Département
Informatique

TP C++
Classe Simple

Dossier de Spécification

Binôme 3213

Tristan Pourcelot
Jordan Vincent

3IF - Groupe 2

Année scolaire 2011-2012

Institut National des Sciences Appliquées de Lyon

Première partie
Spécifications générales du programme

La classe IntervalSet a pour rôle principal de gérer un ensemble d'intervalles

Deuxième partie
Définitions Globales

1. définition d'un intervalle
   Un intervalle est un ensemble défini par ses deux bornes borne_inf et borne_sup. Ces deux bornes sont de type `long signed int`.
   D'autre part on a bien évidemment $borne_sup \geq borne_inf$
   (ie -> on accepte un intervalle composé d'un seul élément t.q. borne_sup = borne_inf)
   De même, il faut noter qu'il est impossible de traiter un intervalle infini.
2. Définition d'un ensemble d'intervalle

Un ensemble d'intervalles est un groupe d'intervalles disjoints, çad qu'ils n'ont aucun élément en commun. Ainsi, deux intervalles [borne_inf1 , borne_sup1] et [borne_inf2 , borne_sup2] t.q. borne_sup1 = borne_inf2 ne sont pas disjoints !

Troisième partie
Rôle de la classe

La classe IntervalSet a pour rôle principal de gérer un ensemble d'intervalles On peut ajouter un intervalle ou un ensemble d'intervalles à la classe. D'autre part, cet ensemble d'intervalles est trié dans l'ordre croissant des valeurs inférieures. La classe permet d' afficher sur la sortie standard la liste des intervalles qu'il contient et leur nombre. A chaque intervalle de la classe corespond un indice. Cet indice permet de repérer de manière unique un intervalle de l'ensemble. Cet indice permet de sélectionner ou de supprimer un intervalle de l'ensemble. On peut effectuer des opérations ensemblistes entre deux sets d'intervalles (union et intersection)

- Caractéristiques d'un objet de notre classe
  - Notre classe contient une collection triée de structures `Interval` contenant l'ensemble des intervalles
  - Un nombre d'index `interval_count` qui répertorie la longueur de la liste d'intervalles
  - FACULTATIF : Un attribut `etendue` de type `Interval` qui recense l'étendue de la collection d'intervalle
- Spécifications de l'interface publique
  - Constructeur `IntervalSet()`
    Ce constructeur ne prends aucun argument, et ne fait qu'initialiser l'espace mémoire pour la structure

  - Constructeur de recopie `IntervalSet(Bordel...)`
    Ce constructeur prends en argument un pointeur vers un IntervalSet existant, et recopie la structure pointée dans un nouvel espace qu'il alloue.

  - Procédure d'affichage `Display`
    Cette procédure ne prends pas d'argument. Elle affiche les différents intervalles de la collection, en partant du premier (borne_inf la plus petite), au plus grand (borne_sup la plus grande).
    La sortie attendue est :
    `"L'intervalle numéro 0 commence à borne_inf_0 et se termine à borne_sup_0."` ...

`"L'intervalle numéro N commence à borne_inf_n et se termine à borne_sup_n."`

- Fonction booléen : `AddInterval(Interval)`
  Cette méthode prends en argument un intervalle de type `Interval` et renvoie un booléen signifiant le succès ou non de l'ajout de l'intervalle à la collection. Elle teste la validité de cet intervalle (disjonction avec la collection déjà existante) puis, si ce test est correct, l'ajoute à l'ensemble à la bonne position (respect de l'ordre). Si jamais l'ensemble n'est pas disjoint de la collection, il n'est pas ajouté à l'ensemble et la méthode retourne la valeur FALSE. Contrat : l'argument Interval doit être défini de manière correcte (voir doc du type Interval)
- Fonction booléen : `AddIntervalSet(IntervalSet)`
  Cette méthode prends en argument un ensemble d'intervalles de type `IntervalSet` et renvoie un booléen signifiant le succès ou non de l'ajout de la collection d'intervalles. Elle teste la validité de chaque intervalle (disjonction avec la collection déjà existante) puis, si tout les tests sont corrects, on ajoute un à un les différents intervalles contenus dans l'ensemble passé en argument. Si jamais au moins un intervalle n'est pas disjoint de la collection, aucun intervalle n'est ajouté, et la méthode retourne la valeur FALSE.
- Fonction Entier : `Count()`
  Cette méthode renvoie le nombre d'intervalles contenus dans l'ensemble, et 0 si l'ensemble est vide.
- Fonction Interval : `GetInterval( Entier indice )`
  Cette méthode renvoie une structure du type `Interval` dont l'indice dans la structure est passé en argument.
  CONTRAT : Cet indice doit être compris entre 0 et IntervalSet.Count().
  Si la collection de IntervalSet est vide (IntervalSet.Count=0), cette méthode renvoie l'ensemble [0,0]
- Fonction Booléen : `Remove(Entier indice)`
  Cette méthode prends en argument un entier `indice` et supprime l'intervalle dont l'index dans la collection correspond à l'indice passé en argument.
  CONTRAT : Cet indice doit être compris entre 0 et IntervalSet.Count().
  En cas de succès, cette méthode renvoie la valeur `TRUE`.
  En cas d'échec (indice > IntervalSet.count, erreur lors de la suppression...), cette méthode renvoie FALSE.
- Fonction IntervalSet : `Union(IntervalSet a_intervalle)`
  Cette méthode renvoie la réunion d'intervalle de deux ensembles.
  Elle prend comme argument une collection d'intervalles de type IntervalSet. On désigne S la collection d'intervalle de sortie, A la collection passée en argument, et C la collection propre de l'objet.
  On teste les différents intervalles de A et de C, afin de trouver les points communs (jointures), et si des intervalles se recoupent, on les fusionnent. On ajoute ensuite cette collection d'intervalles fusionnés et disjoints à la collection S, avant de la retourner.
- Fonction IntervalSet : `Intersection(IntervalSet a_intervalle`
  Cette méthode renvoie l'intersection entre deux ensembles d'intervalles.
  Elle prend comme argument une collection d'intervalles de type `IntervalSet`. On désigne S la collection d'intervalle de sortie, A la collection passée en argument, et C la collection propre de l'objet.
  On teste les différents intervalles de A et de C, afin de trouver les points communs (jointures), et si des intervalles se recoupent, on ne sélectionne que l'intersection. On ajoute ensuite cette collection d'intervalles à la collection S, avant de la retourner.