```cpp
// Programme C++ du calcul de la fréquence de collision sans dimension zc et de la vitesse de réaction sans dimension vr de deux molécules dans une enceinte
// Théorie cinétique de l'équilibre chimique
// Ce programme doit être compilé puis exécuté dans un répertoire qui contient un fichier texte nommé : resultat.txt

#include <iomanip>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <math.h> // librairie des fonctions exp, log, power
#include <time.h> // librairie de la fonction time

using namespace std;

// Déclaration d'une fonction récursive du calcul d'un nombre entier x à la puissance p
int pow_int(int x, int p)
{
    if (p == 0) return 1;
    if (p == 1) return x;
    int tmp = pow_int(x, p / 2);
    if (p % 2 == 0) return tmp * tmp;
    else return x * tmp * tmp;
}

int main()

{
    // Déclaration des variables réelles
    long double alp; // Rapport entre la hauteur du col d'enthalpie de transition et l'énergie de collision kB.T
    long double bet; // Rapport entre les masses de B et A
    long double gam; // Rapport entre les rayons de B et A
    long double del; // Variable sans dimension pour le calcul de la section efficace
    long double f; // Section efficace sans dimension
    long double u[6000], du[6000], fu[6000]; // Vitesse, vitesse élémentaire et densité de vitesse sans dimensions
    long double normalisation; // Normalisation de la fréquence de collision et de la vitesse de réaction en fonction du nombre d'itérations
    const long double pi = 3.1415926535897;
    long double zc, zc_normalisee; // Fréquences de collision sans dimensions
    long double vr, vr_normalisee; // Vitesses de réaction sans dimensions
    long double vx, vy, vz, vab; // Composante et norme de la différence de vitesse sans dimension entre A et B
    long double h, rab; // Variable h utilisée pour le calcul de la variable opérateur de réaction rab
    long double x; // Variable mémoire intermédiaire
    long double ta, tb, fa, fb, wa, wb; // Angles

    // Déclaration des variables de mesure du temps
    clock_t t1, t2, temps_execution;

    // Déclaration des variables entières
    long int i, j; // Variables de boucle
```

```
48      long int k, l, ent[4], n ; // Variable de comptage du nombre d'itérations du calcul
49      long int n1, n2; // Variables mémoires intermédiaire
50      long int a, b; // Variables utilisées pour la détermination aléatoire des vitesses sans dimensions
51
52      // Déclaration des variables chaînes de caractères pour sauvegarder les résultats
53      string resultat, verification;
54
55      // Déclaration d'une variable fichier pour sauvegarder les résultats
56      auto const filename = "resultat.txt";
57
58      // Déclaration de variables de flux pour la sauvegarde des résultats
59      ostringstream formatage;
60      ofstream file1(filename, ios::app);
61
62      // Saisie du paramètre alpha
63      cout << endl;
64      alp = 0;
65      while ((alp < 0.00299) or (alp > 301))
66      {
67          cout << "Entrer la valeur du parametre alpha (comprise entre 0.003 et 300) : ";
68          cin >> alp;
69      }
70
71      // Saisie du paramètre beta
72      bet = 0;
73      while ((bet < 0.099) or (bet > 10.1))
74      {
75          cout << "Entrer la valeur du parametre beta (comprise entre 0.1 et 10) : ";
76          cin >> bet;
77      }
78
79      // Saisie du paramètre gamma
80      gam = 0;
81      while ((gam < 0.099) or (gam > 10.1))
82      {
83          cout << "Entrer la valeur du parametre beta (comprise entre 0.1 et 10) : ";
84          cin >> gam;
85      }
86
87      // Initialisation de la variable t1 avec l'horaire pour calculer le temps de calcul
88      t1 = clock();
89
90      // Calculs des vitesse u, vitesses élémentaires du et densités de vitesses fu sans dimensions sur le domaine de sommation
91      x = alp / 3000;
92      for (i = 0; i < 3000; i++) { du[i] = x; u[i] = (i + 0.5) * x; fu[i] = pow(u[i], 0.5) * exp(-u[i]); }
93      x = (330 - alp) / 3000;
```

```
94        for (i = 3000; i < 6000; i++) { du[i] = x; u[i] = alp + (i - 2999.5) * x; fu[i] = pow(u[i], 0.5) * exp(-u[i]); }
95
96        // Sauvegarde des paramètres du calcul dans le fichier resultat.txt
97        formatage << setprecision(20) << alp;
98        resultat = formatage.str();
99        file1 << "alp = " << resultat << endl;
100       formatage.clear(); formatage.str("");
101
102       formatage << setprecision(20) << bet;
103       resultat = formatage.str();
104       file1 << "bet = " << resultat << endl;
105       formatage.clear(); formatage.str("");
106
107       formatage << setprecision(20) << gam;
108       resultat = formatage.str();
109       file1 << "gam = " << resultat << endl;
110       formatage.clear(); formatage.str("");
111
112       // Modification du terme gamma pour le calcul de la section efficace après son enregistrement
113       if (gam > 1) { gam = 1 / gam; }
114
115       // Initialisation des variables de calcul
116       zc = 0;
117       vr = 0;
118       k = 0;
119       l = 0;
120
121       // Initialisation des variables de boucle
122       ent[0] = 1; ent[1] = 2; ent[2] = 3; ent[3] = 5;
123       k = 0; l = 0;
124       n = 0;
125
126       // Confère à la fonction rand un caractère aléatoire lors d'exécutions successives du programme
127       srand(time(0));
128
129       // Calculs de la fréquence de collision zc et de la vitesse de réaction vr
130       normalisation = 0;
131       for (i = 0; ; i++)
132       {
133           for (j = 1; j < -l + 1 + ent[i] * pow_int(10, k); j++)
134           {
135               // Détermination alléatoire des vecteurs vitesse des molécules A et B
136               a = rand() % 6000;
137               b = rand() % 6000;
138               ta = 180 * (long double)rand() / RAND_MAX;
139               tb = 180 * (long double)rand() / RAND_MAX;
140               fa = 360 * (long double)rand() / RAND_MAX;
141               fb = 360 * (long double)rand() / RAND_MAX;
```

```
142
143             // Calcul du nombre d'itération
144             n = n + 1;
145
146             // Calcul des composantes et de la norme de la différence de vitesse vab sans dimension entre les molécules A et B
147             vx = pow(bet, 0.25) * pow(u[a], 0.5) * sin(ta) * cos(fa) - pow(bet, -0.25) * pow(u[b], 0.5) * sin(tb) * cos(fb);
148             vy = pow(bet, 0.25) * pow(u[a], 0.5) * sin(ta) * sin(fa) - pow(bet, -0.25) * pow(u[b], 0.5) * sin(tb) * sin(fb);
149             vz = pow(bet, 0.25) * pow(u[a], 0.5) * cos(ta) - pow(bet, -0.25) * pow(u[b], 0.5) * cos(tb);
150             vab = pow(pow(vx, 2) + pow(vy, 2) + pow(vz, 2), 0.5);
151
152             // Calcul de la section efficace de collision sans dimension f
153             del = pow(vy * vy + vz * vz, 0.5)/vab;
154             x = del * (1 + gam) - 1 + gam;
155             if (x < 0)
156                { f = 1 / (1 + gam * gam); }
157             else
158                {
159                wa = acos((del * del * (1 + gam) + 1 - gam) / (2 * del));
160                wb = acos((del * del * (1 + gam) - 1 + gam) / (2 * del * gam));
161                f = 1 + ((sin(wa) * cos(wa) + gam * gam * sin(wb) * cos(wb) - 2 * wa - 2 * gam * gam * wb) / (pi * (1 + gam * gam)));
162                }
163
164             // Calcul de la fréquence de collision sans dimension zc
165             zc = zc + f * fu[a] * du[a] * fu[b] * du[b] * vab;
166
167             // Calcul de la vitesse de réaction sans dimension vr
168             h = pow(pow(bet, 0.25) * pow(u[a], 0.5) * sin(ta) * cos(fa) - pow(bet, -0.25) * pow(u[b], 0.5) * sin(tb) * cos(fb), 2) / (pow(bet, 0.25) + pow(bet,
169 -0.25));
170             if (h >= alp) { rab = 1; }
171             else { rab = 0; }
172             vr = vr + rab * f * fu[a] * du[a] * fu[b] * du[b] * vab;
173
174             // Calcul de la constante de normalisation de la fréquence de collision et de la vitesse de réaction
175             normalisation = normalisation + fu[a] * du[a] * fu[b] * du[b];
176          }
177          zc_normalisee = zc / normalisation;
178          vr_normalisee = vr / normalisation;
179          n1 = ent[i];
180          n2 = k;
181
182          // Affichage du résultat des calculs
183          cout << "N iterations = " << n << endl;
184          cout << "zc = " << setprecision(4) << zc_normalisee << endl;
185          cout << "vr = " << setprecision(4) << vr_normalisee << endl << endl;
186
187    // Sauvegarde des résultats du calcul dans le fichier resultat.txt
```

```
188             formatage << setprecision(20) << n1;
189             resultat = formatage.str(); // récupérer une chaîne de caractères
190             file1 << "n1 = " << resultat << endl;
191             formatage.clear(); formatage.str("");
192 
193             formatage << setprecision(20) << n2;
194             resultat = formatage.str(); // récupérer une chaîne de caractères
195             file1 << "n2 = " << resultat << endl;
196             formatage.clear(); formatage.str("");
197 
198             formatage << setprecision(17) << zc;
199             resultat = formatage.str(); // récupérer une chaîne de caractères
200             file1 << "zc non normalisée = " << resultat << endl;
201             formatage.clear(); formatage.str("");
202 
203             formatage << setprecision(17) << vr;
204             resultat = formatage.str(); // récupérer une chaîne de caractères
205             file1 << "vr non normalisée = " << resultat << endl;
206             formatage.clear(); formatage.str("");
207 
208             formatage << setprecision(17) << normalisation;
209             resultat = formatage.str(); // récupérer une chaîne de caractères
210             file1 << "normalisation = " << resultat << endl;
211             formatage.clear(); formatage.str("");
212 
213             formatage << setprecision(17) << zc_normalisee;
214             resultat = formatage.str(); // récupérer une chaîne de caractères
215             file1 << "zc = " << resultat << endl;
216             formatage.clear(); formatage.str("");
217 
218             formatage << setprecision(17) << vr_normalisee;
219             resultat = formatage.str(); // récupérer une chaîne de caractères
220             file1 << "vr = " << resultat << endl << endl;
221             formatage.clear(); formatage.str("");
222 
223             // Condition pour sortir de la boucle pour un nombre d'itérations égal à 10^9
224             if (n2 > 8) { break; }
225             l = ent[i] * pow_int(10, k);
226             if (i == 3) { i = -1; k = k + 1; }
227 
228         }
229 
230         // Affichage du nombre d'itérations et du temps de calcul
231         t2 = clock();
232         temps_execution = (t2 - t1) / 1000;
233         cout << endl << endl << "Le nombre d'iterations est egal a : " << n << endl;
234         cout << endl << "Le temps d execution est de : " << temps_execution << " secondes." << endl;
235         formatage << setprecision(17) << temps_execution;
```

```
236         resultat = formatage.str();
237         file1 << "temps de calcul = " << resultat << " secondes." << endl << endl;
238         formatage.clear(); formatage.str("");
239 
240         // Instruction de fin de programme
241         return 0;
242 
243     }
```