Exercice 1 : Tableau dynamique

Un tableau dynamique est déterminé par un facteur de croissance et comporte généralement deux informations associées aux données : le nombre d'éléments stockés dans le tableau, la capacité du tableau en nombre d'éléments. Lorsque la capacité du tableau atteint la limite de stockage, il est étendu en prenant en compte le facteur de croissance.

Les opérations sur un tableau dynamiques sont : *push* pour ajouter un élément à la fin du tableau, *unshift* pour ajouter un élément au début du tableau, *pop* pour supprimer un élément à la fin du tableau et *shift* pour supprimer un élément en début de tableau.

Parfois d'autres opérations existent : *delete* pour supprimer un élément à l'indice K et *insert* pour ajouter un élément après l'élément K.

Utilisez ici `assert` pour le test des erreurs, et ne réduisez pas la taille du tableau, il ne fait que s'agrandir au gré des données le peuplant.

On souhaite implémenter un tableau dynamique, implémentez les fonctions demandées :

(a) Écrire une structure de donnée représentant un tableau dynamique et permettant de stocker des paires de `double`.

```
typedef struct array {
  size_t capacity;
  size_t elements;
  double (*data)[2];
} Array;
```

(b) Compte tenu de la structure déclarée précédemment, écrire une fonction permettant d'initialiser un tableau. C'est à dire allouer l'espace mémoire nécessaire et renseigner les informations nécessaires. La structure de ce tableau est la suivante :

```
typedef struct array {
  size_t capacity;
  size_t elements;
  double (*data)[2];
} Array;
```

(c) Écrire la fonction `array_init`.

```
void array_init(Array *array) {
  array->capacity = 1;
  array->elements = 0;
  array->data = malloc(sizeof(double[2]) * array->capacity);
}
```

(d) Écrire la fonction `array_push`.

```
void array_push(Array *array, double x, double y) {
  if (array->elements == array->capacity) {
    array->capacity *= 2;
    array->data = realloc(array->data, sizeof(double[2]) * array->capacity);
    assert(array->data != NULL);
  }
  array->data[array->elements][0] = x;
  array->data[array->elements][1] = y;
  array->elements++;
}
```

(e) Écrire la fonction `array_pop`.

```
void array_pop(Array *array, double *x, double *y) {
  assert(array->elements > 0);
  array->elements--;
  *x = array->data[array->elements][0];
  *y = array->data[array->elements][1];
}
```

(f) Écrire la fonction `array_shift`.

```
void array_shift(Array *array, double *x, double *y) {
  assert(array->elements > 0);
  *x = array->data[0][0];
  *y = array->data[0][1];
  for (size_t i = 0; i < array->elements - 1; i++) {
    array->data[i][0] = array->data[i + 1][0];
    array->data[i][1] = array->data[i + 1][1];
  }
  array->elements--;
}
```

(g) Écrire la fonction `array_unshift`.

```
void array_unshift(Array *array, double x, double y) {
  if (array->elements == array->capacity) {
    array->capacity *= 2;
    array->data = realloc(array->data, sizeof(double[2]) * array->capacity);
  }
  for (size_t i = array->elements; i > 0; i--) {
    array->data[i][0] = array->data[i - 1][0];
    array->data[i][1] = array->data[i - 1][1];
  }
  array->data[0][0] = x;
  array->data[0][1] = y;
  array->elements++;
}
```

(h) Écrire la fonction `array_insert`.

```c
void array_insert(Array *array, size_t index, double x, double y) {
  if (array->elements == array->capacity) {
    array->capacity *= 2;
    array->data = realloc(array->data, sizeof(double[2]) * array->capacity);
  }
  for (size_t i = array->elements; i > index; i--) {
    array->data[i][0] = array->data[i - 1][0];
    array->data[i][1] = array->data[i - 1][1];
  }
  array->data[index][0] = x;
  array->data[index][1] = y;
  array->elements++;
}
```

(i) Écrire la fonction `array_delete`.

```c
void array_delete(Array *array, size_t index) {
  for (size_t i = index; i < array->elements - 1; i++) {
    array->data[i][0] = array->data[i + 1][0];
    array->data[i][1] = array->data[i + 1][1];
  }
  array->elements--;
}
```

(j) Quelle est la complexité (Big-O) en temps pour l'opération *push* ?

O(1) amorti.

(k) Quelle est la complexité (Big-O) en temps pour l'opération *insert* ?

O(n)