Exercice : Echantillonnage de signaux

```cpp
/* Programme pour tester le th. d'échantillonnage
 */

#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

#ifndef M_PI
#  define M_PI 3.14159265358979323846
#endif

// -------------------------------------------------
struct Sinusoide {
  double amplitude;
  double frequence;
  double dephasage;
};

// -------------------------------------------------
typedef vector<Sinusoide> Signal;

// -------------------------------------------------
struct Signal_Echant {
  double fe;        // fréquence d'échantillonnage
  double t0;        // temps initial
  vector<double> echantillons;
};

// ========================================================================
// fonction outil : composante sinusoidale pure

double sin_pure(double x, Sinusoide const& s = { 1.0, 1.0, 0.0 })
{
  return s.amplitude * sin(2 * M_PI * s.frequence * x + s.dephasage);
}

// ========================================================================
// fonction outil : signal = somme de sinus

double signal(double x, Signal const& s)
{
  double val(0.0);
  for (auto composante : s) {
    val += sin_pure(x, composante);
  }
  return val;
}

// ========================================================================
// échantillonnage d'un signal

Signal_Echant echantillonne(Signal const& s, double freq,
                            double t_min = 0.0, double t_max = 1.0)
{
  Signal_Echant se;
  se.fe = freq;
  se.t0 = t_min;
  se.echantillons = vector<double>(1+size_t(freq * (t_max - t_min)));

  for (size_t i(0); i < se.echantillons.size(); ++i) {
    se.echantillons[i] = signal(t_min + i / freq, s);
  }
  return se;
}
```

```cpp
// ============================================================================
// fonction outil : sinus cardinal

double sinc(double x, double precision = 1e-8)
{
  return (abs(x) < precision ? 1.0 : sin(M_PI * x)/(M_PI * x));
}

// ============================================================================
// formule de reconstruction

double reconstruction(double x, Signal_Echant const& s)
{
  double valeur(0.0);
  cout << endl << "\t+ fe = " << s.fe << endl << "\t+ t0=" << s.t0 << endl
       << "\t+ nb ech = " << s.echantillons.size() << endl;
  for (size_t i(0); i < s.echantillons.size(); ++i) {
    cout << "\t\t" << s.echantillons[i] << " * sinc(" << s.fe << " * (" << x << " - " <<
    valeur += s.echantillons[i] * sinc(s.fe * (x - s.t0) - i);
  }
  cout << "\t-> " << valeur << endl;
  return valeur;
}

// ============================================================================
int main()
{
  Signal s({
    { 1.0 , 2.0, 0.0 }, // première composante
    { 0.5 , 4.0, 0.1 }, // ...
    { 0.33, 6.0, 0.2 }, // ...
    { 0.25, 8.0, 0.3 }, // ...
   });

  constexpr double t_min(0.0);
  constexpr double t_max(2.0);

  Signal_Echant se1(echantillonne(s, 20.0, t_min, t_max));
  Signal_Echant se2(echantillonne(s, 11.0, t_min, t_max));

  // « dessin » des trois courbes
  constexpr size_t nb_points(1000);
  constexpr double dt((t_max-t_min) / nb_points);

  for (double t(t_min); t <= t_max; t += dt) {
    cout << t
         << " " << signal(t, s)
         << " " << reconstruction(t, se1)
         << " " << reconstruction(t, se2)
      << endl;
  }

  return 0;
}
```