TD 1 : Rappels de programmation C et orientée objet

Objectifs pédagogiques :

- **mémoire, pointeur, classe**
- **syntaxe générale du c++**
- **constructeur, destructeur**
- **opérateurs**

Introduction

Dans ce premier TD, nous allons réaliser en C++ une classe `String` appuyée par une représentation interne à l'aide d'une chaîne de caractères du C : un pointeur `char *` vers un espace mémoire zéro terminé. Cet exercice est à vocation pédagogique, on préférera en pratique utiliser la `std::string` standard du header `<string>` et les fonctions de manipulation de chaînes du C rangées dans `<cstring>`.

Pour être sûr de ne pas entrer en conflit avec d'autres applications, nous utiliserons le namespace `pr` pour notre implémentation.

1.1 Rappels chaîne du C, const.

Les chaînes de caractères du C, sont représentées par un pointeur de caractère qui cible une zone mémoire contenant la chaîne, et se terminant par un `'\0'`.

On souhaite implanter nos propres petites fonctions utilitaires travaillant avec ces chaînes. Cet exercice est à vocation pédagogique, en pratique on préférera utiliser les fonctions standard du C (`strcat,strcmp,strcpy...`) qui se trouvent dans le header standard `<cstring>` en C++.

On considère :

- une fonction `length` rendant une taille pour la chaîne de caractère.
- une fonction `newcopy` prenant une chaîne de caractère en argument et rendant une nouvelle copie de cette chaîne de caractères.

Question 1. Quel sont donc les signatures de ces fonction ? Où et comment les déclarer, sachant qu'on veut en faire des fonctions utilitaires rangées dans le namespace "pr".

Question 2. Donnez deux implantations de ces fonctions. Pour la fonction `length`, comparer une version utilisant la notation str[i] à une version en pointeurs pur. Pour l'allocation, comparer l'utilisation de l'opérateur `new` du C++ à `malloc`. Pour la copie comparer une invocation à `memcpy` à une boucle. Où placer ce code d'implantation ?

Question 3. Ecrivez un programme dans un fichier `exo1.cpp` qui construit une copie d'une chaîne "Hello World", puis affiche les deux chaines, leurs addresses, leur longueurs, et sort **proprement** (pas de fuites mémoire).

Question 4. Expliquez comment compiler séparément ces fichiers puis les assembler (linker) faire un programme exécutable, et l'exécuter.

1.2 Une classe String

La gestion manuelle des pointeurs et de la mémoire pose des problèmes :

- accès hors d'une zone allouée (par dépassement, ou par accès à un pointeur non initialisé, ou par deref de `nullptr`),
- branchement sur une mémoire non alllouée,
- double désallocation.

La structure de classe permet d'éviter un certain nombre de ces erreurs, en encapsulant ces comportements, de manière à assurer par exemple que les allocations et désallocations sont cohérentes.

Le reste de cet exercice consiste à écrire une classe String qui se comporte “bien”.

Question 5. Dans un fichier “String.h”, définir une classe `String` minimale munie d'un atribut logeant un pointeur vers une chaîne du C et d'un constructeur permettant de positionner cet attribut. Ajoutez une opération membre “length" qui calcule la longueur de la chaîne. Donnez également le code de l'implantation de la classe, qu'on placera dans “String.cpp”.

On doit pouvoir s'en servir de la manière suivante :

```
String str = "Hello";
size_t len = str.length();
```

Question 6. Ajoutez les mécanismes permettant d'imprimer une `String` à la manière C++ standard.

Question 7. Le code actuellement proposé ne copie pas la chaîne passée en argument. Rappelez en quoi consiste le comportement par défaut qui est généré par le compilateur pour les opérations :

```
// dtor
virtual ~String();
// par copie de ori
String(const String &ori);
// mise à jour du contenu
String & operator=(const String & other);
```

Qu'affiche actuellement par exemple le programme suivant ?

```
String getAlphabet() {
      char [27] tab;
      for (int i=0; i < 26 ; ++i) {
            tab[i] = 'a'+i;
      }
      tab[26]='\0';
      return String(tab);
}
int main() {
      String abc = getAlphabet();
      std::cout << abc << std::endl;
}
```

Citer d'autres problèmes que cela peut poser.

Question 8. Modifiez la classe String, pour qu'au contraire, elle soit **responsable** de la mémoire qu'elle pointe. Si l'on se contentait de ne modifier que le destructeur et le constructeur, expliquez les problèmes que cela pose sur cet exemple.

```
int main() {
      String abc = "abc";
      {
            String bcd(abc);
      }

      std::cout << abc << std::endl;

      String def = "def";
      def = abc;

      std::cout << abc << " et " << def << std::endl;
}
```

Question 9. Quels autres opérateurs pourrait-on offrir sur String ?

TME 1 : Programmation, compilation et exécution en C++

Objectifs pédagogiques :

- **mise en place**
- **classe simple**
- **opérateurs**
- **compilation, debugger, valgrind**

1.1 Plan des séances

Cette UE de programmation avancée suppose une familiarité avec le C et un langage O-O comme Java (syntaxe de base, sémantique), les premières séances sont l'occasion de se remettre à niveau. Cet énoncé contient donc plusieurs encadrés, en gris, qui rappellent les notions clés à appliquer.

Comme le niveau est assez hétérogène, prenez le temps de bien absorber les concepts si nécessaire, nous pouvons prendre un peu de retard. Les TME proposent souvent des extensions dont les réalisations sont optionelles (bonus) mais qui permettent d'aller un peu plus loin.

Vous soumettrez l'état de votre TME à la fin de chaque séance, en poussant votre code sur un git. La procédure que l'on va mettre en place ce semestre n'étant pas encore opérationnelle pour cette séance, créez vous un git ou contentez vous de zipper le dossier "src" pour cette séance.

1.2 Prise en main

Eclipse (CDT) : un environnement de développement (IDE) configuré pour C/C++.
Cet environnement graphique gèrera votre projet localement sur votre compte PPTI. Il vous permettra d'éditer les sources, de les compiler et de les exécuter. Son utilisation est fortement recommandée (instructions précises dans les supports), mais d'autres outils peuvent aussi être utilisés (NetBeans, Visual Studio Code, ...).

Question 1. Lancement d'Eclipse

Attention, plusieurs versions d'Eclipse cohabitent à la PPTI. Il faut utiliser une version pour développement CDT, qu'on a déployé dans "/usr/local/eclipseCPP/". La manière la plus sûre est d'ouvrir un terminal et d'y entrer la commande : `/usr/local/eclipseCPP/eclipse`.
Si c'est la première fois que vous utilisez Eclipse, celui-ci vous demande le nom du répertoire *workspace* où il placera par défaut vos projets. On recommande d'utiliser un nouveau dossier `~/workspacePR` comme dossier de workspace qui va loger les projets.

Question 2. Contruire un nouveau projet C++

Démarrer eclipse puis construire un "File->New->C/C++ Project"; on va utiliser le "C++ Managed Build" assez facile d'emploi et bien pris en charge par l'IDE comme système de build. On va construire pour commencer un projet hébergeant simplement un éxécutable, prenez le template fourni "Hello World". Assurez-vous que la chaîne de compilation sélectionnée est bien "Linux GCC". Appelez le projet "TME0". On peut valider les options par défaut sur les autres onglets.

On a à présent un projet avec un main C++ qui affiche un message.

Question 3. Compiler le projet

Sélectionner le menu "Project->Build Project".

La "console CDT" (un des onglets dans la partie basse de l'IDE) montre les instructions de compilation qui ont été faites. On y voit une étape de compilation et une étape de link.

Eclipse a configuré une version compilée en mode Debug par défaut. Il place les makefile qu'il a engendré et les fichiers produits par la compilation dans un dossier Debug. A priori, on n'édite pas directement ces fichiers, mais plutôt les "Properties" du projet (clic droit sur le projet, dernier item).

Il propose aussi une version Release, compilée avec des flags plus optimisés, utiliser la flèche/triangle à côté du marteau (build) dans le ruban d'outils en haut de l'IDE pour basculer sur cette

configuration. Si vous ne voyez pas cet outil, assurez vous d'avoir basculé en Persepctive C/C++ (avec le bouton dans le coin en haut à droite de eclipse). On va rester en configuration Debug pour l'instant.

Un nouvel élément "Binaries" est visible, il contient les binaires qu'on vient de construire. On peut clic-droit sur un binaire et faire "Run As...->Local C/C++ application". Cela lance le programme, dans la console d'eclipse.

Question 4. Ajoutez dans le main un un tableau "tab" de dix entiers que vous remplirez avec les entiers 0 à 9. Affichez le contenu du tableau.

On constate une bonne qualité du soulignement au cours de la frappe, et l'auto-complétion disponible sur ctrl-espace.

Certaines fautes ne sont pas soulignées au cours de la frappe, mais seulement si on lance le build complet. Par exemple cette erreur d'utilisation d'une variable non initialisée ne sera indiquée qu'après une compilation.

```
char * a;
cout << a ;
```

Question 5. Découverte du debugger

Recopier le code suivant dans votre main et lancer le programme.

```
for (size_t i=9; i >= 0 ; i--) {
if (tab[i] - tab[i-1] != 1) {
cout << "probleme !";
}
}
```

Si "probleme" s'affiche, on sait qu'on a un souci. Double-cliquez dans la marge de l'éditeur, sur la ligne qui contient l'affichage de "probleme", l'outil crée un Breakpoint pour le debug.

Cliquez sur le binaire, mais cette fois-ci faites "Debug As..->Local C++ Application" au lieu de "Run As". Accepter de basculer en perspective Debug.

La ligne verte indique la position actuelle dans le code, on la fait évoluer en utilisant les outils dans le ruban du haut.

- Continue : poursuit l'exécution jusqu'au prochain breakpoint (qu'on place en double clic dans la marge)
- Step into, Step Over, Step Return : exécution ligne à ligne

Avec le breakpoint positionné sur notre message, avancer jusqu'à l'atteindre, puis inspecter les valeurs des variables (à droite). Pour l'affichage des pointeurs sous forme de tableau, faire un clic droit sur la variable et demander le mode tableau.

Question 6. Visualisation de l'état sous debugger

Quel est le contenu des cellules du tableau ? Combien vaut "i" ? Modifier la déclaration du type de "i", pour que la boucle aie effectivemet lieu 10 fois.

Question 7. Valgrind et détection des fautes mémoire

A présent, sans avoir complétement debuggé le problème, lancer une analyse avec valgrind. Sélectionner le binaire, et faire "Profiling Tools->Profile with Valgrind". Il doit vous aider à détecter vos erreurs (fautes mémoires et fuites mémoire). Les résultats sont visible dans l'onglet Valgrind.

Lire cette page http://valgrind.org/docs/manual/mc-manual.html#mc-manual.errormsgs décrivant les erreurs détectées par cet outil.

Standard C++ dans Eclipse CDT.
Par défaut la version du C++ utilisée est celle par défaut de notre compilateur, ce qui est insuffisant pour notre usage.
Il faut configurer le dialecte pour la version "-std=c++1y", soit la plus récente disponible. Malheureusement il faut faire ce réglage dans deux endroits :

- Pour l'éditeur/correction à la volée des erreurs, on règle une préférence globale, valable pour tous les projets d'un workspace donné. Naviguer vers
  - "Window -> Preferences -> C/C++ -> Build -> Settings"
  - Ouvrir l'onglet "Discovery".
  - Sélectionner dans la liste le "CDT GCC Built-in compiler Settings"
  - Ajouter un "-std=c++1y" au flags, de façon à avoir
    `${COMMAND} ${FLAGS} -std=c++1y -E -P -v -dD "${INPUTS}"`
- Pour le compilateur à proprement parler, on doit faire un réglage pour chaque projet séparément. En partant d'un clic droit sur le projet, accéder à ses propriétés:
  - "Project properties -> C/C++ Build -> Settings"
  - Sous le "GCC C++ compiler" on trouve une rubrique "Dialect"
  - sélectionner : "-std=c++1y" dans le menu déroulant.

Attention, il faut faire le premier réglage dans tout nouveau workspace, et le deuxième pour chaque nouveau projet.

1.3 Réalisation d'une classe String

Question 8. Définir et tester les fonctions `length` et `newcopy` du TD1.

Question 9. Implanter la classe String conformément aux instructions du TD 1.

Au fur et à mesure de la réalisation de la classe, construire un `main` qui teste les éléments réalisés. S'assurer qu'il ne provoque aucune faute mémoire sous valgrind.

On commencera par réaliser et tester :

- Constructeur par copie de l'argument
- Destructeur qui invoque delete
- Ajout dans un flux / impression de la String
- Constructeurs par copie, operator= redéfinis

Question 10. Ajouter une fonction utilitaire `compare` aux fonctions utilitaires rangées dans "strutil.h". Elle doit se comporter comme la fonction `strcmp` standard, elle prend deux chaînes a et b du C en argument, et elle rend une valeur négative si $a < b$, 0 si les deux chaînes sont logiquement égales, et une valeur positive sinon.

Plus précisément, on itére sur les deux chaînes simultanément (avec deux pointeurs) tant que les valeurs pointées sont égales et que la première est différente de '\0'. On compare ensuite les caractères pointés (on peut faire simplement la différence de leurs valeurs).

Question 11. Pour la comparaison entre deux String, on peut proposer soit des opérateurs membres, soit des fonctions extérieures à la classe (plus symétriques).

Ajouter (en s'appuyant sur `compare`) :

- Un opérateur de comparaison d'égalité `bool operator==(const String &a,const String &b)` symétrique, déclaré friend et en dehors de la classe String.
- Un opérateur fournisssant une relation d'ordre `bool operator<(const String & b) const` membre de la classe String.

Question 12. Pour l'accès aux caractères de la String on peut réaliser un opérateur [].

Définissez `char operator[](size_t s)const`. Peut-on modifier la String par ce biais ? Quelle serait la signature d'une variante permettant de modifier le contenu de la String ? Essayez de la définir,

quel problème de typage (const-ness) se pose ?

Question 13. Pour la construction de String plus longues, on peut proposer un `String operator+(const String & a, const String &b)` symétrique (non membre). L'algorithme consiste à calculer la longueur totale du résultat, allouer (sur le stack) un tableau de cette taille, et construire une String avec ce résultat.

Question 14. Lancer votre programme de test sous valgrind. Si vous n'avez pas d'erreurs, introduisez en délibérément, puis détectez là avec Valgrind, interprétez le message qu'il produit.

- commentez un "delete[]" dans String,
- commentez une copie, i.e. au lieu de faire la copie, contentez de l'affecter à l'attribut de la classe.

1.4 Utilisation de `shared_ptr`

Cette partie du TME est considérée comme du "bonus" pour ceux qui sont le plus à l'aise ou qui sont déjà familiers avec le C++.

Avec les String que l'on a défini, deux String ne peuvent pas partager leur espace de stockage, même si elles ont le même contenu. Comme le contenu de la String est stocké de façon const, partager la représentation des String (en lecture seule donc) ne devrait pas poser de problème. Mais la gestion mémoire est plus complexe, à quel moment libérer une chaîne pointée ? Une solution possible consiste à utiliser un compteur de références, où la chaîne est désallouée quand elle n'est plus référencée, mais n'est pas copiée e.g. quand on affecte une String à une autre.

Lisez la documentation de http://www.cplusplus.com/reference/memory/shared_ptr/ et http://www.cplusplus.com/reference/memory/shared_ptr/shared_ptr/.

Question 15. Dans une copie de votre classe String, que l'on appelera StringPtr, substituer au `const char *` un `std::shared_ptr<const char>` qui est défini dans le header `<memory>`. Mettez à jour la gestion mémoire au sein de la classe (constructeur, destructeur, copie, ...) : on ne garde que le constructeur par copie d'un `const char *`, le comportement par défaut des autres opérations est déjà celui qu'on souhaite. Corriger le code pour qu'il compile et fonctionne bien. Pour obtenir le pointeur nu (`const char *`) à partir d'un `shared_ptr<const char> ptr` on utilise `ptr.get()`.

Question 16. Si on lance le programme sous Valgrind on rencontre un problème d'incohérence new[]/delete, qui gène l'utilisation. Expliquez ce qui se passe. Pour corriger le problème, faites une recherche sur StackOverflow pour les mots clés "shared_ptr array C++", on doit y proposer (vu que l'on n'a pas de compilateur C++17 à la PPTI) d'utiliser le constructeur à deux arguments de `shared_ptr` pour lui passer une fonction à invoquer pour la désallocation. Adopter une des syntaxes proposées, et s'assurer que le problème est résolu.