Feuille de révisions n°4 - Programmation en C : manipulation de tableaux

1 Création de tableaux

Solution

Dans toutes les corrections on utilise les types suivants.

```c
//struct pour les tableaux d'entiers munis de leur taille
struct tableau_s {
  int  taille;
  int* tab   ;
};
typedef struct tableau_s tableau;
```

```c
//struct pour les tableaux de tableaux d'entiers munis de leur taille
struct tabtab_s {
  int      taille;
  tableau* tab   ;
};
typedef struct tabtab_s tabtab;
```

```c
//struct pour les tableaux de booléens munis de leur taille
struct btableau_s {
  int   taille;
  bool* tab   ;
};
typedef struct btableau_s btableau;
```

R. 4-1 Écrire une fonction prenant en argument un entier naturel non nul n et retournant un tableau de n booléens rempli de `false` à l'aide d'une boucle **`while`**.

Solution

```c
btableau genere_tab_false(int n){
  bool* tab = malloc (n*sizeof(bool));
  int  i    = 0 ;
  while (i < n) {
    tab[i] = false;
  }
  btableau res = {n, tab};
  return res;
}
```

R. 4-2 Écrire une fonction prenant en argument un entier naturel non nul n et retournant un tableau contenant les entiers de 1 à n à l'aide d'une boucle **`for`**.

Solution

```
tableau genere_tab_n (int n){
  int* tab = malloc (n*sizeof(int));
  for(int i = 0 ; i < n ; i++){tab[i] = i+1;}
  tableau res = {n, tab};
  return res;
}
```

R. 4-3 Écrire un **programme** qui crée le tableau contenant les entiers 1, 2, 4, 8, 12, 56, -3 et 19, puis qui affiche un à un ces éléments.

Solution

Après avoir défini la fonction d'affichage suivante :

```
void affiche_tableau (tableau t){
  if (t.taille == 0){
    printf("tableau VIDE \n");
  }
  else{
    printf("tableau contenant les %d éléments suivants :  ", t.taille);
    for(int i = 0 ; i < t.taille ; i++){
      printf("%d   ", t.tab[i]);
    }
    printf("\n");
  }
}
```

On définit le programme suivant :

```
int main(){
  int tab1[9] = {1, 2, 4, 8, 7, 12, 56, -3, 19};
  tableau tableau1 = {9,tab1};
  affiche_tableau(tableau1);
  return 0;
}
```

R. 4-4 Écrire une fonction prenant en argument un entier naturel non nul n, et deux entiers relatifs a et b tels que $a < b$ et et retournant un tableau de n entiers aléatoires tirés uniformément dans $[\![a, b[\![$. On veillera à ce que le tableau construit par l'appel à cette fonction change potentiellement à chaque exécution.

Solution

```
tableau genere_tab_n_alea (int n, int a, int b){
  //hyp : n>0 && a<b
  //initialiser la graine de rand avant usage !
  int* tab = malloc (n*sizeof(int));
  int l = b-a;
  for(int i = 0 ; i < n ; i++){
    tab[i] = rand() % l + a;
  }
```

```
  tableau res = {n,tab};
  return res;
}
```

On peut tester cette fonction avec le programme suivant :

```
int main(){
  srand(time (NULL));
  tableau t1 = genere_tab_n_alea (10, 0, 100);
  tableau t2 = genere_tab_n_alea (10, 50, 100);
  tableau t3 = genere_tab_n_alea (20, -12, -5);
  affiche_tableau(t1);
  affiche_tableau(t2);
  affiche_tableau(t3);
  return 0;
}
```

2 Agrégation

R. 4-5 Écrire une fonction testant l'existence d'un `0` dans un tableau d'entiers parcouru par boucle `while`.

Solution

```
bool existe0(tableau t) {
  int i = 0;
  while (t.tab[i] != 0) {
    i = i+1;
  }
  return (i < t.taille);
}
```

R. 4-6 Écrire une fonction calculant l'indice du premier `0` dans un tableau d'entiers parcouru par boucle `while` arrêtée dès que possible, sans utiliser de `break`, ni de `return` dans la boucle. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
int find0_v0(tableau t) {
  int i = 0;
  while (t.tab[i] != 0) {
    i = i+1;
  }
  if (i == t.taille) {
    return -1;
  } else {
    return i;
  }
}
```

R. 4-7 Écrire une fonction calculant l'indice du premier `0` dans un tableau d'entiers parcouru par boucle `while` arrêtée dès que possible grâce à une instruction `return` dans la boucle. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
int find0_v1(tableau t) {
  int i = 0;
  while (i < t.taille) {
    if (t.tab[i] == 0) {return i;}
    i = i + 1;
  }
  return -1;
}
```

R. 4-8 Écrire une fonction calculant l'indice du premier `0` dans un tableau d'entiers parcouru par boucle `for` arrêtée dès que possible. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
int find0_v2(tableau t) {
  for (int i = 0 ; i < t.taille ; i ++) {
    if (t.tab[i] == 0) {return i;}
  }
  return -1;
}
```

R. 4-9 Écrire une fonction calculant l'indice du dernier `0` dans un tableau d'entiers parcouru par boucle `while` arrêtée dès que possible grâce à une instruction `return` dans la boucle. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
int find0_v3(tableau t) {
  int i = t.taille-1;
  while (i >= 0) {
    if (t.tab[i] == 0) {return i;}
    i = i-1;
  }
  return -1;
}
```

R. 4-10 Écrire une fonction prenant en argument un tableau d'entiers et calculant la somme des éléments du tableau. On fournira une implémentation au moyen d'une boucle `for`.

Solution

```
int somme(tableau t) {
  int res = 0 ;
  for (int i = 0 ; i < t.taille ; i ++) {
    res = res + t.tab[i];
  }
```

```
  return res;
}
```

R. 4-11 Écrire une fonction calculant l'indice du minimum dans un tableau d'entiers non vide parcouru par une boucle `for`.

Solution

```
int argminimum(tableau t) {
  if (t.taille == 0) {return -1;}
  else {
    int vmin = t.tab[0] ;
    int imin = 0 ;
    for (int i = 1 ; i < t.taille ; i ++) {
      if (t.tab[i] < vmin) {
        vmin = t.tab[i];
        imin = i;
      }
    }
    return imin;
  }
}
```

R. 4-12 Écrire une fonction calculant la conjonction d'un tableau de booléens parcouru par une boucle `while`.

Solution

```
bool conjonction(btableau t) {
  bool res = true;
  for (int i = 0 ; i < t.taille ; i ++) {
    res = res && t.tab[i];
  }
  return res;
}
```

R. 4-13 Écrire une fonction prenant en argument un tableau d'entiers à valeurs dans $[\![0, m-1]\!]$ où m est donné en argument, et calculant l'entier de $[\![0, m-1]\!]$ ayant le plus d'occurrences dans le tableau. On fournira une implémentation en $\mathcal{O}(n+m)$.

Solution

```
int max_occurrences(tableau t, int m) {
  int* nb_occur = malloc(m*sizeof(int));
  for (int i = 0 ; i < m ; i ++) {
    nb_occur[i] = 0;
  }
  for (int i = 0 ; i < t.taille ; i ++) {
    nb_occur[t.tab[i]] = nb_occur[t.tab[i]]+1;
  }
  int vmax = nb_occur[0] ;
```

```
  int imax = 0 ;
  for (int i = 1 ; i < m ; i ++) {
    if (nb_occur[i] > vmax) {
      vmax = nb_occur[i];
      imax = i;
    }
  }
  return imax;
}
```