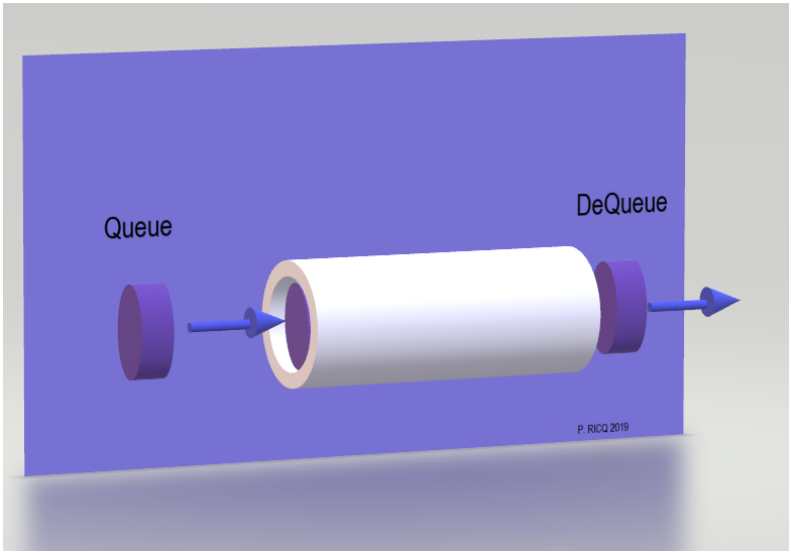
Cours d'algorithmique et langage C

TDADS1120 – Files

Une file est une structure qui permet de stocker des données.



L'accès à la file ou **Queue** en anglais, se fait par ses deux extrémités.

Deux méthodes d'accès sont possibles :

Queue : permet d'**enfiler** un élément par l'entrée, derrière le dernier élément enfilé.

DeQueue : permet de **défiler** par la sortie l'élément le plus en tête.

La méthode d'accès est connue sous l'acronyme **F.I.F.O**, First In First Out.

Cela signifie que le premier élément entré dans la pile en sortira le premier.

Cela correspond typiquement au comportement d'une file d'attente.

Les files en informatique sont des structures de données implémentées de manière à fournir ce principe de fonctionnement. On pourra par exemple y ranger des nombres entiers ou tout autre type de données.

La file peut simplement être implémentée à l'aide d'un tableau de valeurs.

Les files informatiques permettent de construire des algorithmes qui vont faciliter la résolution de problèmes spécifiques. Elles peuvent simplifier l'affichage des valeurs stockées dans un arbre par exemple, ou encore, être utilisées pour gérer des files d'attente d'impression.

En complément des fonctions Queue et DeQueue, il est intéressant de disposer d'outils complémentaires mais non indispensables comme :

isQueueFull : Qui indique si la file est pleine.

isQueueEmpty : Qui indique si la file est vide.

Afin de garantir un fonctionnement correct de la file et d'optimiser le stockage des données dans la file, on en réalise une implémentation à l'aide d'un tableau géré de manière circulaire.

Confère le cours pour le concept de tableau circulaire.

I. Etudiez le code suivant et complétez les fonctions Queue() et DeQueue()

Fichier pilestab.h

```
// implémentation des Files avec des tableaux dynamiques

typedef struct Queue {

	int *tab;		// Pointer towards dynamically allocated array
	int first;		// index of the first element in the array
	int last;		// index of the last element in the array
	int queueMaxSize;	// maximum number of elements in the Queue
	int queueNbElemt;	// current count of elements in the Queue

} Queue;

void NewQueue(Queue **queue, int initialQueueSize);
// teste si la file est vide
bool isQueueEmpty(Queue *queue);
// teste si la file est pleine
bool isQueueFull(Queue *queue);
// ajoute un élément dans la file
int queue(Queue *queue, int value);
// sort un élément de la file
int deQueue(Queue *queue, int *value);
```

Fichier piletab.c

```
void NewQueue(Queue **queue, int initialQueueSize) {
	*queue = (Queue *)malloc(sizeof(Queue));
	if (*queue != NULL) {
		(*queue)->tab = (int *)malloc(sizeof(int)*initialQueueSize);
		if (((*queue)->tab) != NULL) {
			for (int i = 0; i < initialQueueSize; i++) (*queue)->tab[i] = 0;
			(*queue)->first = -1;
			(*queue)->last = -1;
			(*queue)->queueNbElemt = 0;
			(*queue)->queueMaxSize = initialQueueSize;
		}
		else {
			free(*queue);
			*queue = NULL;
		}
	}
}

bool isQueueEmpty(Queue *queue) {
	return(queue->queueNbElemt == 0);

}
bool isQueueFull(Queue *queue) {
	return(queue->queueNbElemt == queue->queueMaxSize);
}

// Ajout d'un élément dans la file
int queue(Queue *queue, int value) { // gestion de la file en tableau circulaire
	// si la file est pleine, on ne fait rien
	if (isQueueFull(queue)) return(EXIT_FAILURE);
	// si la file est vide
	// écriture du premier élément dans la file
	if (isQueueEmpty(queue)) {
		queue->first = 0;
		queue->last = 0;
		queue->queueNbElemt = 1;
		queue->tab[queue->last] = value;
	} else {
	// écriture d'un élément dans la file


	// COMPLETER ICI
	// calculer la position du prochain élément et le stocker

	}
	return(EXIT_SUCCESS);
}

// Sortie d'un élément de la file
int deQueue(Queue *queue, int *value) {
	// COMPLETER ICI









	return(EXIT_SUCCESS);
}
```