```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <random>

#define TRIALS 100000ul

static const double sqrt2 = std::sqrt(2.);
static const double sqrt2_PI = std::sqrt(2.*M_PI);
static const double sqrt2_div_PI = std::sqrt(2./M_PI);
static const double four_minus_PI_div_2 = 0.5*(4.-M_PI);
static const double two_PI = 2.*M_PI;

std::default_random_engine generator;
using namespace std;

double rand_uniform ( ) {

  uniform_real_distribution<double> distribution(0.,1.);
  return distribution(generator);

}

double rand_skewGauss ( double xi, double omega, double alpha ) {  // do NOT
 use for alpha of zero

  double delta = alpha / sqrt ( 1. + alpha * alpha );
  double gamma1 = four_minus_PI_div_2 * (pow(delta * sqrt2_div_PI, 3.) /
   pow(1. - 2. * delta * delta / M_PI, 1.5));  // skewness
  double muz = delta * sqrt2_div_PI;
  double sigz = sqrt(1. - muz * muz);
  double m_o;
  if (alpha > 0.) m_o = muz - 0.5 * gamma1 * sigz - 0.5 * exp(-two_PI / alpha);
  if (alpha < 0.) m_o = muz - 0.5 * gamma1 * sigz + 0.5 * exp( two_PI / alpha);
  double mode = xi + omega * m_o;
  // the height should be the value of the PDF at the mode
  double height = exp(-0.5 * (pow((mode - xi) / omega, 2.))) / (sqrt2_PI *
   omega) * erfc(-1. * alpha * (mode - xi) / omega / sqrt2);
  bool gotValue = false;
  double minX = xi - 6. * omega;
  double maxX = xi + 6. * omega;  // +/- 6sigma should be essentially +/-
   infinity
  //  can increase these for even better accuracy, at the cost of speed
  double testX, testY, testProb;
  while (gotValue == false) {
    testX = minX + (maxX - minX) * rand_uniform();
    testY = height * rand_uniform();  // between 0 and peak height
    // calculate the value of the skewGauss PDF at the test x-value
    testProb = exp(-0.5 * (pow((testX - xi) / omega, 2.))) / (sqrt2_PI *
     omega) * erfc(-1. * alpha * (testX - xi) / omega / sqrt2);
```

```
    if (testProb >= testY) gotValue = true;
  }
  return testX;

}

double popWorld ( double time, unsigned short int mode ) {

  double total = 0.0;

  if ( mode == 0 )
    total = 8.183e9*exp(-0.5*pow((time-2049.1)/63.746,2.));
  if ( mode == 1 )
    total = 1.027e10+(2.96e8-1.027e10)/pow(1.+pow(time/2020.9,56.388),1.988);
  if ( mode == 2 )
    total = 4.0181e+11-4.7813e+8*time+1.4014e+5*time*time;
  if ( total < 0. ) total = 0.;

  return total;

}

double smartFrac ( double time, unsigned short int mode ) {

  double smartPhoneFraction = 0.0;

  smartPhoneFraction =
   -0.26506+(-1.5883+0.26506)/(1.+pow((time-1999.5)/12.387,6.7706));
  smartPhoneFraction = pow(10.,smartPhoneFraction);
  if ( mode == 1 && time >= 2022. )
    smartPhoneFraction += 0.0045238 * (time-2021.);
  if ( mode == 2 && time >= 2022. )
    smartPhoneFraction += 0.01 * (time-2021.);
  if ( smartPhoneFraction > 1. ) smartPhoneFraction = 1.;
  if ( smartPhoneFraction < 0. ) smartPhoneFraction = 0.;
  if ( time < 2007. ) smartPhoneFraction = 0.;

  return smartPhoneFraction;

}

int main ( int argc, char **argv ) {

  default_random_engine genMain;

  unsigned short mode, numCrash;
  cout << "World total population clock? (low=0,medium=1,high=2): ";
  cin >> mode;

  double rate;
  cout << "Crashes per century: ";
```

```
cin >> rate;
rate *= 1e-2;

unsigned int start;
cout << "The starting year is: ";
cin >> start;

double distance; unsigned short witMin = 1;
if ( mode == 0 ) distance = 0.075;
if ( mode == 1 ) distance = 0.15;
if ( mode == 2 ) distance = 0.300;

double year = (double)start;
poisson_distribution<unsigned short> distribution(rate);
unsigned int yrSeen[TRIALS];
for ( unsigned long trialNum = 0; trialNum < TRIALS; ++trialNum ) {

  double popNow = 8e9, xi = 1.8149;
  numCrash = 0;
  unsigned long int peopleActual = 0; //those with smartphones
  while ( peopleActual < witMin ) {
    numCrash = distribution(genMain);
    if ( numCrash > 0 ) {
  popNow = popWorld ( double(year), mode );
  xi = 1.5467 + 4.2773e-11 * popNow;
    }
    for ( unsigned short u = 0; u < numCrash; ++u ) {
  double logPopRho = -2.1;
  while ( logPopRho < -2. || logPopRho > 5. ) {
    logPopRho = rand_skewGauss ( xi, 1.711, -2.4 );
    if ( mode == 2 ) break;
  }
  double popRho = pow(10.,logPopRho); //people/km^2
  double people = popRho * M_PI * distance * distance;
  poisson_distribution<unsigned long int>
   distribution2(people*smartFrac(double(year),mode));
  peopleActual = distribution2(genMain);
  cerr.precision(3);
  cerr << trialNum << "\t" << year << "\t" << popRho << "\t" << people <<
   "\t" << peopleActual << endl;
  if ( peopleActual >= witMin ) { yrSeen[trialNum] = (unsigned int)year;
   break; }
    }
    //peopleActual = 0;
    ++year;
    numCrash = 0;
  }

  year = start; numCrash = 0;

}
```

```
  for ( unsigned long i = 0; i < TRIALS; ++i ) cout << yrSeen[i] << endl;

  return 0;
}
```