Travaux Dirigés sur le langage *C* n°.4

Cours d'Informatique de Deuxième Année

—Deug SSM M3—

Les objectifs de ce *TD* sont les suivants :

- Étude de la représentation du type abstrait *ensemble d'entiers* (algorithmes élémentaires),
- Mise en évidence de l'importance de la nature de la représentation d'un type abstrait.

Les ensembles...

▸ **Exercice 1**. On souhaite représenter en machine le type abstrait *ensemble d'entiers*. On rappelle qu'un ensemble est une collection d'éléments distincts. Quelles structures de données pensez-vous utiliser ? Quelles seront les opérations de base qui permettront de manipuler des objets de ce type ? Comparez vos différentes idées de représentation : les critères de comparaison seront par exemple la complexité en temps des algorithmes permettant de manipuler ces ensembles, la complexité en espace ou encore la "simplicité de programmation".

On choisit de représenter le type abstrait "ensemble d'entiers" par le type *C* "tableau **triés** d'entiers". On donnera à chaque fois, la complexité de l'algorithme implémenté.

▸ **Exercice 2**. Écrire une fonction de nom `PosIterative` prenant en argument un ensemble `Ensemble`, le nombre d'éléments `NbElem` de cet ensemble et un entier `E`. Si l'entier `E` est bien dans l'ensemble `Ensemble`, la fonction retournera la position à laquelle il est stocké, et dans le cas contraire, la fonction retournera un nombre négatif dont la valeur absolue sera interprétée comme la position qu'occuperait l'entier `E` si on devait le stocker dans l'ensemble `Ensemble`.

▸ **Exercice 3**. On demande dans cet exercice d'écrire une fonction de nom `Decale` prenant en argument un tableau d'entiers `t` et trois entiers `Debut`, `Fin` et `d`. Cette fonction devra décaler la portion de tableau comprise entre les indices `Debut` et `Fin` compris, du déplacement `d`, où `d` sera positif (décalage à droite) ou négatif (décalage à gauche).

▸ **Exercice 4**. Déduire des fonctions définies dans les exercices précédents une fonction de nom `Insere`, prenant en argument un ensemble `Ensemble`, un entier `E` et le nombre d'éléments `NbElem` de cet ensemble. La fonction devra insèrer l'élément `E` dans l'ensemble `Ensemble`. Elle pourra par exemple retourner le nombre d'éléments de l'ensemble après l'insertion.

▸ **Exercice 5**. En s'aidant encore une fois des fonctions déjà écrites dans les exercices précédents, on demande d'écrire une fonction de nom `Supprime`, prenant en argument un ensemble `Ensemble`, un entier `E` et le nombre d'éléments `NbElem` de cet ensemble. La fonction devra cette fois supprimer l'élément `E` de l'ensemble `Ensemble` et comme précédemment, on pourra retourner la position de `E` avant la suppression, ou bien si l'élément `E` n'était pas dans l'ensemble `Ensemble`, la position à laquelle il aurait dû être.

▸ **Exercice 6**. Écrire une fonction de nom `Union`, prenant en argument un ensemble résultat `Ensemble`, deux ensembles sources `Src1` et `Src2`, et leurs nombres respectifs d'éléments `NbSrc1` et `NbSrc2`. La fonction devra réaliser l'union des deux ensembles `Src1` et `Src2`, et stocker le résultat dans l'ensemble `Ensemble`. Elle retournera utilement le nombre d'éléments contenus dans l'union.

▸ **Exercice 7**. Écrire une fonction de nom `Intersection`, prenant en argument un ensemble résultat `Ensemble`, deux ensembles sources `Src1` et `Src2`, et leurs nombres respectifs d'éléments `NbSrc1` et `NbSrc2`. La fonction devra réaliser l'intersection des deux ensembles `Src1` et `Src2`, et stocker le résultat dans l'ensemble `Ensemble`. Elle retournera utilement le nombre d'éléments contenus dans l'intersection.