L3 MIASHS
TP C++ N° 1

Présentation de l'environnement

Pour les séances de TP C++ :

— utilisez l'éditeur de votre choix pour écrire du code C++ ;
— sauvegardez les fichiers avec l'extension .hpp (pour les fichiers d'en-tête) et .cpp (pour les fichiers contenant les définitions) ;
— lancez le compilateur en utilisant la commande :

```
$ g++ [nom-fichier].cpp -o [nom-fichier-sortie]
```

— lancez l'exécution du programme en utilisant :

```
$./[nom-fichier-sortie]
```

— vous pouvez utiliser l'option *−std* pour spécifier le standard ISO C++ à être pris en compte par le compilateur. L'option *−c* peut être utilisée pour indiquer au compilateur de compiler ou assembler les fichiers sources, sans que l'étape de linkage soit faite (voir *man g++* pour plus d'information) :

```
$ g++ -std=c++11 [nom-fichier].cpp -o [nom-fichier-sortie]
$ g++ -std=c++11 -c [nom-fichier].cpp
```

Alternativement, vous pouvez utiliser *clang++*.

Exercice 1 : les fonctions *strlen*, *strcpy* et *strncpy*

Les fonctions *strlen*, *strcpy* et *strncpy* sont définies comme suit dans *<string.h>* :

— `size_t strlen (const char *str)` : renvoie la longueur de la chaîne de caractères *str*. La longueur d'une chaîne est déterminée à l'aide du caractère de terminaison nul ('\0'). La chaîne est aussi longue que le nombre de caractères entre le début de la chaîne et le caractère de terminaison (sans compter le caractère de terminaison lui-même).

```
char str1[]="Exemple chaîne de caractères";
cout << strlen(str1);
```

— `void strcpy (char *destination, const char *source)` : copie la chaîne de caractères pointée par *source* dans le tableau pointé par *destination*, y compris le caractère de terminaison nul. Pour éviter les débordements, la taille du tableau pointé par *destination* doit être suffisamment longue pour contenir la chaîne à être copiée (y compris le caractère de terminaison). Les dépassements de mémoire ne sont pas vérifiés au sein de la fonction (source d'erreurs !).

```
char str1[]="Exemple chaîne de caractères";
char str2[31];
char str3[31];
```

```
strcpy (str2,str1);
strcpy (str3,"Un autre exemple");
```

— `char* strncpy (char *destination, const char *source,size_t num)` : copie les *num* premiers caractères de la chaîne pointée par *source* dans le tableau pointé par *destination*. Si la fin de la chaîne *source* est trouvée avant *num* caractères, le restant du tableau est complété par des zéros jusqu'à ce qu'un total de *num* caractères soit copiés. *destination* est renvoyé. Le caractère de terminaison n'est pas implicitement ajouté à la fin de la chaîne pointée par *destination* si la longueur de *source* est supérieure à *num*. Ainsi, dans ce cas, la destination ne doit pas être considérée comme une 'chaîne style-C'. Comme pour *strcpy*, les dépassements de mémoire ne sont pas vérifiés au sein de la fonction.

```
char str1[]= "Exemple chaîne de caractères";
char str2[15];
char str3[15];
// overflow safe
strncpy(str2,str1,sizeof(str2));
strncpy(str3,str2,8);
str3[8] = '\0';
```

Vous devez :

1. définir un fichier .hpp contenant la signature des trois fonctions (pour éviter le conflit de noms avec les fonctions dans <string>, ajoutez un '_' au nom de chaque fonction)
2. utiliser l'*espace de noms* `fonctions`
3. écrire le code des fonctions dans un fichier .cpp
4. tester votre code à l'aide des exemples fournis ci-dessus (n'oubliez pas de tester le comportement des fonctions en cas de débordement de mémoire)

Exercice 2 : les structures

Considérez les attributs suivants décrivant une personne : `id`, `nom`, `prénom`, et `adresse mail`

1. Définissez la structure `TPersonne`, en utilisant *string* comme type pour chaque champ
2. Écrivez les fonctions suivantes :
   — `TPersonne* new_personne()` : lit les valeurs des champs (entrée console) et renvoie un pointeur de l'objet crée
   — `void affiche(TPersonne *p)` : affiche les valeurs des champs de l'objet pointé par *p*

La déclaration de la structure et des fonctions se fera dans un fichier .hpp et les définitions dans un fichier .cpp.

N'oubliez pas de libérer toutes les ressources dynamiquement allouées, à la fin de l'exécution !

Exercice 3 : les listes doublement chaînées

Dans une liste doublement chaînée, chaque noeud possède, en plus de la donnée elle-même, un pointeur sur l'élément précédent et un pointeur sur l'élément suivant. Cela permet notamment de simplifier l'insertion ou la suppression d'un élément donné, ainsi que le parcours en sens inverse. Pour stocker dynamiquement les données du type `TPersonne`, vous allez implémenter une liste doublement chaînée.

1. Définissez un nouveau type *struct* `TNoeud` contenant trois champs : *pers* (`TPersonne*`), *pred* (`TNoeud*`), et *suiv* (`TNoeud*`)
2. Définissez un nouveau type *struct* `TListe` contenant deux champs : *tete* (`TNoeud*`) et *queue* (`TNoeud*`)
3. Écrivez les fonctions suivantes :
   - `bool ajoute(TListe*,TPersonne*)` : ajoute une personne (en créant le noeud correspondant) dans la liste (la liste ne doit pas stocker deux personnes ayant le même *id*)
   - `TPersonne* supprime(TListe*,const string)` : supprime de la liste l'objet `TPersonne` contenant l'*id* envoyé comme paramètre (renvoie l'objet supprimé ou `nullptr` si l'objet n'existe pas)
   - `void affiche(const TListe*)` : affiche les champs de chaque noeud (TPersonne) de la liste

Vous pouvez définir d'autres fonctions si besoin. La déclaration des structures et des fonctions se fera dans un fichier .hpp et les définitions dans un fichier .cpp. L'espace de nom `liste` devra être utilisé.

Vous devez (au moins !) :

1. créer 4 objets du type `TPersonne` et les ajouter dans la liste
2. imprimer tous les noeuds de la liste
3. supprimer 2 noeuds
4. imprimer à nouveau tous les noeuds de la liste

N'oubliez pas de libérer toutes les ressources dynamiquement allouées, à la fin de l'exécution ! Pour cela, une fonction `desalloc(TListe*)` peut être créée.

Pour finir cet exercice, modifiez la fonction `ajoute_noeud` de façon à ajouter les noeuds par ordre alphabétique de nom.