
Creative Commons 'Attribution-Share Alike 3.0'
See: http://creativecommons.org/licenses/by-sa/3.0/

(Summary: do [mostly] what you want with it - other than forget who wrote the original.
Allow other people to do the same with your version of the code as you do;
don't blame me if it causes damage)



=head2 Summary

This program demonstrates the use of the 'Storable' and 'XML::Simple' module
(if available) to store / recover hash structures to / from files.

These modules offer two ways to take / create simple hash arrays in memory and
store / retrieve them from a file.

The hash arrays are then inspected (printed!) using the Data::Dumper module and using
conventional hash access techniques.  Probably then it is also a demonstration of
Data::Dumper as well.

It is assumed that Data::Dumper and Storable are installed on the system as they
are 'core' modules.

=head2 Usage

The program is self contained - just run it and enjoy the output.

Files called 'myXML.xml' & 'myFile.hash' are created in the current directory
(hence write permission is needed).  These are both small (less than 300 bytes)
but are not deleted for the purposes of demonstration.

=cut

#Example of how to adjust the module loading path in Perl:

use lib "/home/bioinf/michael/install/XML-Simple/lib/perl5/site_perl/5.8.5";

#Load Core Module(s):
use Storable;
use Data::Dumper;
#This is not-so-core; hence query first:
eval ("use XML::Simple");
#Check XML::Simple module is installed:
#Keep a little optimism here...(0 will prevent sections of code running)
my $XMLSimple =1;

unless ($@)         #Optimism (misplaced) might result in an error.
    {#Whey!  We have the module installed and working...
    use XML::Simple;
    print "XML::Simple reports ready!\n";
    }
else
    {
```

```
63:     $XMLSimple =0; warn "XML::Simple was not detected - check the 'use lib \"/some
    location\" path if you think it should be\n";
64:     }
65:
66: =head1 storableDemonstration.pl
67:
68: It creates a small file on disk 'myFile.hash' that is essentially a binary dump of
69: the hash contents.  This is not deleted afterwards - for the purposes of demonstration
70: and is small 119 bytes
71: If produced the XML file (myXML.xml) is 170 bytes and not deleted afterwards.
72:
73:
74: =head2 Expected Output
75:
76: This is more or less fixed as the hash contents are defined in code though the actual
77: order of elements in the hash arrays and XML are arbitary:
78:
79:     [michael@bioinf-dual07 readcounting]$ ./storableDemonstration.pl
80:     XML::Simple reports ready!
81:     Populating Hash array
82:     Hash data loaded as:
83:     $VAR1 = {
84:               'Second' => {
85:                             'U0' => 20,
86:                             'U1' => 21
87:                           },
88:               'First' => {
89:                             'U0' => 1,
90:                             'U1' => 1,
91:                             'Run_Date' => '2009-05-01',
92:                             'U2' => 7
93:                           }
94:             };
95:     --------
96:     Dumping to file (Storable.pm)
97:     Readloading from file (using Storable.pm)
98:     Results of reloaded hash:
99:     FirstKey        ,       2ndKey  =       Value
100:    First   ,       Run_Date         =      2009-05-01
101:    First   ,       U0       =      1
102:    First   ,       U1       =      1
103:    First   ,       U2       =      7
104:    Second  ,       U0       =      20
105:    Second  ,       U1       =      21
106:    --------
107:    Deleting $OtherHashRef
108:    --------
109:    Starting the output of the hash in XML
110:    Contents of XML file are:
111:    '<opt>
112:       <First>
113:         <Run_Date>2009-05-01</Run_Date>
114:         <U0>1</U0>
115:         <U1>1</U1>
116:         <U2>7</U2>
117:       </First>
118:       <Second>
119:         <U0>20</U0>
120:         <U1>21</U1>
121:       </Second>
122:     </opt>
123:    '
```

```
124:     Reloading XML from file
125:     Results of reloaded hash: (access code already demonstrated so just Data::Dumper
     used here:
126:     $VAR1 = {
127:                'Second' => {
128:                             'U0' => '20',
129:                             'U1' => '21'
130:                           },
131:                'First' => {
132:                             'U0' => '1',
133:                             'U1' => '1',
134:                             'U2' => '7',
135:                             'Run_Date' => '2009-05-01'
136:                           }
137:              };
138:
139:
140:
141: =cut
142:
143: #Create and Populate the hash:
144:
145: print "Populating Hash array\n";
146:
147: my %Hash;
148: $Hash {"First"}{"U0"}++;
149: $Hash {"First"}{"U1"}++;
150: $Hash {"First"}{"U2"}=7;
151: $Hash {"First"}{"Run_Date"} = "2009-05-01";
152: $Hash {"Second"}{"U0"}=20;
153: $Hash {"Second"}{"U1"}=21;
154: print "Hash data loaded as:\n";
155: print Dumper (\%Hash);
156: print "--------\n";
157:
158: #Dump to file 'myFile.hash';
159: print "Dumping to file (Storable.pm)\n";
160:
161: store \%Hash, 'myFile.hash';
162:
163: print "Readloading from file (using Storable.pm)\n";
164:
165: #(re)Load the contents of the disk file into another hash:
166: my $OtherHashRef = retrieve ("myFile.hash");
167:
168: #Print these out: (note sytax as 'retrieve' returns a reference to a hash:
169: print "Results of reloaded hash:\nFirstKey\t,\t2ndKey\t=\tValue\n";
170: foreach my $TopKey (sort keys %{$OtherHashRef})
171:     {
172:     #print "D: TopKey='$TopKey'\n";
173:     foreach my $ndKey (sort keys %{$$OtherHashRef{$TopKey}})
174:         {
175:         print "$TopKey\t,\t$ndKey\t =\t",$$OtherHashRef{$TopKey}{$ndKey},"\n";
176:         }
177:     }
178:
179: print "--------\n";
180: print "Deleting \$OtherHashRef\n";
181:
182: #Run this section if XML::Simple was found.
183: if ($XMLSimple ==1)
184: {
```

```
185: undef $OtherHashRef;
186: print "--------\nStarting the output of the hash in XML\n";
187:
188: #Here the "'NoAttr' => 1" option means that any XML 'attributes' get converted to tags
189: #(Not entirely sure how this works in practice though - how are attributes marked
190: #in hash arrays to start with?)
191:
192: my $XML = XMLout (\%Hash, 'NoAttr' => 1);
193: open XMLOUT, ">myXML.xml" or die "Cannot open 'myXML.xml' for output\n";
194: print XMLOUT $XML;
195: close XMLOUT;
196:
197: my @Results = `cat myXML.xml`;
198: print "Contents of XML file are:\n'@Results'\n";
199: print "Reloading XML from file\n";
200: my $ReloadedXML = XMLin('myXML.xml');
201:
202:
203: print "Results of reloaded hash: (access code already demonstrated so just Data::Dumper
     used here:\n";
204: print Dumper ($ReloadedXML);
205: print "\n";
206: }
```