LO11 : examen médian

Durée : 2h, poly et transparents de cours autorisés. **Rédiger chaque exercice sur une copie différente.**

Exercice 1- La classe string (1 copie) - 12 points

On veut écrire une implémentation simple d'une classe `string` qui permet de faire un certain nombre d'opérations similaires à la classe standard `std::string` du C++. Voici ci-dessous un programme utilisant une telle classe dont l'interface a été fournie dans le fichier d'entête `"mystring.h"` :

```
#include <iostream> #include "mystring.h"
void truc(const string& str) {
      std::cout<<"truc="; // affiche un caractère sur 2
      for(int i=0; i<str.length(); i++) if (i%2) std::cout<<str[i];
}
int main(){
      string s1; // string::string()
      string s2="bonjour"; // string::string(const char*)
      string s3=s2;
      s1="bonsoir";
      s1[1]='a';
      s1[2]='d';
      std::cout<<"s1="<<s1<<"\n";
      std::cout<<"s2="<<s2<<"\n";
      std::cout<<"s3="<<s3<<"\n";
      string s4="hello";
      s4+=";";
      s4+=s2;
      std::cout<<"s4="<<s4<<"\n";
      s3=s1+";"+s4;
      std::cout<<"s3="<<s3<<"\n";
      truc(s3); std::cout<<"\n";
      return 0;
}
```

L'exécution de ce programme provoque l'affichage suivant :

```
s1=badsoir
s2=bonjour
s3=bonjour
s4=hello;bonjour
s3=badsoir;hello;bonjour
truc=asi;el;oju
```

Question 1- Recopier le programme ci-dessus en ajoutant des commentaires qui indiquent les prototypes complets des méthodes de la classe `string` ou des fonctions liées à la classe `string` qui sont appelées dans les instructions du programme (dans les fonctions `truc` et `main`). Les prototypes des fonctions utilisées pour les deux premières instructions de la fonction `main` sont fournis à titre d'exemple.

On rappelle que les chaînes de caractères de type `C` sont des tableaux de `char`. Le caractère `'\0'` suit toujours le dernier caractère de la chaîne pour marquer la fin de celle-ci. La taille du tableau doit donc être supérieure ou égale au nombre de caractères de la chaîne + 1.

Question 2- Ecrire le fichier d'entête `"mystring.h"` et une unité de compilation `"mystring.cpp"` qui permettraient d'exécuter correctement l'ensemble du programme précédent. La classe `string` englobera un pointeur de `char` qui pointera sur un tableau de `char` **alloué dynamiquement**. La taille de ce tableau sera toujours égale au nombre de caractères de la chaîne + 1 caractère pour stocker `'\0'`. On fera particulièrement attention à la gestion de la mémoire. En particulier, les allocations et les désallocations mémoire devront être faites de manière pertinente. A la fin de l'exécution du programme, toute la mémoire allouée dynamiquement aura dû être désallouée. **Votre capacité à utiliser les différentes notions vues en cours et en TD/TP sera particulièrement jugée.**

Dans l'implémentation des méthodes de la classe `string`, vous pouvez utiliser (et c'est vivement conseillé) les fonctions du C suivantes déclarées dans le fichier d'entête `<cstring>` :

- `int std::strlen(const char* str)` : qui renvoie la longueur de la chaîne de caractères pointée par `str` (caractère `'\0'` non compris).
- `const char* std::strcpy(char* dest, const char* source)` : copie les caractères de la chaîne de caractères pointée par `source` dans le tableau de caractères pointé par `dest` (caractère `'\0'` compris).

Exercice 2- Remise en forme (1 autre copie !!!) - 8 points

Les questions sont indépendantes : même si vous n'avez pas répondu à une question, vous pouvez utiliser les éléments de cette question dans les questions suivantes.

On veut développer un ensemble de classes qui permettent de manipuler des formes graphiques simples comme des carrés ou des rectangles. Pour cela, nous allons développer les classes suivantes : « `Forme` », « `Polygone` », « `Rectangle` » et « `Carre` ». Les déclarations de ces classes seront supposées être mises dans le fichier « `forme.h` » et les définitions seront supposées être écrites dans le fichier « `forme.cpp` ».

Question 0- Lire tout l'exercice et repérer l'ensemble des classes, leurs méthodes et leurs attributs et les relations qui les lient entre elles. Dessiner une hiérarchie de classes qui résume tous ces éléments.

Question 1- Qu'est ce qu'une classe abstraite ? A quoi cela sert-il ? De quelle manière pouvons-nous créer une classe abstraite en C++ ?
Ecrire une classe abstraite « `Forme` » qui contient les méthodes **virtuelles pures** suivantes :

```
void Afficher(std::ostream& aff=std::cout) const;
int GetPerimetre() const;
int GetSurface() const;
```

Un polygone est une forme particulière caractérisée par un nombre de côtés. Soit la classe « `Polygone` » dérivée de la classe « `Forme` » dont la déclaration est fournie ci-dessous. Le nombre de côtés du polygone est stocké dans l'attribut `nbCotes`.

```
class Polygone : public Forme {
protected:
      int nbCotes;
public:
      Polygone(int nb_s); // argument = nombre de cotes du polygone
      virtual ~Polygone(); // destructeur
      int GetNbCotes() const; // retourne le nombre de sommets du polygone
      virtual int GetPerimetre() const=0; // calcul du périmètre
      virtual int GetSurface() const=0; // surface du polygone
      // affichage des caractéristiques d'un polygone
      void Afficher(std::ostream& aff=std::cout) const;
};
```

Question 2- La classe « `Polygone` » est-elle concrète ou abstraite ? Justifiez. Ecrire les définitions des méthodes de la classe `Polygone` qui doivent être définies.

Question 3- Un rectangle est un polygone particulier à 4 côtés. Soit la classe « `Rectangle` » dérivée de la classe « `Polygone` ». On veut écrire pour cette classe :

- un constructeur qui prend en arguments la longueur et la largeur d'un rectangle.
- 2 méthodes « `GetLargeur` » et « `GetLongueur` » qui renvoient respectivement la largeur et la longueur d'un rectangle.
- une méthode « `Afficher` » dédiée spécialement à l'affichage des caractéristiques d'un rectangle sur un objet `ostream`.
- une méthode « `==` » qui permettra de comparer 2 rectangles : une instruction telle que « `r1==r2` » devra renvoyer « `true` » si deux rectangles `r1` et `r2` sont superposables et « `false` » sinon.

Ecrire la déclaration de la classe et la définition de ces méthodes. On sera particulièrement attentif à utiliser de façon pertinente les qualificatifs du C++. On fera en sorte que cette classe soit instanciable (c'est-à-dire qu'il soit possible de créer un objet rectangle).

Question 4- La classe « `Rectangle` » est-elle concrète ou abstraite ? Justifiez.

Question 5- Un carré est un rectangle particulier dont la largeur est égale à la longueur. Ecrire (déclaration et définition) une classe « `Carre` » qui dérive de « `Rectangle` ». Implémenter de plus une méthode « `GetCote` » qui renvoie la longueur du côté d'un carré et une fonction membre « `Afficher` » dédiée spécialement à l'affichage des caractéristiques d'un carré.