```c
#include <stdlib.h>
#include <stdio.h>
#include "memoire.h"
#include "arbre.h"

typedef int *couple;

static couple
creer_couple (int a, int b)
{
  couple c = (int *) mem_malloc (2 * sizeof (int));
  c[0] = a;
  c[1] = b;
  return c;
}

int
cmp_sur_1 (const couple a, const couple b)
{
  return (a[0] > b[0]) - (a[0] < b[0]);
}

void
afficher (couple c)
{
  printf ("(%d, %d)\n", c[0], c[1]);
}

void
afficher_elem (objet vide, objet cpl)
{
  afficher ((couple) cpl);
}

/**********************************************************************/
/* la genericité permet un developpement rapide et evite, en partie,    */
/* la duplication de code au detriment - parfois - des perfomances      */
/* en voici une illustration                                            */
/**********************************************************************/

static void
incremente (int *n, objet elt)
{
  (*n)++;
}

static unsigned int
cardinal (arbre a)
{
  unsigned int taille = 0;
  abr_appliquer (a, (appliquer) incremente, &taille);
  return taille;
}

/************************************************************************/
/* voici une version un peu mysterieuse de abr2tableau                  */
/* exo: faire plus efficace à l'aide d'un type tableau dynamique        */

static void
ranger (objet ** tab, objet elt)
/* *tab contient l'adresse de la prochaine case à remplir     */
{
```



```c
  *(*tab)++ = elt;              /* equivalent à **tab = elt; *tab = *tab + 1; */
}

static couple *
abr2tableau (arbre a)
{
  couple *tableau, *tmp;
  tmp = tableau = (couple *) mem_malloc (cardinal (a) * sizeof (couple));
  abr_appliquer (a, (appliquer) ranger, &tmp);
  return tableau;
}

/**********************************************************************/
struct resultat
{
  int compteur;
  couple valeur;
};

static void
valeur_ieme (struct resultat *r, objet elt)
{
  r->compteur--;
  if (r->compteur == 0)
    r->valeur = elt;
}

static objet
tester_dixieme (arbre a)
{
  struct resultat r = { 10, NULL };
  abr_appliquer (a, (appliquer) valeur_ieme, &r);
  return r.valeur;
}

/***********************************************************************/

int
main (int argc, char *argv[])
{
  arbre a = abr_creer ((comparer) cmp_sur_1);
  {                             /* Creation d'un arbre un peu tordu */
    int i;
    for (i = 0; i < 12; i += 4)
      {
        abr_inserer (a, creer_couple (i + 3, 20 - i));
        abr_inserer (a, creer_couple (i, 40 - i));
        abr_inserer (a, creer_couple (i + 2, 30 - i));
        abr_inserer (a, creer_couple (i + 1, 100 - i));
      }
  }
  abr_appliquer (a, (appliquer) afficher_elem, NULL);
  {                             /* test abr2tableau et cardinal */
    unsigned int i, taille;
    couple *tableau = abr2tableau (a);
    taille = cardinal (a);
    for (i = 0; i < taille; i++)
      afficher (tableau[i]);
  }
  afficher (tester_dixieme (a));
  return EXIT_SUCCESS;
}
```