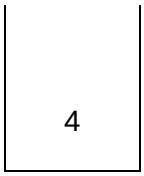
I2181A : Langage C - Modularisation (TP3)

1. Les structures récursives : la pile (LIFO)

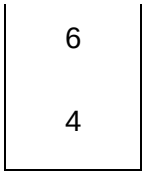
La HE Vinci voudrait un programme pour évaluer le résultat d'une expression arithmétique écrite en notation polonaise inverse. Pareille évaluation se fait aisément en utilisant une pile. Il faut parcourir l'expression : lorsqu'un nombre est rencontré, il est mis sur la pile ; lorsqu'un opérateur est rencontré, il faut dépiler deux nombres, calculer le résultat de l'opération entre le premier opérande (sous le sommet) et le second opérande (au sommet) et ensuite empiler le résultat.

Voici un petit exemple d'évaluation d'une expression arithmétique écrite en npi : **`4 6 + 3 * 8 3 - /`**

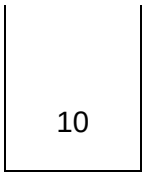
4 est mis sur la pile



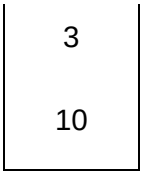
6 est ajouté au sommet de la pile



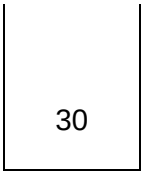
+ provoque le dépilage de 6 puis de 4 et la somme de 6 et 4 est mise sur la pile



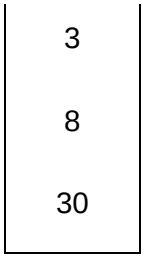
3 est ajouté au sommet de la pile



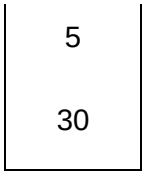
***** provoque le dépilage de 3 puis de 10 et le produit de 10 par 3 est mis sur la pile



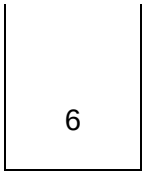
8 est ajouté au sommet de la pile, puis ***3***



- provoque le dépilage de 3 puis de 8 et la soustraction de 8 et 3 est mise sur la pile



/ provoque le dépilage de 5 puis de 30 et le quotient, résultat de 30/5 est mis sur la pile



L'expression a été entièrement prise en compte, le résultat est le nombre qui reste dans la pile.

L'équipe informatique de Vinci a écrit le programme **`npi.c`** qui évalue une expression arithmétique écrite en npi. Cependant, il manque le module pile. La HE vous confie donc la tâche d'implémenter ce module.

Voici l'**interface d'une pile** :

`Pile initPile ();`	qui renvoie une pile vide
`bool pileVide (Pile);`	qui teste si la pile est vide
`bool push (Pile*, int);`	qui renvoie vrai si l'entier a été placé sur la pile
`int pop (Pile*);`	qui retire l'entier du sommet de la pile et le renvoie
`void viderPile (Pile*);`	qui vide entièrement une pile
`void afficherPile (Pile);`	qui affiche le contenu de la pile

Schéma de la pile quand elle est vide :

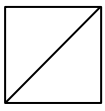


Schéma de l'ajout d'un élément au début (sommet) de la pile:

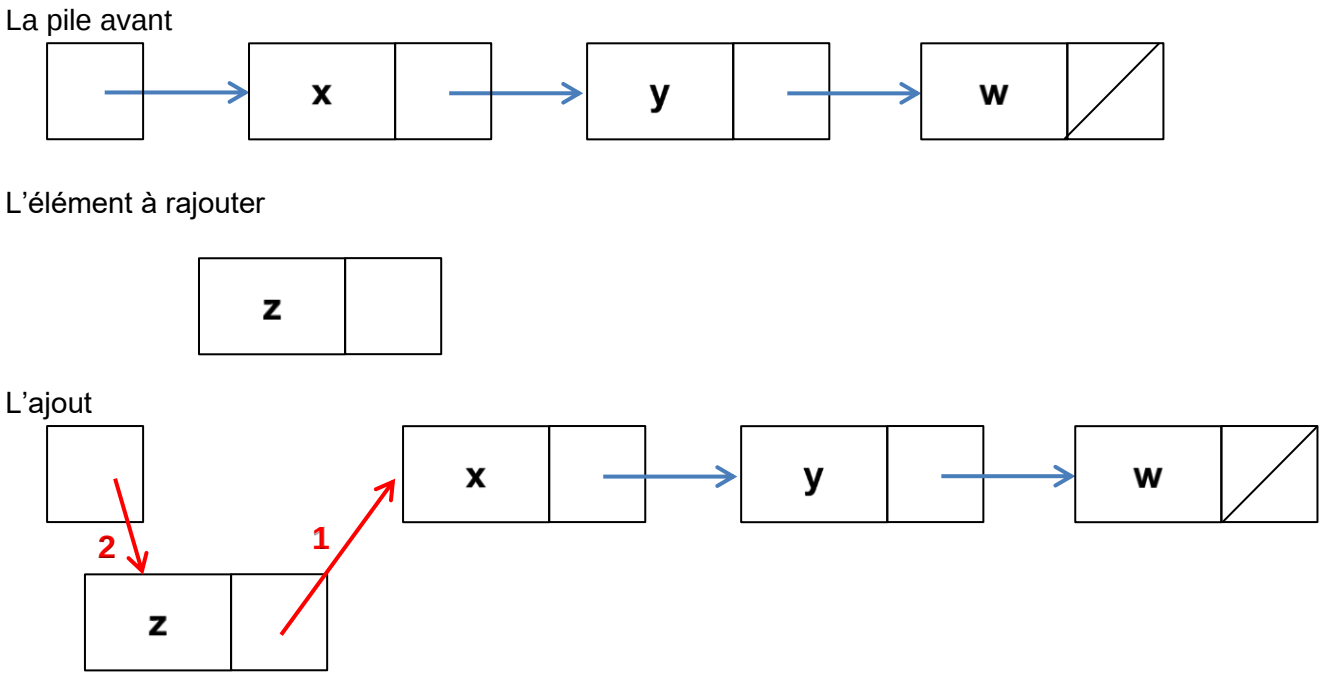
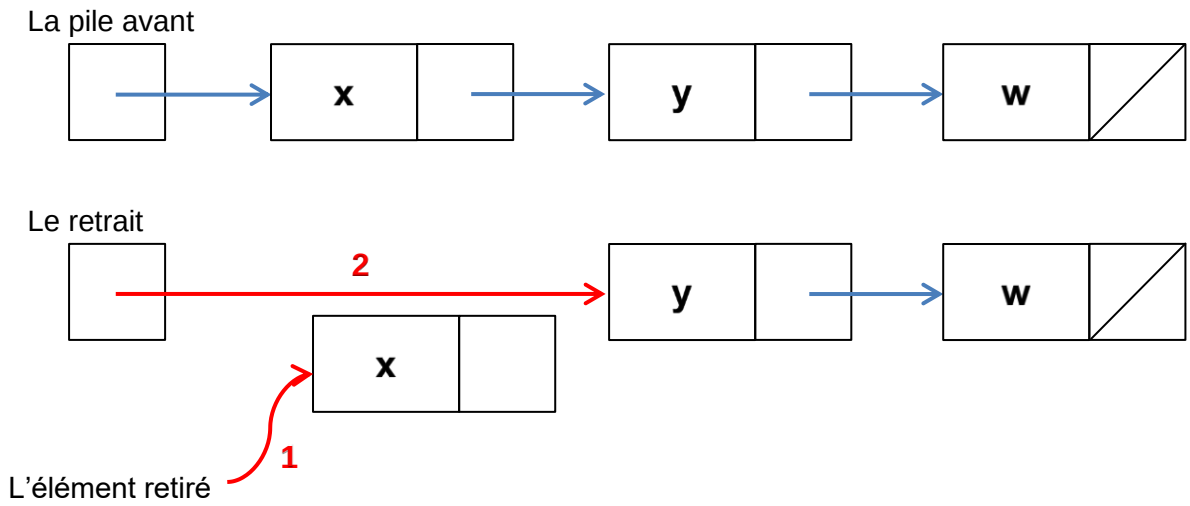


Schéma du retrait de l'élément du sommet de la pile:



1. Commencez par écrire un **module *pile***. Celui-ci contiendra les fonctions nécessaires à l'implémentation d'une pile d'entiers à l'aide une liste chaînée, en respectant l'interface fournie.

2. Testez ensuite votre implémentation de pile avec un petit programme qui **ajoute** (*push*) et **retire** (*pop*) des entiers à une pile et **affiche** son contenu après chaque opération. Pour ce faire, écrivez un ***makefile***. Utilisez *valgrind* pour vérifier si votre module gère correctement la mémoire.

3. Pour finir, testez votre module pile à l'aide du programme fourni ***npi.c*** qui implémente l'interpréteur d'expressions en notation polonaise inverse.
   - Ce programme lit à l'entrée standard des lignes d'au plus 256 caractères qui contiennent chacune une expression arithmétique en notation polonaise inverse (pour simplifier, nous supposons que les nombres sont des entiers d'un seul chiffre et les opérateurs arithmétiques sont limités aux 4 suivants : +, -, *, /).
   - Il vérifie que la ligne lue ne contient que les caractères autorisés grâce à la fonction `strspn`
   - Il évalue l'expression lue
   - Il affiche le résultat.

   Mettez à jour votre *makefile* et testez votre programme avec le fichier `npi.txt` comme suit :

   ```
   ./npi < npi.txt
   ```

   Voici un exemple d'exécution avec ce fichier de test :

```
anthony@IP-DEA044-2:solutions$ ./npi < npi.txt
  --> manque des operandes: pas de reponse
1 2 +  --> la reponse est : 3
4 5 + 9 - 7 *  --> la reponse est : 0
4 8 + 2 - 0 /  --> division par 0
5 4 +  --> la reponse est : 9
4 8 + 2 - 0 / 9 +  --> division par 0
6 2 /  --> la reponse est : 3
4 8 + 2 - 0 9 +  --> trop d'operandes: expression incorrecte
4 8 + 2 - 0 9  --> trop d'operandes: expression incorrecte
3 3 *  --> la reponse est : 9
4 5 6  --> trop d'operandes: expression incorrecte
4 2 *  --> la reponse est : 8
1 2 a 3 + +  --> expression incorrecte en a
8 4 /  --> la reponse est : 2
4 5 6 - - - - -  --> pile vide: pas de première opérande...
9 2 +  --> la reponse est : 11
- 4 5  --> pile vide: pas de seconde opérande...
5 3 *  --> la reponse est : 15
4 - 5  --> pile vide: pas de première opérande...
8 8 +  --> la reponse est : 16
```

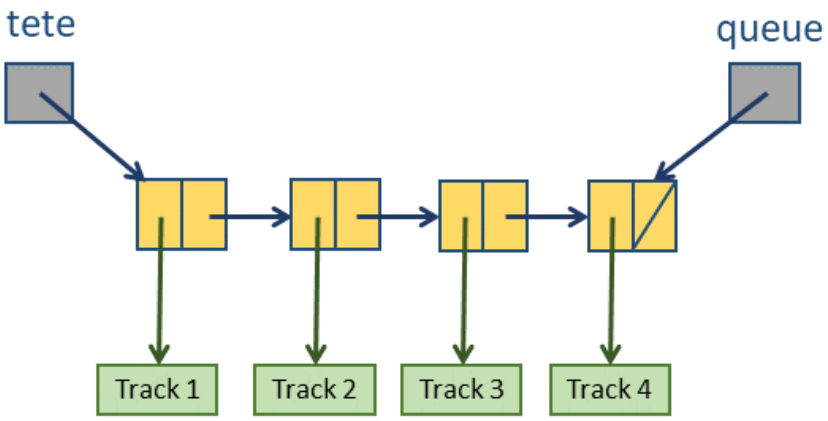
   Assurez-vous que votre programme ne provoque pas des fuites de mémoires à l'aide de **`valgrind`**.

2. Les structures récursives : la file (FIFO)

L'IPL voudrait implémenter son propre lecteur de fichiers audio. Pour ce faire, il vous est demandé d'implémenter certaines fonctionnalités permettant de gérer la liste de lecture.

Hypothèses :

- Les morceaux audio sont implémentés par un type *Track* contenant un nom d'artiste, un titre et une durée. Les champs titres et artistes sont de longueur limitée (voir fichier fourni *track.h*)
- Une liste de lecture correspond à un objet de type *Playlist* contenant deux pointeurs vers un *Noeud* : *tete* et *queue*. Dans une playlist vide, les deux pointeurs *tete* et *queue* sont NULL.
- Un *Noeud* contient un pointeur vers un *Track* (champ *track*) et un pointeur suivant (champ *svt*).
- Le pointeur *svt* du dernier élément de la liste est NULL.



Nous vous fournissons les fonctions *afficherTrack()* dans le module track et *afficherPlaylist()* dans le module playlist. Elles sont utilisées par le programme de test. Par conséquent ne les modifiez pas !

Voici un exemple d'affichage généré par la fonction *afficherPlaylist()* :

```
1 Happy Mondays - Loose Fit - 257
2 Panda Dub - Shankara - 403
3 Peter Tosh - Equal Rights - 360
4 Yves Simon - Amazoniaque - 260
```

Fonctionnalités à implémenter

Nous vous fournissons les fichiers *track.h* et *track.c*, *playlist.h* et *playlist.c* ainsi que *testplaylist.c*. Certains de ces fichiers contiennent des sections TODO à compléter.

Module Track : Définition de types

Dans *track.h*, il vous est demandé d'écrire la définition du type *Track*.

Module Track: initTrack

Cette fonction renvoie un pointeur vers une *Track* contenant la durée du morceau ainsi que la copie des chaines de caractères passées en paramètre ou NULL si la création n'a pas pu être faite. Si elles sont trop longues, ces chaînes seront tronquées à ARTISTE_LENGTH et TITLE_LENGTH.

Module Track: compareTrack

Cette fonction compare deux *Track* (cf. track.h pour les specs).

Module Playlist : Définition de types

Dans *playlist.h*, il vous est demandé d'écrire la définition des types *Nœud* et *Playlist*.

Module Playlist : initPlayList

Crée une playlist avec les pointeurs *tete* et *queue* à *NULL*.

Module Playlist : addTrack

Ajoute un morceau en queue de liste.

Module Playlist : destructiveMerge

La fonction *Playlist* destructiveMerge(Playlist* p1, Playlist* p2)* fusionne les playlists p1 et p2 en conservant l'ordre des tracks et en enlevant les doublons. De plus les listes p1 et p2 sont vidées. Elle renvoie donc une nouvelle playlist contenant les éléments des playlist p1 et p2. Après son exécution p1 et p2 sont donc vides (la tête et la queue sont mises à NULL).

On suppose que chacune des deux listes fournies est triée, ne contient pas de doublon et que la liste fusionnée ne comporte pas non plus de doublon. Par conséquent, assurez-vous qu'un morceau apparaissant dans les deux listes n'apparaîtra qu'une seule fois dans la liste renvoyée.

En cas de doublon, on libère la mémoire du *Nœud* qui n'est pas repris dans la liste fusionnée, ainsi que celle du *Track* qu'il contient.

Exemple :

```
Liste 1:
1 Happy Mondays - Loose Fit - 257
2 Panda Dub - Shankara - 403
3 Peter Tosh - Equal Rights - 360
4 Yves Simon - Amazoniaque - 260

Liste 2:
1 Eminem - Venom - 269
2 Karl Biscuit - La Morte - 236
```

```
3 Peter Tosh - Equal Rights - 360

Liste fusionnée:
1 Eminem - Venom - 269
2 Happy Mondays - Loose Fit - 257
3 Karl Biscuit - La Morte - 236
4 Panda Dub - Shankara - 403
5 Peter Tosh - Equal Rights - 360
6 Yves Simon - Amazoniaque - 260
```

Makefile

Il vous est demandé d'écrire un *makefile* avec les différentes cibles pertinentes pour ce projet.

Jeux de tests

Nous vous fournissons un programme de test *testplaylist.c* contenant quelques cas intéressants. Notez que tous les cas limites de chaque fonction ne sont pas testés par cette classe. Libre à vous de modifier ce fichier pour ajouter des tests supplémentaires.

Testez votre solution avec *valgrind*.