

```
1 #ifndef __TOOLS_H
2
3 #define __TOOLS_H
4
5 typedef struct node_t {
6     int id;
7     struct node_t * fg, *fd;
8 } node_t;
9 #define N 256
10 #define MAX(a, b) ((a) > (b) ? (a) : (b))
11
12 extern void push(node_t * v);
13 extern node_t * pop(void);
14 extern int empty(void);
15 extern void enfile(int v);
16 extern int defiler(void);
17 extern int vide(void);
18
19 #endif
20 #include "tools.h"
21
22 static int haut = 0, file[N + 1], debut = 0, fin = 0;
23 static node_t * pile[N];
24
25 extern void push(node_t * v) {
26     pile[haut++] = v;
27 }
28 extern node_t * pop(void) {
29     return pile[--haut];
30 }
31 extern int empty(void) {
32     return !haut;
33 }
34 extern void enfile(int v) {
35     file[fin++] = v;
36     if(fin > N)
37         debut = 0;
38     fin = 0;
39 }
40 extern int defiler(void) {
41     int v = file[debut++];
42     if(debut > N)
43         debut = 0;
44     return v;
45 }
46 extern int vide(void) {
47     return debut == fin;
48 }
```

FIGURE 1 – Arbres binaires, piles et files (fichiers tool.h et tool.c)

```

1  #include "tools.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5
6  node_t * newNode(int id) {
7      node_t * n = malloc(1 * sizeof *n); assert(n);
8      n->id = id; n->fg = n->fd = NULL;
9      return n;
10 }
11 node_t ** findPosition(int id, node_t ** tree) {
12     if(*tree == NULL) return tree;
13     if(id < (*tree)->id) return findPosition(id, &((*tree)->fg));
14     return findPosition(id, &((*tree)->fd));
15 }
16 void printTree(node_t * ptr) {
17     if(ptr == NULL) return;
18     printTree(ptr->fg); printf("%d ", ptr->id); printTree(ptr->fd);
19 }
20 void printTreeP(node_t * ptr) {
21     while(ptr != NULL || !empty()) {
22         if(ptr) {
23             push(ptr);
24             ptr = ptr->fg;
25             continue;
26         } else
27             ptr = pop();
28         printf("%d ", ptr->id);
29         ptr = ptr->fd;
30     }
31 }
32 void tree2File(node_t * ptr) {
33     if(ptr == NULL) return;
34     tree2File(ptr->fg); enfile(ptr->id); tree2File(ptr->fd);
35 }
36 void printFile() {
37     while(!vide()) printf("%d ", defiler());
38     printf("\n");
39 }
40 int maxDepth(node_t * ptr, int cd) {
41     int a, b;
42     if(ptr == NULL) return cd;
43     a = maxDepth(ptr->fg, ++cd); b = maxDepth(ptr->fd, cd);
44     return MAX(a, b);
45 }
46 void freeTree(node_t * ptr) {
47     node_t * tmp;
48     if(ptr == NULL) return;
49     freeTree(ptr->fg);
50     tmp = ptr->fd;
51     free(ptr); freeTree(tmp);
52 }
53 int main(void) {
54     node_t * a = NULL;
55     int i, n = 100, v;
56     for(i = 0; i < n; i++) {
57         v = n * (rand() / (RAND_MAX + 1.0));
58         *(findPosition(v, &a)) = newNode(v);
59     }
60     v = maxDepth(a, 0); assert(v <= N);
61     printf("Profondeur de l'arbre : %d\n", v);
62     /* Imprimer l'arbre récursivement */
63     printTree(a); printf("\n");
64     /* Imprimer l'arbre en utilisant Notre PILE */
65     printTreeP(a); printf("\n");
66     /* Utiliser Notre FILE */
67     tree2File(a); printFile();
68     /* Libérer l'arbre */
69     freeTree(a); a = NULL;
70     return 0;
71 }

```

FIGURE 2 – Arbres binaires, piles et files (fichier ab.c)

```
1  digraph G {  
2      A [shape=rect];  
3      A -> B;  
4      B -> C;  
5      C -> D;  
6      D -> A;  
7 }
```

FIGURE 3 – Exemple de description de graphe avec GraphViz

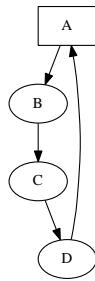


FIGURE 4 – Résultat obtenu pour l'exemple Figure 3

```
1  digraph G {  
2      A [shape=box];  
3      B [label="Je change de nom"];  
4      C [color=red];  
5      A -> B;  
6      B -> C;  
7      C -> D;  
8      D -> A [style=dotted];  
9 }
```

FIGURE 5 – Autre exemple de description de graphe avec GraphViz

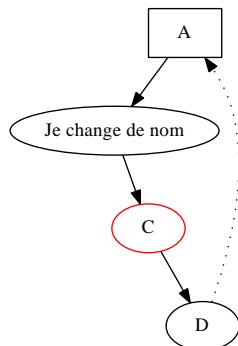


FIGURE 6 – Résultat obtenu pour l'exemple Figure 5

```

1  #include <time.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5  typedef struct node_t node_t;
6  struct node_t {
7      int id;
8      struct node_t * fg, *fd;
9  };
10 #define MAX(a, b) ((a) > (b) ? (a) : (b))
11 node_t * newNode(int id) {
12     node_t * n = malloc(1 * sizeof *n); assert(n);
13     n->id = id; n->fg = n->fd = NULL;
14     return n;
15 }
16 node_t ** findPosition(int id, node_t ** tree) {
17     if(*tree == NULL) return tree;
18     if(id < (*tree)->id) return findPosition(id, &((*tree)->fg));
19     return findPosition(id, &((*tree)->fd));
20 }
21 int maxDepth(node_t * ptr, int cd) {
22     int a, b;
23     if(ptr == NULL) return cd;
24     a = maxDepth(ptr->fg, ++cd);
25     b = maxDepth(ptr->fd, cd);
26     return MAX(a, b);
27 }
28 void freeTree(node_t * ptr) {
29     node_t * tmp;
30     if(ptr == NULL) return;
31     freeTree(ptr->fg);
32     tmp = ptr->fd;
33     free(ptr); freeTree(tmp);
34 }
35 static int myNum = 0;
36 void tree2GVsub(node_t * ptr, FILE * out, int cd) {
37     if(ptr->fg) {
38         fprintf(out, "\t\"%p\" -> \"%p\";\n", ptr, ptr->fg);
39         tree2GVsub(ptr->fg, out, cd - 1);
40     }
41     fprintf(out, "\t\"%p\" [label=\"%d\", pos=\"%f,%f!\"];\n", ptr, ptr->id, myNum / 10.0, cd / 10.0);
42     myNum++;
43     if(ptr->fd) {
44         fprintf(out, "\t\"%p\" -> \"%p\";\n", ptr, ptr->fd);
45         tree2GVsub(ptr->fd, out, cd - 1);
46     }
47 }
48 void tree2GV(node_t * ptr, FILE * out) {
49     fprintf(out, "digraph G {\n");
50     if(ptr) {
51         myNum = 1;
52         tree2GVsub(ptr, out, maxDepth(ptr, 0));
53     }
54     fprintf(out, "}\n");
55 }
56 int main(void) {
57     node_t * a = NULL;
58     srand(time(NULL));
59     int i, n = 10, v;
60     for(i = 0; i < n; i++) {
61         v = n * (rand() / (RAND_MAX + 1.0));
62         *(findPosition(v, &a)) = newNode(v);
63     }
64     /* Méta-prog produisant du GraphViz */
65     /* faire quelque chose comme:
66        gcc -Wall ab.c -o ab && ./ab > out.gv &&
67        dot -Kfdp -Tps out.gv -o out.ps && ps2pdf out.ps
68        pour obtenir un fichier pdf contenant votre arbre
69     */
70     tree2GV(a, stdout);
71     /* Libérer l'arbre */
72     freeTree(a); a = NULL;
73     return 0;
74 }

```

FIGURE 7 – Graphes (ici arbre binaire) et métaprogrammation pour l’outil GraphViz.

- Voir l'outil GraphViz ;
- Voir (sur la page web du cours) le support concernant la génération de graphes connexes (labyrinthes) et la résolution de plus courts chemins dans ces graphes.