ICC (SMA/SPH) / DEVOIR NOTÉ : Discussions on-line

Introduction et instructions générales

On s'intéresse ici à écrire un programme de gestion de discussions on-line (genre « tchat-room » privée).

Nous vous demandons d'écrire tout votre code dans un seul fichier nommé `msg.cc` sur la base d'un fichier fourni, à compléter en utilisant le langage C++ et une approche « procédurale » avec le moins de duplication de code possible.

Instructions :

1. Cet exercice doit être réalisé **individuellement !** L'échange de code relatif à cet exercice noté et l'utilisation d'outil automatique pour le réaliser sont **strictement interdit !** Cela inclut également la diffusion de code sur le forum du cours ou sur tout site de partage.

   Le code rendu doit être le résultat de *votre propre production*.
   Le plagiat de code, de quelque façon que de soit et quelle qu'en soit la source, sera considéré comme de la tricherie.

   En cas de tricherie, vous recevrez la note « NA » pour l'entièreté de la branche et serez de plus dénoncé(e) et puni(e) suivant l'ordonnance sur la discipline.
2. Vous devez donc également garder le code produit pour cet exercice dans un endroit à l'accès strictement personnel.

   Le fichier (source !) `msg.cc` à fournir comme rendu de ce devoir ne devra plus être modifié après la date et heure limite de rendu.
3. Veillez à rendre du code *anonyme* : **pas** de nom, ni de numéro SCIPER, ni aucun identifiant personnel d'aucune sorte !
4. Utilisez le site Moodle du cours pour rendre votre exercice.

   Vous avez jusqu'au **mercredi 29 novembre, 23h59** (heure du site Moodle du cours faisant foi) pour soumettre votre rendu.
   Aucun délai supplémentaire ne sera accordé.

Indication : Si un comportement ou une situation donnée n'est pas définie dans la consigne ci-dessous, vous êtes libre de définir le comportement adéquat. On considérera comme comportement adéquat toute solution qui ne viole pas les contraintes données et qui ne résulte pas en un crash du programme.

Travail à faire

Une start-up désire offrir un nouveau système de discussion on-line (« tchat »). Celui-ci est composé de client(e)s possédant chacun(e) un nom, un genre et un pseudonyme.

Chaque usager peut avoir à un même instant un nombre quelconque de discussions en cours. Chaque usager se verra donc associer la liste des autres personnes auxquelles il envoie des messages et (séparément) la liste des autres personnes qui lui adressent des messages.
(Voir l'exemple de déroulement à la fin.)

Dans le fichier `msg.cc` fourni (respectez strictement ce nom), on vous demande :

1. d'implémenter au moins [1] un type de données adéquat permettant de représenter un(e) client(e) (et ses interlocuteurs/interlocutrices en cours), ainsi qu'un type de données permettant de représenter le système (ensemble des client(e)s) ;

   (conseil : lire totalement la donnée avant de commencer ;)

   le genre sera représenté par un seul caractère : 'F' pour féminin, 'M' pour masculin et 'X' pour autre ou non précisé ;

2. d'implémenter au moins [1] les fonctions suivantes (voir des exemples dans le `main()` fourni) :

   — `add()` qui crée une nouvelle personne cliente et l'ajoute au système ;

   les différents champs nécessaires (p.ex. nom, genre, ...) seront bien sûr fournis comme arguments à cette fonction ;

   le pseudo sert d'identifiant unique ; cette fonction ne fait donc pas l'ajout, mais affiche un message d'erreur, si le pseudo fourni est déjà présent dans le système ; (voir l'exemple de déroulement à la fin ;)

   de même, on ne doit pas accepter de genre autre que 'F', 'M' ou 'X' ;

   — `chat()` qui ouvre une discussion entre un(e) client(e) donné(e) et un(e) autre client(e) spécifié(e) par son pseudo ; c.-à-d. met à jour la liste des interlocuteurs/interlocutrices des client(e)s concerné(e)s ;

   attention à ne pas ajouter un(e) interlocuteur/interlocutrice déjà existant(e) ; (voir l'exemple de déroulement à la fin ;)

   notez bien que la personne qui est contactée a aussi une discussion ouverte avec l'appelant (on supposera, pour simplifier ici, que l'appelé accepte toujours une discussion demandée) ;

   — `state()` qui affiche l'état courant du système, c.-à-d. qui, pour tou(te)s les client(e)s, affiche la liste complète de ses interlocuteurs en cours (nom et pseudonyme) ;

   on devra distinguer les appelés (en affichant un « `->` » devant) des appelants (identifiés par un « `<-` » devant) ; voir l'exemple de déroulement à la fin.

Notes :

1. Il n'y a *rien* à faire saisir au clavier par l'utilisateur ; tout est fourni directement dans le code source comme donné en exemple dans le `main()` fourni ;
2. le `main()` fourni n'est qu'un *exemple* possible ; vous pouvez le modifier à votre guise, **mais** ce `main()` fourni doit pouvoir compiler **en l'état** (= sans modification) avec votre code produit.

suite au dos ☞

1. Vous êtes libre d'ajouter vos propres types de données ou fonctions intermédiaires si nécessaire.

Exemple de déroulement

Voici ce que le `main()` fourni doit produire :

```
Nouveau client : Paul Hice - interPaul
Nouvelle cliente : Sylva Froid - sylv21
Nouvelle cliente : Melusine Titgoute - kerrig@n
ERREUR: pseudo "sylv21" déjà présent
ERREUR: genre 'Y' non valide
Nouveau/Nouvelle client(e) : Bob Razowsky - bobichon

Paul Hice contacte serv@l
 ==> ERREUR : PAS DANS LE RESEAU
Paul Hice contacte sylv21
 ==> OK (Sylva Froid)
Paul Hice contacte kerrig@n
 ==> OK (Melusine Titgoute)
Paul Hice contacte kerrig@n
 ==> ERREUR : APPEL DEJA FAIT
Sylva Froid contacte interPaul
 ==> ERREUR : COMMUNICATION DEJA EXISTANTE
Melusine Titgoute contacte sylv21
 ==> OK (Sylva Froid)
Melusine Titgoute contacte bobichon
 ==> OK (Bob Razowsky)

Discussions de Paul Hice :
  -> sylv21 (Sylva Froid)
  -> kerrig@n (Melusine Titgoute)
Discussions de Sylva Froid :
  <- interPaul (Paul Hice)
  <- kerrig@n (Melusine Titgoute)
Discussions de Melusine Titgoute :
  -> sylv21 (Sylva Froid)
  -> bobichon (Bob Razowsky)
  <- interPaul (Paul Hice)
Discussions de Bob Razowsky :
  <- kerrig@n (Melusine Titgoute)
```