

PL1 - Travaux pratiques - Séance 6 (3h)

Type Séquences : contiguïté et chaînage

Avant de commencer cette séance :

1. créez un répertoire **PL1/TP6** et placez-vous dans le répertoire
2. récupérez les fichiers nécessaires à ce TP : `cp ~mounlaur/CCI_PL1/TP6.tar.gz .`
3. dé-compressez et dé-archiver ce fichier : `gunzip TP6.tar.gz ; tar -xvf TP6.tar`

On souhaite implémenter de deux manières différentes un type “séquence d’entiers” ainsi que des actions et fonctions portant sur ce type (de sorte que ces deux implémentations soient ré-utilisables dans la suite du semestre). On définit donc le lexique suivant :

SeqInt : type Séquence d’entiers

LireSeq : action (resultat s : SeqInt)
 {lit au clavier un entier n, puis n entiers et les mémorise dans s}

EcrireSeq : action (donnée s : SeqInt)
 {affiche la séquence s à l’écran}

EstPresent : fonction (donnée s : SeqInt, donnée x : entier) → booléen
 {EstPresent(s,x) vaut vrai ssi x appartient à s}

Inserer : action (donnée-résultat s : SeqInt, donnée x : entier)
 {Inserer (s, x) insere x dans la sequence s (à une position quelconque)}
 {s est inchangée si l’insertion ne peut avoir lieu (par exemple si s est “pleine”)}

Supprimer : action (donnée-résultat s : SeqInt, donnée x : entier)
 {Supprimer (s) supprime de s une occurrence de l’élément x (s est inchangée si elle ne contient pas x)}

Exercice 1 - Représentation contigüe

Q1. Votre répertoire contient un fichier `sequence.h` qui contient une traduction en C du type `SeqInt` et des en-têtes des fonctions et actions proposées ci-dessus.

Ecrivez dans un fichier `sequence.c` le corps de ces fonctions et actions. N’oubliez pas d’inclure le fichier `sequence.h` en tête de `sequence.c` (ainsi que d’autres fichiers `.h` le cas échéant ...).

Q2. Votre répertoire contient également un fichier `main.c` qui contient un exemple de programme principal.

Ecrivez un fichier `Makefile` pour produire par compilation séparée un exécutable de nom `test_sequence` à partir des fichiers `sequence.h`, `sequence.c` et `main.c`.

Q3. Compilez et exécutez `test_sequence` pour tester votre implémentation de `sequence.c`. Vous pouvez compléter `main.c` pour renforcer vos tests.

Exercice 2 - Représentation chaînée

On propose maintenant de modifier l'implémentation du type `SeqInt` au profit d'une solution par chaînage à l'aide de pointeurs.

Q1. Copiez les fichiers `sequence.h` et `sequence.c` en fichiers `sequence_contigue.h` et `sequence_contigue.c` (pour les sauvegarder).

Modifiez alors le fichier `sequence.h` pour que `SeqInt` soit défini par chaînage à l'aide de pointeurs.

Q2. Modifiez le fichier `sequence.c` pour l'adapter à cette nouvelle représentation. Est-ce que la fonction `Inserer` conserve un comportement similaire à la version écrite dans l'exercice 1 ?

Q3. Compilez à nouveau `test_sequence` pour tester votre nouvelle implémentation de `sequence.c`. Vous pouvez là encore compléter `main.c` pour renforcer vos tests.

Exercice 3 - Séquences ordonnées

On se place maintenant dans le cas où les séquences considérées sont supposées être ordonnées (croissantes), toujours en considérant une implémentation par chaînage à l'aide de pointeurs.

Q1. Après avoir sauvegardé le fichier `sequence.c`, modifiez-le pour prendre en compte cette hypothèse.

Q2. Ré-écrivez le fichier `main.c` pour tester cette nouvelle version.

Q3 (*). Ajoutez une implémentation de l'action suivante :

Trier : action (la donnée `s1` : `SeqInt` ; le résultat `s2` : `SeqInt`)
{Trier(`s1`) produit dans `s2` un tri de la séquence `s1`}.

Testez votre implémentation ...

Q4 ()**. Ajoutez enfin une implémentation de l'action suivante :

Trier2 : action (la donnée-résultat `s` : `SeqInt`)
{Trier2(`s`) trie la séquence `s`}.

Testez votre implémentation ...