Université de la Nouvelle-Calédonie
Licence 1ère année – M, PC et SPI

Algorithmique et Programmation 1
2015

Projet
(à rendre pour le 29 mai 2015)

Dans les exercices suivants, il est demandé d'écrire les programmes demandés en Langage C. Il est recommandé de ne pas oublier la description de l'algorithme utilisé et de l'environnement associé.

Vous rendrez un dossier papier contenant les algorithmes avec leur environnement, les programmes C imprimés et des jeux d'essais (exemple d'exécution du programme) obtenus par recopies d'écran (recopie de l'image de la fenêtre active par ALT+ImpEcr).

Même si seules des procédures sont demandées, il faudra évidemment écrire le programme principal pour utiliser ces procédures.

1/ Intervalles de réels

(extrait des examens 1ère et 2ème session 2014)

On désire stocker des intervalles mathématiques (source des illustrations Wikipédia) :

Initialement, on appelle **intervalle réel** un ensemble de nombres délimité par deux nombres réels constituant une borne inférieure et une borne supérieure. Un intervalle contient tous les nombres réels compris entre ces deux bornes.

Cette définition regroupe les intervalles des types suivants (avec $(a, b) \in \mathbb{R}^2,\ a < b$)

- $\{x \in \mathbb{R} \mid a < x < b\} =]a; b[$ (ouvert et non fermé)
- $\{x \in \mathbb{R} \mid a \leq x \leq b\} = [a; b]$ (fermé et non ouvert)
- $\{x \in \mathbb{R} \mid a < x \leq b\} =]a; b]$ (ni ouvert, ni fermé)
- $\{x \in \mathbb{R} \mid a \leq x < b\} = [a; b[$ (ni ouvert, ni fermé)

Les intervalles du premier type sont appelés **intervalles ouverts** ; les seconds **intervalles fermés**, et les deux derniers **intervalles semi-ouverts**.

a) Écrire un type pour stocker un intervalle. Il faudra mémoriser les bornes mais aussi savoir si elles sont incluses ou pas.

Remarque : pour l'instant, on ne tiendra pas compte des bornes infinies.

b) Un intervalle peut être l'ensemble vide. En utilisant le type préalablement défini, proposez une manière de stocker l'intervalle vide.

Écrire une fonction booléenne qui rend vrai si l'intervalle passé en paramètre est vide, faux sinon.

```
Fonction EstVide(A) : Booléen
{qui rend Vrai si l'intervalle A est vide, Faux sinon}
```

c) Écrire une procédure AfficherIntervalle(A) qui affiche à l'écran l'intervalle A donné en paramètre sous les formes indiquées ci-dessus (c'est à dire [inf ; sup] et toutes les autres variantes).

On suppose qu'on pourra utiliser les caractères spéciaux (∅ et ∞ si besoin cf. f/) ainsi que la fonction EstVide ci-dessus.

d) Écrire une procédure pour faire l'intersection de 2 intervalles.

Une intersection d'intervalles de $\mathbb{R}$ est toujours un intervalle. Par exemple,

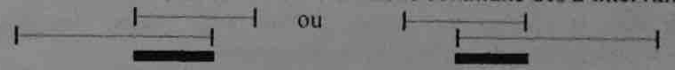
- $[-3; 5[\cap] -\infty; 2] = [-3; 2]$
- $[-3; 5[\cap[2; +\infty[= [2; 5[$
- $[3; 5[\cap] -\infty; 2] = \varnothing$

pour rappel, 3 cas possibles :

- les intervalles sont disjoints, l'intersection est vide,

- un intervalle est inclus dans l'autre, l'intersection est le premier intervalle,

- dans les autres cas, l'intersection est la zone commune des 2 intervalles.



e) Union de 2 intervalles :

Une union d'intervalles de $\mathbb{R}$ n'est pas toujours un intervalle. Ce sera un intervalle si l'ensemble obtenu reste convexe (intuitivement s'il n'y a pas de "trou"). Dans le cas d'une union de deux intervalles, il suffit que l'intersection de ces intervalles soit non vide pour que leur réunion soit convexe. Par exemple,

- $] -\infty; 2] \cup [-3; 5[=] -\infty; 5[$
- $[-3; 5[\cup[2; +\infty[= [-3; +\infty[$
- $[3; 5[\cup] -\infty; 2] =] -\infty; 2] \cup [3; 5[$ (On préfère ranger les intervalles par ordre croissant de leurs bornes.)

Cette union ne forme pas un intervalle étant donné qu'il y a un trou entre 2 et 3.

En utilisant les procédures et fonctions précédentes (Intersection et EstVide), écrire une procédure pour faire l'union de 2 intervalles dans le cas où le résultat est un intervalle c'est à dire quand il n'y a pas de « trou » donc que l'intersection est non vide, dans le cas contraire, on donnera un message d'erreur.

f) Questions subsidiaires sur les bornes infinies (points bonus)

Que pouvez-vous proposer pour représenter les intervalles utilisant l'infini ($-\infty$ ou $+\infty$) ? Le type ci-dessus peut éventuellement être modifié pour répondre à la question.

Quels sont les changements à effectuer sur les procédures précédentes ? Les effectuer si vous en avez le temps.

Remarque : en attendant de voir les types composés en Cours, TD et TP, vous pouvez raisonner avec des variables pour représenter un intervalle et pour commencer à écrire les algorithmes.

2/ Tri par comptage

(extrait de l'examen de 1ère session 2014)

Soit T un tableau de M entiers compris entre 1 et N.

Le tri par comptage ne fonctionne que sur des entiers compris entre 1 et N, N étant connu à l'avance, contrairement à d'autres tris qui s'appliquent à tous les types de données. Il est basé sur le comptage du nombre d'occurrences (d'apparitions) des valeurs à trier.
La figure ci-dessous présente son fonctionnement :

- une première boucle initialise un tableau « compteurs » à 0 ;
- une deuxième boucle compte les occurrences de chaque donnée à trier et range le résultat dans le tableau « compteurs » à l'indice indiqué par l'entier à trier (étape 1).
  Exemple : Si l'entier 12 est rencontré deux fois, la valeur 2 est rangée à la case d'indice 12 du tableau « compteurs ».
  A la fin de cette boucle, le tableau « compteurs » contient le nombre d'occurrences pour chacun des entiers.
- une troisième boucle le transforme pour qu'il indique, pour chaque entier, l'ordre de rangement dans le tableau trié (étape 2). Pour cela, chaque case du tableau compteurs est augmentée du contenu de la case précédente.
- une quatrième boucle range les entiers dans un tableau temporaire selon l'ordre indiqué par compteurs (étape 3).

Exemple : Ainsi, l'entier 4 contenu dans la case 2 du tableau T est en quatrième position (position indiquée par la case 4 de compteurs). Il doit être rangé dans la case 4 du tableau « TMP ». Pour gérer les occurrences multiples, à chaque fois qu'un entier est rangé dans le tableau trié, le contenu de la case qui lui correspond dans compteurs est diminué de 1. Ainsi, la case 4 de « compteurs » contenait la valeur 4 ; après le rangement de la première occurrence de la valeur 4 dans « TMP », elle passe à 3. Quand l'entier 4 est de nouveau rencontré, il est rangé en troisième position (valeur de la case 4 de « compteurs »)donc dans la case 3 de « TMP ». Ainsi, tous les entiers contenus dans « T » sont rangés dans l'ordre dans « TMP », grâce au tableau « compteurs » ;

- la dernière boucle recopie « TMP » dans « T ».

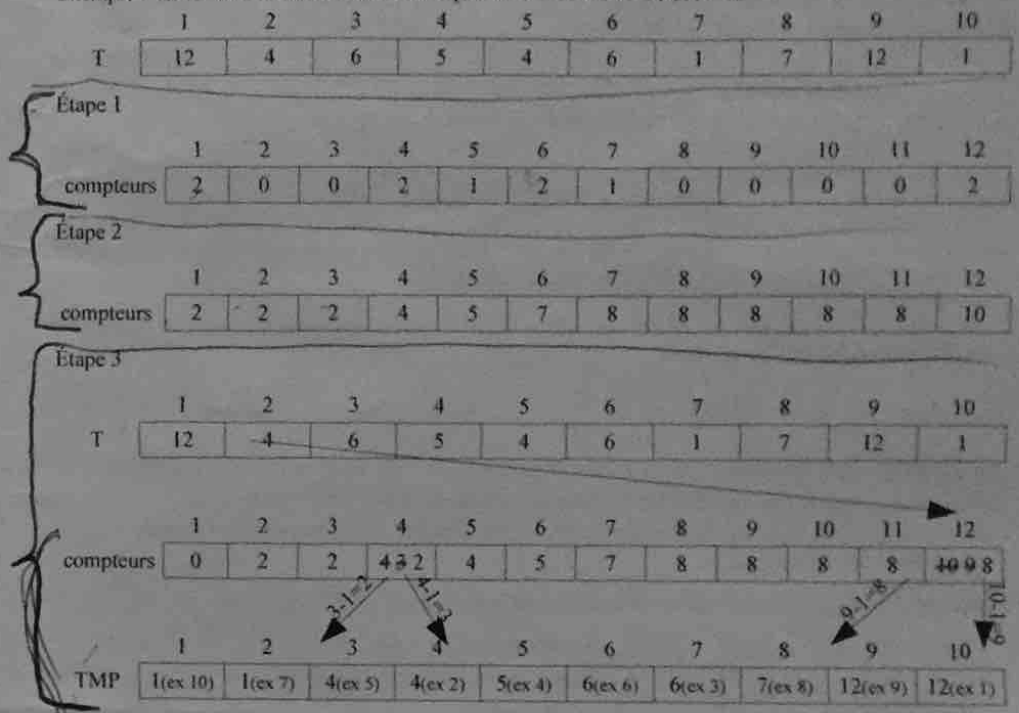
Const N..
type TABENT : ta. 1 a N EN
TTMP ..
Indice : 0 . . N+1

T[i] = T[i+1]

T[1] = T[1+1]

12 =

Exemple : Tri d'un tableau de M entiers compris entre 1 et N ; M=10 et N=12



a) Écrire les types nécessaires pour ces traitements
b) Écrire la procédure Tri :
   - Écrire l'entête de la procédure Tri et son environnement
   - Écrire les différentes parties de la procédure Tri (notées séparément) correspondant aux différentes étapes décrites ci-dessus :

```
Début
{Initialisation de compteurs}
...
{Comptage dans compteurs du nombre d'apparition de chaque valeur de T}
...
{Calcul dans compteurs de l'ordre de rangement }
...
{Tri de T dans TMP}
...
{Recopie de TMP dans T}
...
Fin
```

Question subsidiaire : Quel est l'ordre de l'algorithme ? En nombre de comparaisons ? En nombre d'affectation ?