Jean Perrin
Faculté des Sciences
Université d'Artois

C avancé
TD n° 4

UNIVERSITÉ D'ARTOIS
Licence 3 – semestre 1
2022 – 2023

Allocation dynamique

Le but de l'exercice est de manipuler des tableaux d'entiers et de conserver deux représentations de ces tableaux : les tableaux et une liste d'occurrences qui étant donné un entier retourne la liste des tableaux où cet entier apparaît. Dans un premier temps nous souhaitons pouvoir récupérer les tableaux à partir d'un fichier. Pour cela, nous allons parser un fichier pris sur l'entrée standard. Les fichiers auront le format suivant :

```
val_abs_max
e1 e2 e3 0
e4 e5 0
e6 e7 e8 e9 0
0
```

Plus précisément, vous allez lire sur le flux de l'entrée standard à l'aide de la fonction `scanf`. La premier valeur que vous lirez sera la valeur absolue maximale que vous pourrez rencontrer dans les tableaux qui suivent. Ensuite, vous pourrez lire les tableaux qui sont représentés par une suite d'entiers terminée par 0. Nous terminons l'étape de *parsing* lorsque nous lisons un tableau de taille zéro, c'est-à-dire lorsque nous lisons deux 0 de suite. Le listing suivant représente un fichier `test.cnf` suivant ce format :

```
3
1 2 3 0
-1 2 0
-2 3 0
0
```

Afin de simplifier l'exercice, nous supposerons que le format est correct, c'est-à-dire que nous ne pouvons pas rencontrer de valeur `ei` telle que la valeur absolue de `ei` soit supérieur à la première valeur `val_abs_max` récupérée dans le fichier. De plus, nous ne pouvons pas avoir `ei` et `-ei` dans le même tableau.

Une fois que nous avons récupéré nos tableaux nous souhaitons pouvoir être mesure de les afficher, avec la liste des occurrences. Par exemple, les données récupérées du listing précédent donnerons quelque chose du genre :

```
############################
Tableaux:
[0] 1 2 3
[1] -1 2
[2] -2 3

Occurrences:
-3 ->
-2 -> 2
-1 -> 1
1 -> 0
2 -> 0 1
3 -> 0 2
```

Comme vous pouvez le voir, vous devrez associer à chaque clause un identifiant unique. Ici, nous avons choisi d'associer une valeur entière.

Nous aimerions aussi pouvoir ajouter et supprimer des tableaux en fonction de leur identifiant après avoir parser le fichier. Pour cela, vous devrez implémenter les fonctions permettant de réaliser cela. Vous devrez faire attention à ce que les listes d'occurrences soient mises à jour en conséquence. Par exemple la suppression du tableau zéro produira le résultat suivant :

```
############################
```

```
Tableaux:
[1] -1 2
[2] -2 3

Occurrences:
-3 ->
-2 -> 2
-1 -> 1
1 ->
2 -> 1
3 -> 2
```

Lorsque vous ajouterez des tableaux, vous essaierez aussi d'occuper les indices libérés par des suppressions précédentes. Par exemple, la suppression du tableau 0 suivie de l'ajout de deux tableaux `{-1, -2, -3}` et `{-1, -2}` produira le résultat suivant :

```
############################
Tableaux:
[0] -1 -2 -3
[1] -1 2
[2] -2 3
[3] -1 -3

Occurrences:
-3 -> 0 3
-2 -> 2 0
-1 -> 1 0 3
1 ->
2 -> 1
3 -> 2
```

Puisque nous sommes soucieux des ressources, vous ferez en sorte qu'à la fin de votre programme l'espace mémoire que vous avez alloué dynamiquement soit libéré. Pour vérifier que cela a été réalisé vous devrez utiliser l'utilitaire `valgrind` afin d'en avoir le cœur net. Normalement vous devrez obtenir un affichage à peu près similaire à ce qui suit :

```
$ valgrind ./exec < test.cnf
==9634== Memcheck, a memory error detector
==9634== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9634== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==9634== Command: ./exec
==9634==
==9634==

blablabla

==9634== HEAP SUMMARY:
==9634==     in use at exit: 0 bytes in 0 blocks
==9634==   total heap usage: 19 allocs, 19 frees, 19,136 bytes allocated
==9634==
==9634== All heap blocks were freed -- no leaks are possible
==9634==
==9634== For lists of detected and suppressed errors, rerun with: -s
==9634== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Vous devrez aussi faire attention au valeur de retour des fonctions d'allocation et donc vérifier si tout c'est bien passé. Si cela n'est pas le cas, vous devrez agir en conséquence afin de vous assurer que votre programme quitte proprement, c'est-à-dire que les ressources utilisées soient restituées.

Dernier point, contrairement à l'affichage d'une image, il est très difficile de vous assurer que ce que vous avez programmé est correct. Néanmoins, il est toujours possible d'écrire une petite fonction extérieure qui permet de vérifier l'état de nos structures. Identifiez ce qui peut et doit être vérifié, et implémentez les routines nécessaires pour réaliser cela.