1. Introduction.

EGGSHELL

Workbench for Analysing Egg Summary Files

by John Walker
http://www.fourmilab.ch/


```
#define REVDATE "27th May 2001"
```

2. Program global context.

```
#include "config.h"      /* System-dependent configuration */
  ⟨ Preprocessor definitions ⟩
  ⟨ System include files 3 ⟩
  ⟨ Application include files 4 ⟩
  ⟨ Global variables 5 ⟩
  ⟨ Class definitions 6 ⟩
```

3. We include the following POSIX-standard C library files. Conditionals based on a probe of the system by the `configure` program allow us to cope with the peculiarities of specific systems.

```
⟨ System include files 3 ⟩ ≡        /* C++ include files */
#include <iostream>
#include <fstream>
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <map>
  using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <math.h>
#include <assert.h>
#ifdef HAVE_STRING_H
#include <string.h>
#else
#ifdef HAVE_STRINGS_H
#include <strings.h>
#endif
#endif
#ifdef HAVE_GETOPT
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif
#else
#include "getopt.h"      /* No system getopt—use our own */
#endif
```

This code is used in section 2.

4. The following include files provide access to external components of the program not defined herein.

```
⟨ Application include files 4 ⟩ ≡
#include "statlib.h"      /* Probability and statistics library */
#include "timedate.h"      /* Time and date utilities */
#include "eggdata.h"      /* Egg database utilities */
```

This code is used in section 2.

5. These variables are global to all procedures; many are used as "hidden arguments" to functions in order to simplify calling sequences.

⟨ Global variables 5 ⟩ ≡

```
typedef unsigned char byte;      /* Byte type */
static FILE *fi;      /* Input file */
static FILE *fo;      /* Output file */
static bool errcheck = true;      /* Check for errors ? */
```

This code is used in section 2.

6. The following classes are defined and their implementations provided.

⟨ Class definitions 6 ⟩ ≡

This code is used in section 2.

7. Utility functions.

8. Procedure *usage* prints how-to-call information.

```
static void usage(void)
{
  cout ≪ PRODUCT ≪ "␣␣--␣␣Analyse␣eggsummary␣files.␣␣Call:\n";
  cout ≪ "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" ≪ PRODUCT ≪ "␣[options]␣[infile]␣[outfile]\n";
  cout ≪ "\n";
  cout ≪ "Options:\n";
  cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--copyright␣␣␣␣␣␣␣Print␣copyright␣information\n";
  cout ≪ "␣␣␣␣␣␣␣␣␣␣␣-n,␣--noerrcheck␣␣Ignore␣errors\n";
  cout ≪ "␣␣␣␣␣␣␣␣␣␣␣-u,␣--help␣␣␣␣␣␣␣␣Print␣this␣message\n";
  cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--version␣␣␣␣␣␣␣␣␣Print␣version␣number\n";
  cout ≪ "\n";
  cout ≪ "by␣John␣Walker\n";
  cout ≪ "http://www.fourmilab.ch/\n";
}
```

9. Main program.

```
int main(int argc, char *argv[])
{
  extern char *optarg;      /* Imported from getopt */
  extern int optind;
  try {
    int f, opt;
    char *cp;
    fi = stdin;
    fo = stdout;
    ⟨ Process command-line options 10 ⟩;
    ⟨ Process command-line arguments 11 ⟩;
    string s;
    systemtime td, td1;
    td.now();
    cout << td.dateToString() << " " << td.timeToString() << "\n";
    td1 = td;
    td1.nextDay();
    cout << "Tomorrow: " << td1.dateToString() << " " << td1.timeToString() << "\n";
    td1 = td;
    td1.previousDay();
    cout << "Yesterday: " << td1.dateToString() << " " << td1.timeToString() << "\n";
    td1 = td;
    td1.dayStep(7);
    cout << "Next week: " << td1.dateToString() << " " << td1.timeToString() << "\n";
    td.fromString("1981-04-12 13:51:18");
    cout << td.dateToString() << " " << td.timeToString() << "\n";
    td.fromString("2021-04-12");
    cout << td.dateToString() << " " << td.timeToString() << "\n";
    cout << "Weekday = " << systemtime::jwday(systemtime::utctoj(2001, 5, 12, 23, 59, 59)) << "\n";
    td.fromJulian(2436116.31);
    cout << td.dateToString() << " " << td.timeToString() << "\n";
    cout << "To Julian = " << td.toJulian() << " Weekday = " << td.weekday() << "\n";
    td.fromString("1987-04-10");
    cout << "Sidereal time = " << td.siderealTime() << "\n";
    cout.flush();
    int t1, t2, t3;
    systemtime::jyear(2436116.31, &t1, &t2, &t3);
    printf("%04d-%02d-%02d\n", t1, t2, t3);
    systemtime::jhms(systemtime::utctoj(1989, 11, 14, 16, 23, 18), &t1, &t2, &t3);
    printf("%02d:%02d:%02d\n", t1, t2, t3);
    eggdatabases edb;
    cout << "Pathname for GCP is " << edb.path("gcp") << "\n";
#if 0
    td.fromString("Bob");
    cout << td.dateToString() << " " << td.timeToString() << "\n";
#endif      /* Statistical analyses tests */
    {
      normalDistribution nd(100, 7.07);
      cout << "Normal dist: mu = " << nd.get_mu() << " sigma = " << nd.get_sigma() <<
          " variance = " << nd.variance() << "\n";
```

```
    cout ≪ "␣␣P␣=␣" ≪ nd.CDF_P(110) ≪ "␣Q␣=␣" ≪ nd.CDF_Q(110) ≪ "\n";
    nd.writeParameters(cout);
    nd.approximateBinomial(200);
    cout ≪ "Normal␣dist:␣mu␣=␣" ≪ nd.get_mu( ) ≪ "␣sigma␣=␣" ≪ nd.get_sigma( ) ≪
        "␣variance␣=␣" ≪ nd.variance( ) ≪ "\n";
    cout ≪ "␣␣P␣=␣" ≪ nd.CDF_P(110) ≪ "␣Q␣=␣" ≪ nd.CDF_Q(110) ≪ "␣␣z␣=␣" ≪
        nd.z_score(110) ≪ "\n";
  }
  {
    chiSquareDistribution xd(32);
    xd.writeParameters(cout);
    cout ≪ "Chi-square␣dist:␣k␣=␣" ≪ xd.get_k( ) ≪ "\n";
    cout ≪ "␣␣P␣=␣" ≪ xd.CDF_P(36) ≪ "␣Q␣=␣" ≪ xd.CDF_Q(36) ≪ "\n";
  }
  {
    binomialDistribution bd(200, 0.5);
    cout ≪ "Binomial␣dist:␣n␣=␣" ≪ bd.get_n( ) ≪ "␣r␣=␣" ≪ bd.get_r( ) ≪ "␣variance␣=␣" ≪
        bd.variance( ) ≪ "\n";
    cout ≪ "␣␣P␣=␣" ≪ bd.CDF_P(110) ≪ "␣Q␣=␣" ≪ bd.CDF_Q(110) ≪ "␣␣z␣=␣" ≪
        bd.z_score(110) ≪ "\n";
    bd.writeParameters(cout);
  }
  cout.flush( );
#if 0    /* Eggsummary file I/O tests */
  eggsummary es;
#if 0
  ifstream ci("../test.csv");
  es.load_from_CSV(ci);
  ci.close( );
#endif
  es.load_from_CSV("../test.csv.gz");
  for (int j = 0; j < es.eggs_reporting; j++) {
    cout ≪ es.egg_number[j] ≪ ":␣" ≪ (int) es.get_egg_index(86399, j) ≪ "\n";
  }
#endif
#if 1    /* Eggsummary cache tests */
  {
    int j;
    eggsummary * es;
    eggsummary_cache ec;
    es = ec.get("../test.csv.gz");
    eggsummary & esr = *es;
    for (j = 0; j < esr.eggs_reporting; j++) {
      cout ≪ esr.egg_number[j] ≪ ":␣" ≪ (int) esr.get_egg_index(86399, j) ≪ "\n";
    }    /* Once more to test cache */
    es = ec.get("../test.csv.gz");
    esr = *es;
    for (j = 55860; j < (55860 + 60); j++) {
      cout ≪ (j - 55860) ≪ "␣=␣" ≪ ((int) esr.get_egg_number(j,
          37)) ≪ "␣" ≪ (esr.isSampleValid(j, 37) ? "" : "(Bad)") ≪ endl;
    }
```

```
        esr.exclude_bad_data("rotten_egg.csv");
        esr.limit_to_range(70, 130, &cout);
        for (j = 0; j < esr.eggs_reporting; j++) {
          cout << esr.egg_number[j] << ": " << (int) esr.get_egg_index(86399, j) << "\n";
        }
        for (j = 55860; j < (55860 + 60); j++) {
          cout << (j - 55860) << "\t=\t" << ((int) esr.get_egg_number(j,
              37)) << "\t" << (esr.isSampleValid(j, 37) ? "" : "(Bad)") << endl;
        }
      }
#endif
#if 0    /* Interactively test CSV parsing */
      while ((cout << "-->\t"), getline(cin, s)) {
        csv_parserp(s);
        string f;
        cout << s;
        cout << "\n";
        while (p.next_field(f)) {
          cout << "{" + f + "}\n";
        }
      }
#endif
    }
    catch(exception &e)
    {
      cout << "Blooie!!!\t\tException\tpopped:\t" << e.what( ) << endl;
#ifndef CORE_DUMP
#ifdef STACK_TRACE
      char s[160];
      sprintf(s,
          "/bin/echo\t'where\nq'\t>/tmp/gdbcmd\t;\tgdb\t\t-batch\t--command\t"/tmp/gdbcmd\t%s\t%d",
          argv[0], getpid( ));
      system(s);
      sleep(5);
#endif
      throw;     /* Re-throw exception to dump core */
#endif
    }
    return 0;
  }
```

10. We use *getopt* to process command line options. This permits aggregation of options without arguments and both `-d`*arg* and `-d` *arg* syntax.

```
⟨ Process command-line options 10 ⟩ ≡
  while ((opt = getopt(argc, argv, "nu-:")) ≠ −1) {
    switch (opt) {
    case 'n':    /* -n Suppress error checking */
      errcheck = false;
      break;
    case 'u':    /* -u Print how-to-call information */
      case '?': usage( );
      return 0;
    case '-':    /* -- Extended options */
      switch (optarg[0]) {
      case 'c':    /* --copyright */
        cout ≪ "This␣program␣is␣in␣the␣public␣domain.\n";
        return 0;
      case 'h':    /* --help */
        usage( );
        return 0;
      case 'n':    /* --noerrcheck */
        errcheck = false;
        break;
      case 'v':    /* --version */
        cout ≪ PRODUCT ≪ "␣" ≪ VERSION ≪ "\n";
        cout ≪ "Last␣revised:␣" ≪ REVDATE ≪ "\n";
        cout ≪ "The␣latest␣version␣is␣always␣available\n";
        cout ≪ "at␣http://www.fourmilab.ch/eggtools/eggshell\n";
        return 0;
      }
    }
  }
```

This code is used in section 9.

11. This code is executed after *getopt* has completed parsing command line options. At this point the external variable *optind* in *getopt* contains the index of the first argument in the *argv*[ ] array.

```
⟨ Process command-line arguments 11 ⟩ ≡
  f = 0;
  for ( ; optind < argc; optind++) {
    cp = argv[optind];
    switch (f) {
    case 0:
      if (strcmp(cp, "-") ≠ 0) {
        if ((fi = fopen(cp, "r")) ≡ Λ) {
          cerr ≪ "Cannot␣open␣input␣file␣" ≪ cp ≪ "\n";
          return 2;
        }
      }
      f++;
      break;
    case 1:
      if (strcmp(cp, "-") ≠ 0) {
        if ((fo = fopen(cp, "w")) ≡ Λ) {
          cerr ≪ "Cannot␣open␣output␣file␣" ≪ cp ≪ "\n";
          return 2;
        }
      }
      f++;
      break;
    default: cerr ≪ "Too␣many␣file␣names␣specified.\n";
      usage( );
      return 2;
    }
  }
```

This code is used in section 9.

12. Index. The following is a cross-reference table for `eggshell`. Single-character identifiers are not indexed, nor are reserved words. Underlined entries indicate where an identifier was declared.

⟨ Application include files 4 ⟩ Used in section 2.
⟨ Class definitions 6 ⟩ Used in section 2.
⟨ Global variables 5 ⟩ Used in section 2.
⟨ Process command-line arguments 11 ⟩ Used in section 9.
⟨ Process command-line options 10 ⟩ Used in section 9.
⟨ System include files 3 ⟩ Used in section 2.

EGGSHELL