```
 1: #include<stdio.h>
 2: #include<stdlib.h>
 3:
 4: //#define MAX_PILE 10000
 5:
 6: typedef int element;
 7:
 8: typedef
 9: struct pile_contigue {
10:         element * espace ;
11:         int dernier;
12:         int taille;
13: }pile_contigue;
14:
15: void init_pile(pile_contigue * pp, int taille) {
16:         pp->dernier = -1;
17:         pp->taille = taille ;
18:         pp->espace = malloc(taille * sizeof(element));
19: }
20:
21: void empiler(pile_contigue * pp, int v) {
22:         //tester si pleine
23:         pp->dernier = pp->dernier + 1;
24:         pp->espace[pp->dernier] = v;
25: }
26:
27: //void dÃ©piler2(pile_contigue *pp, element *v) {
28: //}
29:
30: element dÃ©piler(pile_contigue * pp) {
31:         return pp->espace[pp->dernier--];
32: }
33:
34: int sommet(pile_contigue *pp) {
35:         return pp->espace[pp->dernier];
36: }
37:
38: int pile_vide(pile_contigue *pp) {
39:         return pp->dernier < 0;
40: }
41:
42: int pile_pleine(pile_contigue *pp) {
43:         return pp->dernier == pp->taille -1;
44: }
45:
```

```
46:
47: int main() {
48:         pile_contigue p2;
49:         init_pile(&p2, 10);
50:
51:         pile_contigue p;
52:         init_pile(&p, 10000);
53:
54:         empiler(&p, 45);
55:         if (!pile_pleine(&p))
56:                 empiler(&p, 5);
57:         empiler(&p, 4);
58:         empiler(&p, 25);
59:
60:         while(!pile_vide(&p)) {
61:                 printf("%d\n",dÃ©piler(&p));
62:         }
63:         empiler(&p2, 17);
64:
65:         free(p.espace);
66:         free(p2.espace);
67: }
68:
```