```c
#include <stdlib.h>
#include <stdio.h>

//~ local
#include "graph.h"
#include "const.h"

graphe* init_graphe(){
    graphe* G = malloc(sizeof(graphe));

    int i, j, p = 0;

    // struct liste
    for(i = 0; i < TAILLE_GRAPHE; i++) // ini toute le tableau a -1
    {
        for(j = 0; j < TAILLE_GRAPHE; j++)
        {
            G->list[i][j] = -1;
        }
    }

    // struct Insert
    int found = 0;
    int cmp2;
    while(!found)
    {
        cmp2 = 0;
        for(i = 0; i < TAILLE_GRAPHE; i++)
        {
            if( i < nbTier1 )
            {
                G->I.proba[i][0] = 0;
            }
            if( i >= debTier2 && i < finTier2)
            {
                p = rand()%2 + 2;
                G->I.proba[i][0] = p;
            }
            if( i >= debTier3 && i < finTier3)
            {
                G->I.proba[i][0] = 1;
            }
            for(j = 0; j < 3; j++)
            {
                G->I.compteur[i][j] = 0;
            }
        }

        for(i = debTier2; i < finTier2; i++)
        {
            cmp2 += G->I.proba[i][0] = p;
        }

        if( cmp2 % 2 == 0 )
        {
            found = 1;
        }
```

```
58      }
59      return G;
60  }
61
62  int verifSiSommetInListe(graphe* G, int i, int j) {
63
64      if(G->list[i][j]!= -1)
65          return 1;
66      return 0;
67  }
68
69  int test_noeuds_max(graphe* G, int i, int noeudsMax, int etat) {
70
71          if(etat == tier1)
72          {
73              if(G->I.compteur[i][etat] < noeudsMax)
74                  return 1;
75          }
76          else
77          {
78              if(G->I.compteur[i][etat] < noeudsMax &&  G->I.compteur[i][etat] <
                G->I.proba[i][0])
79                  return 1;
80          }
81
82      return 0;
83  }
84
85  int calcul_noeuds(graphe* G, int sommet, int deb, int fin, int noeudsMax, int
    etat_sommet, int etat_i) {
86
87      int distance = fin - deb;
88      int pointeur[distance+1];
89      int i, j = 0;
90
91      for(i = 0; i < distance; i++)
92          pointeur[i] = 0;
93
94      for(i = deb; i < fin; i++)
95      {
96          if(G->I.compteur[i][etat_i] < noeudsMax
97              && !verifSiSommetInListe(G, sommet, i)
98              && i != sommet)
99          {
100             pointeur[j] = i;
101             j++;
102         }
103     }
104     if(j != 0)
105     {
106         int a, b;
107         b = rand();
108         a = b % j;
109         b = pointeur[a];
110         return b;
111     }
112     return -1;
```

```
113 }
114
115 void calculTier1(graphe* G) {
116
117
118     int i, j, k;
119     int p = 7500;
120
121     for(i = debTier1; i < finTier1 ; i++)
122     {
123         for(j = debTier1; j < finTier1; j++)
124         {
125             if(i != j && !verifSiSommetInListe(G, i, j))
126             {
127                 k = rand()%1000;
128                 if(k < p)
129                 {
130                     k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
131
132                     G->list[i][j] = k;
133                     G->list[j][i] = k;
134                     G->I.compteur[i][0]++;
135                     G->I.compteur[j][0]++;
136                 }
137              }
138          }
139     }
140 }
141
142 void calculTier2(graphe* G){
143
144
145     int i, j;
146     int p, noeuds, k;
147
148     for(i = debTier2; i < finTier2; i++)
149     {
150         // pour les arc vers le tier precedent
151         p = rand()%2 + 1;
152         for(j = 0; j < p; j++)
153         {
154                 noeuds = calcul_noeuds(G, i, debTier1, finTier1, 100, tier2, tier1);
155                 k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
156                 if(noeuds != -1)
157                 {
158                     G->list[i][noeuds] = k;
159                     G->list[noeuds][i] = k;
160                 }
161         }
162
163         // pour les arc vers le tier current
164         for(j = 0; G->I.compteur[i][tier2] < G->I.proba[i][0] && j <
        G->I.proba[i][0]; j++)
165         {
166             if(test_noeuds_max(G, i, noeudsMaxTier2, tier2))
167             {
168                 noeuds = calcul_noeuds(G, i, debTier2, finTier2, noeudsMaxTier2,
```

```
                tier2, tier2);
169             k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
170             if(noeuds != -1)
171             {
172                 G->list[i][noeuds] = k;
173                 G->list[noeuds][i] = k;
174
175                 G->I.compteur[i][1]++;
176                 G->I.compteur[noeuds][1]++;
177             }
178         }
179     }
180  }
181  }
182
183  void calculTier3(graphe* G){
184
185      int i, j, k;
186      int p, noeuds;
187
188      for(i = debTier3; i < finTier3; i++)
189      {
190          // pour les arc vers le tier precedent
191          p = 2;
192          for(j = 0; j < p; j++)
193          {
194                  noeuds = calcul_noeuds(G, i, debTier2, finTier2, 100, tier3, tier2);
195                  k = rand()%(poidsMaxTier3 - poidsMinTier3 + 1) + poidsMinTier3;
196
197                  if(noeuds != -1)
198                  {
199                      G->list[i][noeuds] = k;
200                      G->I.compteur[i][tier2]++;
201
202                      G->list[noeuds][i] = k;
203                      G->I.compteur[noeuds][tier3]++;
204                  }
205          }
206
207          // pour les arc vers le tier current
208          for(j = 0; G->I.compteur[i][tier3] < G->I.proba[i][0] && j <
          G->I.proba[i][0]; j++)
209          {
210              if(test_noeuds_max(G, i, noeudsMaxTier3, tier3))
211              {
212                  noeuds = calcul_noeuds(G, i, debTier3, finTier3, noeudsMaxTier3,
                  tier3, tier3);
213                  k = rand()%(poidsMaxTier3 - poidsMinTier3 + 1) + poidsMinTier3;
214
215                  if(noeuds != -1)
216                  {
217                      G->list[i][noeuds] = k;
218                      G->I.compteur[i][tier3]++;
219
220                      G->list[noeuds][i] = k;
221                      G->I.compteur[noeuds][tier3]++;
222                  }
```

```
223                 }
224             }
225         }
226     }
227
```