IN3S02 : Sujet du TP C++

Groupe ESIEE, Denis BUREAU, novembre 2004.
Attention ! Le sujet peut être modifié jusqu'à la veille du TP.

1 Les objectifs

- Savoir éditer/compiler/exécuter un programme donné avec l'environnement de programmation Visual C++.
- Écrire des programmes simples en C++.
- Réaliser sur machine les exercices du TD.
- Expérimenter la liaison Java $\leftrightarrow$ C++.

2 Visual C++

1. Lancer Visual C++
2. Passer en plein écran (si ce n'est pas déjà le cas)
3. Clicker sur l'icone la plus à gauche (nouveau fichier).
4. Taper dans la fenêtre d'édition la première ligne d'un programme C++ : `// Bonjour.CPP`
5. Sauvegarder ce programme dans le fichier Bonjour.CPP dans le répertoire `C:\TEMP\TPCPP` pour que la syntaxe C++ soit mise en couleurs au cours de la frappe.
6. Finir de taper le programme de la question 3 du TD.
7. Sauvegarder ce programme (toujours dans le fichier Bonjour.CPP)
8. Manipuler les différents menus pour connaître les différentes possibilités
9. Compiler le programme : (Compile dans le menu Build)
   Répondre OUI à la question.
   - En cas d'erreur de syntaxe, les messages d'erreur apparaissent dans la fenêtre tout en bas (faire défiler vers le haut et double-cliquer sur chaque erreur)
   - En cas de succès, Bonjour.OBJ est créé dans le sous-répertoire `DEBUG` (le vérifier sous DOS)
10. Faire l'édition de liens (link) par la commande Build ; Bonjour.EXE est créé (le vérifier sous DOS)
11. Lancer l'exécution (Execute dans le menu Build).
12. Vérifier que l'on peut aussi lancer l'exécution sous DOS en tapant simplement `Bonjour` (attention au répertoire courant).
13. Ajouter au programme la saisie d'un caractère et son affichage aprs `"Bonjour"`.
14. Sauvegarder et lancer Build
15. Lancer l'exécution
16. **Fermer l'espace de travail avant de commencer un nouvel exercice** (Close workspace dans le menu File; répondre OUI à la question)
17. Ne conserver que les .CPP (et, quand ils sont utiles, les .EXE) et recopiez le répertoire TPCPP de `C:\TEMP` dans `U:\TPCPP`.

3 Exercices du TD (1h max)

1. Passage de paramètres
2. Chaînes de caractères
3. Pointeurs et chaînes de caractères
4. Pointeurs et allocation dynamique

4 Listes chaînées (2h max)

1. Déclarer un type `TypElement` pour pouvoir ensuite changer aisément ce que l'on stockera dans chaque maillon; dans un premier temps, on pourra décider par exemple que `TypElement` est juste un `double`.
2. Déclarer un type `TypMaillon` qui contient un champ `element` de type `TypElement` et un champ `suivant` de type pointeur de `TypMaillon`.
3. Déclarer un type `TypPtrMaillon` (pointeur de maillon).
4. Déclarer un type `TypListe` (aussi un pointeur de maillon) juste pour pouvoir distinguer un pointeur de maillon en général (qui peut pointer sur n'importe quel maillon d'une liste) du pointeur sur le premier maillon de la liste (qui permet d'avoir ainsi accès à toute la liste !).
5. Écrire une fonction `creeMaillon` qui prend en entrée un `TypElement` et un `TypPtrMaillon`, qui alloue dynamiquement un nouveau maillon, qui le remplit avec les deux paramètres, et qui retourne l'adresse du maillon ainsi créé.
6. Écrire une procédure `ajoutDebut` qui prend une liste en paramètre mixte (à la fois d'entrée et de sortie) et un `TypElement` en entrée, et qui ajoute au début de cette liste un nouveau maillon contenant l'élément passé en paramètre. Cette procédure appellera bien évidemment la fonction `creeMaillon`.
7. Écrire une procédure `enleveDebut` qui prend une liste en paramètre mixte, et qui supprime le premier maillon de cette liste (ne pas oublier de restituer la mémoire allouée au système).
8. Écrire une fonction `longueur` qui calcule et retourne le nombre de maillons de la liste passée en paramètre d'entrée.
9. Écrire une procédure `afficheListe` qui prend une liste en paramètre d'entrée et qui affiche tous ses maillons comme dans l'exemple suivant :
   `[0.12|-]->[34.567|-]->///` ou seulement `///` si la liste est vide.
10. Écrire un programme principal qui teste tous les sous-programmes ci-dessus.
11. *Travail personnel*
    Beaucoup d'autres fonctions de manipulation de listes sont intéressantes à programmer (voir les exercices 11.2 à 11.6 du polycopié C++).

.../...

5 Java $\longrightarrow$ C++

Il s'agit dans cette partie de montrer comment un programme java peut appeler des fonctions et des procédures écrites en C et/ou en C++. L'objectif n'est pas de tout retenir par cœur (ce n'est pas au programme du QCM !), mais de comprendre le fonctionnement et de pouvoir se reporter à ce tp ultérieurement (notamment pour le projet). Cela va vous paraître très long d'une part car c'est la première fois que vous mettrez en œuvre ces outils, mais aussi car les instructions sont très détaillées. Il faut donc utiliser ce qu'on appelle JNI (Java Native Interface).

5.1 Un programme en Java, C, et C++

5.1.1 La classe Java

1. Écrire une classe Java `Java2CCPP` contenant un `main` qui affiche simplement le message `"affichage Java"`.
2. Compiler la classe sous DOS en tapant `javac Java2CCPP.java`.
3. Vérifier sous DOS que le fichier `Java2CCPP.class` a bien été créé.
4. Modifier cette classe pour qu'elle puisse appeler une procédure écrite en C et une procédure écrite en C++. Pour cela :
   - il faut dire que l'on voudra utiliser une bibliothèque de fonctions écrites en C que l'on aura choisi d'appeler `Java2C.DLL` (l'extension DLL sous Windows indique une Dynamic Link Library) :
     `static { System.loadLibrary( "Java2C" ); }`
     à placer avant le main()
   - il faut déclarer les méthodes écrites en C que l'on voudra utiliser parmi celles qui sont dans la bibliothèque prcédemment chargée; on se contentera d'une seule procédure :
     `native public static void cProcedure();`
     `native` *indique que ce n'est pas une méthode java*
     à placer également avant le main()
   - il faut dire que l'on voudra utiliser une bibliothèque de fonctions écrites en C++ que l'on aura choisi d'appeler `Java2CPP.DLL` :
     `static { System.loadLibrary( "Java2CPP" ); }`
   - il faut déclarer les méthodes écrites en C++ que l'on voudra utiliser parmi celles qui sont dans la bibliothèque prcédemment chargée; on se contentera d'une seule procédure :
     `native public static void cppProcedure();`
5. Ajouter dans le main() l'appel aux 2 procédures prcédemment déclarées (comme si c'était des procédures écrites dans la classe java).
6. Recompiler la classe sous DOS jusqu'à ce qu'il n'y ait plus d'erreurs.
7. Générer le fichier `.h` qu'il faudra inclure dans les fichiers écrits en C ou en C++ :
   `javah Java2CCPP`
8. Regarder sous DOS le contenu du fichier `Java2CCPP.h` qui vient d'être créé. Il fonctionne à la fois pour le C et pour le C++, et les noms des procédures ont été quelque peu rallongés...

5.1.2 La bibliothèque en C

1. Lancer Visual C++.
2. Créer un nouveau projet (`File/New/Projects`).
3. Cliquer sur `Win32 Dynamic-Link Library`.
4. Saisir `Java2C` dans le champ `Project name:`.
5. Modifier éventuellement le chemin dans le champ `Location:`.
6. Cliquer sur `OK`.
7. Sélectionner `An empty DLL project.`.
8. Cliquer sur `Finish` puis sur `OK`.
9. Cliquer sur l'onglet `FileView`.
10. Cliquer sur le `+` de `Java2C files`.
11. Cliquer avec le bouton droit sur `Header Files`.
12. Sélectionner `Add Files to Folder...`.
13. Double-cliquer sur `Java2CCPP.h` (dans le répertoire parent).
14. Cliquer sur le `+` de `Header Files`.
15. Double-cliquer sur `Java2CCPP.h`.
16. Sélectionner et copier les 2 lignes du prototype de `cProcedure()`.
17. Créer un nouveau fichier (`File/New/Files`).
18. Cliquer sur `C++ Source file`.
19. Saisir `Java2C.c` dans le champ `File name:`.
20. Cliquer sur `OK`.
21. Coller les 2 lignes copiées précédemment.
22. Supprimer le point-virgule final et donner des noms aux 2 paramètres, par exemple `env` et `cla`.
23. Écrire en C le corps de la procédure qui se contentera d'afficher le message `"affichage C"`.
24. Pour pouvoir utiliser `printf`, il faut inclure le fichier `<stdio.h>`.
25. Enfin, pour pouvoir utiliser la communication entre ce programme et la classe Java, il faut inclure `"..\Java2CCPP.h"`.
26. Sauvegarder le fichier (il est terminé !).
27. Configurer Visual C++ pour qu'il puisse communiquer avec Java :
    - Ouvrir `Project/Settings...`.
    - Cliquer sur `C/C++` et choisir la catégorie `Preprocessor`.
    - Saisir dans le champ `Additional Include Directories:` : `c:\jdk\include,c:\jdk\include\win32` où **jdk** est le répertoire dans lequel est installé le JDK sur votre PC.
    - Cliquer sur `OK`.
28. Compiler le programme.
29. Vérifier sous DOS que le fichier `Java2C.obj` a bien été créé dans le répertoire `Java2C\DEBUG`.
30. Faire l'édition de liens (Build).
31. Vérifier sous DOS que le fichier `Java2C.dll` a bien été créé dans le répertoire `Java2C\DEBUG`.
32. Fermer le projet (`File/Close Workspace`).

5.1.3 La bibliothèque en C++

Répéter les 32 opérations de la section précécente en s'adaptant au C++ :

- Aux points 4, 10, 29, et 31, remplacer `Java2C` par `Java2CPP`.
- Au point 16, remplacer `cProcedure()` par `cppProcedure()`.
- Au point 19, remplacer `Java2C.c` par `Java2CPP.cpp`.
- Au point 23, remplacer `C` par `CPP`.
- Au point 24, remplacer `printf` par `cout` et `<stdio.h>` par `<iostream.h>`.

5.1.4 Tout faire marcher ensemble

1. Créér un répertoire `essai` pour regrouper les seuls fichiers nécessaires à l'exécution.
2. Y recopier `Java2CCPP.class`, `Java2C\DEBUG\Java2C.dll`, et `Java2CPP\DEBUG\Java2CPP.dll`.
3. Aller dans le répertoire courant.
4. Lancer le programme java : `java Java2CCPP` :
   vous devriez voir s'afficher les 3 lignes suivantes :
   ```
   affichage Java
   affichage C
   affichage C++
   ```

5.2 Pour aller plus loin

Il est possible d'accéder depuis une procédure C ou C++ aux données des clases et des objets Java, et de renvoyer des données vers le programme Java.

Pour cela, lire les liens présents à la fin de la page web IN3S02 de Denis Bureau.