OCaml - Listes

Aubin SIONVILLE

MPI Clemenceau - 2021-2023

Types utilisés

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

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

// Type pour les tableaux de booleens munis de leur taille
typedef struct {
    bool* tab;
    int taille;
} b_tableau;
```

1 Création de tableaux

Par boucle while

Fonction prenant en argument un entier naturel non nul n et retournant un tableau de n booléens rempli de 0 à l'aide d'une boucle **while**.

```
b_tableau creer_tableau_while(int n) {
    bool* tab = malloc(n * sizeof(bool));
    int i = 0;
    while (i < n) {
        tab[i] = false;
        i++;
    }
    b_tableau t = {tab, n};
    return t;
}
```

Par boucle for

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**.

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

Création et affichage

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.

```
void afficher_tableau(tableau t) {
    if (t.taille == 0) {
        printf("Tableau vide \n")
    }
    else {
        printf("Tableau de taille %d : \n", t.taille);
        for (int i = 0; i < t.taille; i++) {
            printf("%d ", t.tab[i]);
        }
        printf("\n");
    }
}

int main() {
    int tab[8] = {1, 2, 4, 8, 12, 56, -3, 19};
    tableau t = {tab, 8};
    afficher_tableau(t);
    return 0;
}
```

Tableau d'entiers aléatoires

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.

```
tableau creer_tableau_aleatoire(int n, int a, int b) {
    int* tab = malloc(n * sizeof(int));
    int l = b - a;
    for (int i = 0; i < n; i++) {
        tab[i] = rand() % l + a;
    }
    tableau t = {tab, n};
    return t;
}

int main() {
    srand(time(NULL));
    tableau t = creer_tableau_aleatoire(10, 0, 10);
    afficher_tableau(t);
    return 0;
}
```

2 Agrégation

Existence d'un élément

Fonction testant l'existence d'un 0 dans un tableau d'entiers parcouru par boucle **while**.

```
bool existe_0(tableau t) {
    int i = 0;
    while (i < t.taille && t.tab[i] != 0) {
        i++;
    }
    return i < t.taille;
}
```

Indice de la première occurrence

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.

```
int indice_premier_0(tableau t) {
    int i = 0;
    while (i < t.taille && t.tab[i] != 0) {
        i++;
    }
    if (i == t.taille) {
        return -1;
    }
    else {
        return i;
    }
}
```

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.

```
int indice_premier_0_bis(tableau t) {
    int i = 0;
    while (i < t.taille) {
        if (t.tab[i] == 0) {
            return i;
        }
        i++;
    }
    return -1;
}
```

Fonction calculant l'indice du premier 0 dans un tableau d'entiers parcouru par boucle **for**.
La fonction devra retourner -1 si le tableau ne contient aucun 0.

```
int indice_premier_0_for(tableau t) {
    for (int i = 0; i < t.taille; i++) {
        if (t.tab[i] == 0) {
            return i;
        }
    }
    return -1;
}
```

Indice de la dernière occurrence

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.

```
int indice_dernier_0(tableau t)
{
    int i = t.taille - 1;
    while (i >= 0) {
        if (t.tab[i] == 0) {
            return i;
        }
        i--;
    }
    return -1;
}
```

Somme des éléments

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**.

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

Recherche de minimum

Fonction calculant l'indice du minimum dans un tableau d'entiers non vide parcouru par une boucle **for**.

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

Conjonction d'un tableau de booléens

Fonction calculant la conjonction des éléments d'un tableau de booléens parcouru par une boucle **while**.

```
bool conjonction(tableau t) {
    bool conj = true;
    int i = 0;
    while (i < t.taille && conj) {
        conj = conj && t.tab[i];
        i++;
    }
    return conj;
}
```

Maximum d'occurrences

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 $\mathcal{O}(n+m)$

```
int maximum_occurrences(tableau t, int m) {
    int* nb_occurrences = malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) {
        nb_occurrences[i] = 0;
    }

    for (int i = 0; i < t.taille; i++) {
        nb_occurrences[t.tab[i]]++;
    }

    int max = 0;
    for (int i = 1; i < m; i++) {
        if (nb_occurrences[i] > nb_occurrences[max]) {
            max = i;
        }
    }

    free(nb_occurrences);
    return max;
}
```