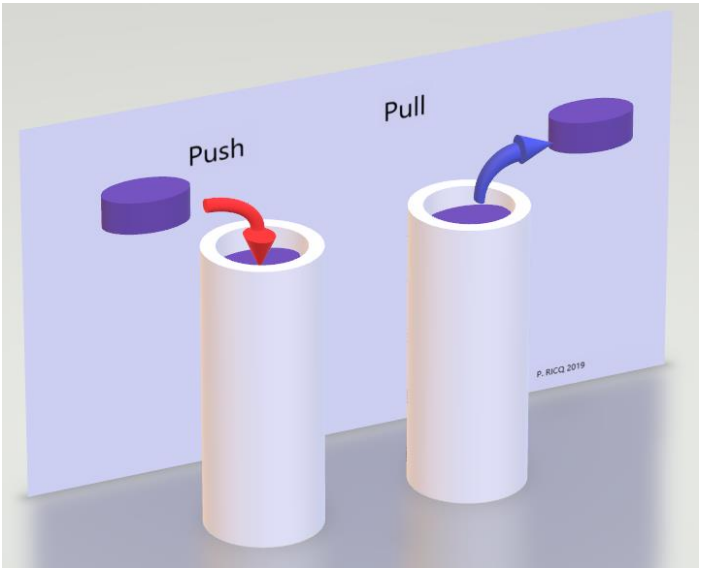
Cours d'algorithmique et langage C

TDADS1110 – Piles

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



L'accès à la pile ou **stack**, se fait par le haut.

Deux méthodes d'accès sont possibles :

Push : permet d'empiler un élément au-dessus du dernier élément empilé.

Pop ou Pull : permet de dépiler le dernier élément empilé.

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

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

Les piles 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 pile peut simplement être implémentée à l'aide d'un tableau de valeurs.

Les piles informatiques permettent de construire des algorithmes qui vont faciliter la résolution de problèmes spécifiques.

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

Peek : Qui regarde la valeur stockée au sommet de la pile sans la dépiler.

isStackFull : Qui indique si la pile est pleine.

isStackEmpty : Qui indique si la pile est vide.

I. Etudiez le code suivant et complétez les fonctions pull() et peek()

Fichier pilestab.h

```
#pragma once
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <stdbool.h>

// définition du symbole de dépassement de pile
#define STACKOVERFLOW -1

// définition d'une pile à l'aide d'une structure
typedef struct Stack {

	int *tab;          // le tableau de données qu'il faut allouer
	int stackMaxSize;  // taille max
	int stackNbElemts; // nb d'éléments actuellement dans la pile

} Stack;



// implémentation des Piles avec des tableaux dynamiques

// création d'un pile
void NewStack(Stack **stack, int initialStackSize);
bool isStackFull(Stack * stack);
bool isStackEmpty(Stack *stack);
int push(Stack *stack, int value);
int pull(Stack *stack, int *value);
int peek(Stack *stack, int *value);
```

Fichier piletab.c

```c
#include "pilestab.h"
#include <stdbool.h>

#define STACKOVERFLOW -1


// création d'une pile
// stack : nom de la pile
// initialStackSize : taille de la pile en nombre de valeurs empilables
void NewStack(Stack **stack, int initialStackSize) {
	// allocation de la structure pile
	*stack= (Stack *) malloc(sizeof(Stack));
	if (*stack != NULL) {
		// allocation du tableau de données
		(*stack)->tab = (int *)malloc(sizeof(int)*initialStackSize);
		if (((*stack)->tab) != NULL) {
			// initialisation des paramètres de la pile
			(*stack)->stackMaxSize = initialStackSize;
			(*stack)->stackNbElemts = 0;
		}
		else {
			// la pile n'a pas pu être créée, stack vaut NULL
			free(*stack);
			*stack = NULL;
		}
	}
}

// teste si la piles est pleine
bool isStackFull(Stack * stack) {
	return((stack->stackNbElemts >= stack->stackMaxSize));
};
// teste si la pile est vide
bool isStackEmpty(Stack *stack) {
	return(stack->stackNbElemts == 0);
}

// pousse une valeur sur la pile
int push(Stack *stack, int value) {
	if (!isStackFull(stack)) {
		stack->tab[stack->stackNbElemts] = value;
		stack->stackNbElemts++;
		return(0);
	}
	else {
		return(STACKOVERFLOW);
	}
}
// récupère la valeur au sommet de la pile
int pull(Stack *stack, int *value) {
	if (!isStackEmpty(stack)) {
		// COMPLETER LE CODE ICI ************************
		…
		…
		return(EXIT_SUCCESS);
	}
	return(EXIT_FAILURE);

}
```

```
int peek(Stack *stack, int *value) {
	if (!isStackEmpty(stack)) {
		// COMPLETER LE CODE ICI ************************
		…

		return(EXIT_SUCCESS);
	}
	return(EXIT_FAILURE);

}
```

II. Palindrome

Le palindrome est un mot qui peut s'écrire dans les deux sens comme : radar.

Utiliser une pile afin de vérifier qu'un mot est un palindrome.

L'idée est d'empiler toutes les lettres du mot en partant de la première.

Ensuite on dépile les lettres, c'est la dernière lettre du mot qui sort en premier de la pile. On peut donc la comparer avec la première lettre du mot.

On recommence l'opération autant de fois qu'il y a de lettres dans le mot. Si toutes les comparaisons montrent que les lettres sont identiques alors le mot est un palindrome.

Ecrire la fonction isPalindrome qui vérifie qu'un mot est un palindrome.

```
bool isPalindrome(char *word);
```