

```
// RestGen.cpp - REST BREAK PROFILE DATABASE GENERATOR
```

```
//
```

```
// MODULE INDEX
```

```
// NAME          CONTENTS
```

```
// GenRand      Generate a random number in a given range
```

```
// Normal       Calculate area under the normal distribution
```

```
// InvNorm      Inverse normal function
```

```
// RandomTime   Randomise a shift start or end time
```

```
// main         Main line
```

```
//
```

```
// MAINTENANCE HISTORY
```

```
// DATE         PROGRAMMER AND DETAILS
```

```
// 02-09-11 JS  Original
```

```
//
```

```
//-----
```

```
#include <cstring>          // C-style string manipulation functions
```

```
#include <cctype>           // C-style character typing functions
```

```
#include <cstdlib>         // C-style standard library
```

```
#include <cmath>           // C-style Mathematical functions
```

```
#include <iostream>        // C++ input/output streams
```

```
using namespace std;      // Expand the standard namespace
```

```
#include "dtime.h"        // Double time declarations
```

```
exec sql include sqlca;   // Include SQL communications area
```

```
//-----
```

```
// DEFINITIONS
```

```
static const char  FIRST_DATE[] = "2011-10-01";
```

```
    // First date
```

```
static const char  LAST_DATE[] = "2011-10-31";
```

```
    // Last date
```

```
static const size_t LANE_CNT = 5;
```

```
    // Number of checkout lanes
```

```
static const double STD_DEV = 120.0;
```

```
    // Standard deviation of shift changeover time
```

```
//-----
```

```
// NOMINAL SHIFT CHANGEOVER TIMES
```

```
static const char *SHIFT_ARR[][LANE_CNT] = {  
    {"08:00", "08:20", "08:40", "09:00", "09:20"},  
    {"09:40", "10:00", "10:20", "10:40", "11:00"},  
    {"11:20", "11:40", "12:00", "12:20", "12:40"},  
    {"13:00", "13:20", "13:40", "14:00", "14:20"},  
    {"14:40", "15:00", "15:20", "15:40", "16:00"},  
    {"16:20", "16:40", "17:00", "17:20", "17:40"},  
    {"18:00", "18:20", "18:40", "19:00", "19:20"},  
};
```

```
static const size_t SHIFT_CNT = sizeof(SHIFT_ARR) / sizeof(SHIFT_ARR[0]) - 1;
```

```
//-----
```

```
// CASHIER/SHIFT ASSIGNMENTS
```

```
static const char *CASHIER_ARR[][LANE_CNT] = {  
    {"ALICE", "BARBY", "CINDY", "DIANA", "ELSIE"},
```

```

{"FARAH", "ALICE", "BARBY", "CINDY", "DIANA"},
{"ELSIE", "FARAH", "ALICE", "BARBY", "CINDY"},
{"DIANA", "ELSIE", "FARAH", "ALICE", "BARBY"},
{"CINDY", "DIANA", "ELSIE", "FARAH", "ALICE"},
{"BARBY", "CINDY", "DIANA", "ELSIE", "FARAH"},
{"ALICE", "BARBY", "CINDY", "DIANA", "ELSIE"},
{"FARAH", "ALICE", "BARBY", "CINDY", "DIANA"},
};

```

```
//-----
```

```
// GENERATE A RANDOM NUMBER IN A GIVEN RANGE
```

```

double
GenRand (
    double    minVal,    // Minimum value
    double    maxVal)    // Maximum value
{
    double    r;        // Random value

    modf (minVal + (maxVal - minVal + 1) * static_cast<double>(rand())
        / (static_cast<double>(RAND_MAX) + 1), &r);
    return r;
}

```

```
//-----
```

```
// CALCULATE AREA UNDER THE NORMAL DISTRIBUTION
```

```

double
Normal (
    double    x)        // X in standard deviations
{
    // This algorithm is adapted from "Evaluating the Normal
    // Distribution", George Marsaglia, Florida State University,
    // Journal of Statistical Software, July 2005, Volume 11, Issue 4.

    int i, j = static_cast <int> (0.5*(fabs(x)+1));
    static const long double R[9]= {
        1.25331413731550025L, .421369229288054473L,
        .236652382913560671L, .162377660896867462L,
        .123131963257932296L, .0990285964717319214L,
        .0827662865013691773L, .0710695805388521071L,
        .0622586659950261958L
    };
    if (j >= 9) {
        if (x <= 0) return 0.0;
        return 1.0;
    }
    long double pwr=1, a=R[j], z=2*j, b=a*z-1, h=fabs(x)-z, s=a+h*b, t=a, q=h*h;

    for (i = 2; s != t && i < 1024; i += 2) {
        a = (a+z*b)/i;
        b = (b+z*a)/(i+1);
        pwr *= q;
        s = (t = s)+pwr*(a+h*b);
    }
    if (i >= 1024) abort ();
    s = s*exp(-.5*x*x-.91893853320467274178L);
}

```

```

    if (x <= 0) return static_cast <double> (s);
    return static_cast <double> (1.0-s);
}

```

```

//-----

```

```

// INVERSE NORMAL FUNCTION

```

```

double
InvNorm (
    double      s)          // Area to the left of x
{
    double      h, l, x;    // Binary search variables
    int         i;         // General purpose index

    h = 9.0;
    l = -9.0;
    for (i = 0; i < 32; i++) {
        x = (h + l) / 2;
        if (Normal(x) > s)
            h = x;
        else
            l = x;
    }
    return x;
}

```

```

//-----

```

```

// RANDOMISE A SHIFT START OR END TIME

```

```

Dtime_t
RandomTime (
    Dtime_t     curDate,    // Current date
    const char *nomTime)   // Nominal time string
{
    int         yr, mo, dy; // Date components
    int         hr, mi, se; // Time components
    const char *p;         // Decoding pointer
    Dtime_t     randTime;  // Randomised time

    // Generate the nominal shift start time

    UnpackDtime (curDate, &yr, &mo, &dy, &hr, &mi, &se);
    p = nomTime;
    hr = 0;
    while (isdigit(*p)) hr = hr * 10 + *p++ - '0';
    if (*p++ != ':') abort ();
    mi = 0;
    while (isdigit(*p)) mi = mi * 10 + *p++ - '0';
    if (*p != '\0') abort ();
    se = 0;

    // Adjust the nominal shift start time for the random variation

    modf (PackDtime (yr, mo, dy, hr, mi, se) +
        STD_DEV * InvNorm(GenRand(1.0, 9999.0)/10000.0), &randTime);
    return randTime;
}

```

```
//-----
```

```
// MAIN LINE
```

```
int
main ()
{
    Dtime_t    firstDate; // Begin date
    Dtime_t    lastDate;  // End date
    Dtime_t    curDate;   // Current date
    Dtime_t    curTime;   // Current time
    size_t     lane;      // Lane number
    size_t     i;         // General purpose index

    exec sql begin declare section;
        char    shiftDate[10+1]; // Shift date
        char    shiftOpenTime[26+1]; // Shift open time
        char    shiftCloseTime[26+1]; // Shift close time
        char    cashierCode[8+1]; // Cashier code
    exec sql end declare section;

    // Connect to the database

    exec sql connect to restdb;

    // Drop tables

    exec sql whenever sqlerror continue;
    exec sql drop table shifts;
    exec sql commit work;

    // Jump to DbError whenever an SQL error occurs

    exec sql whenever sqlerror goto DbError;

    // Create the database tables

    exec sql create table shifts (
        shiftDate    date not null,
        shiftOpenTime    timestamp not null,
        shiftCloseTime    timestamp not null,
        cashierCode    char(8) not null
    );

    // Initialise the random number generator

    srand (20360L);

    // Process each trading day

    firstDate = DtimeFromSql (FIRST_DATE);
    lastDate = DtimeFromSql (LAST_DATE);
    for (
        curDate = firstDate;
        curDate <= lastDate;
        curDate += 24.0*60.0*60.0
    ) {
        // Process each lane
```

```

for (lane = 0; lane < LANE_CNT; lane++) {

    // Generate the nominal shift start time

    curTime = RandomTime (curDate, SHIFT_ARR[0][lane]);

    // Process each shift

    for (i = 0; i < SHIFT_CNT; i++) {
        SqlDateFromDtime (shiftDate, curDate);
        SqlTimestampFromDtime (shiftOpenTime, curTime);
        curTime = RandomTime (curDate,
            SHIFT_ARR[i+1][lane]);
        SqlTimestampFromDtime (shiftCloseTime, curTime);
        strcpy (cashierCode, CASHIER_ARR[i][lane]);
        exec sql insert into shifts (
            shiftDate,
            shiftOpenTime, shiftCloseTime,
            cashierCode
        ) values (
            :shiftDate,
            :shiftOpenTime, :shiftCloseTime,
            :cashierCode
        );
    }
}

// Commit one day at a time to avoid filling the
// rollback log

exec sql commit work;
}

// And that's all

return 0;

// Process database errors

```

```

DbError:
    cerr << "Error: SQLCODE=" << SQLCODE << '\n';
    return 1;
}

```