PATCHWORK

Programmation en C++

modes console, graphique et client-serveur

L'objectif est de créer un programme fonctionnel en C++ utilisant les concepts de la P.O.O.
L'application devra simuler la réalisation d'une grande fresque murale réalisée à partir de plusieurs dessins d'enfants. Les enfants (les clients) envoie à leur maîtresse (le serveur) leurs dessins afin de recueillir son avis et ses suggestions d'amélioration. La maîtresse retourne à l'élève son dessin accompagné de ses annotations. La maîtresse peut à tout moment retrouver le dessin d'un enfant.
Les dessins modifiés sont renvoyés à la maîtresse qui, une fois tous les dessins reçus dans leur version définitive, les assemble afin d'en obtenir un plus grand qui constituera la fresque murale.
Une gestion de la **concurrence** devra être mis en place côté serveur.

Il est demandé une version en **mode console** des interactions élèves-maîtresse.
Une seconde version en **mode graphique** pourra être réalisée (affichage d'une palette de formes et couleur, positionnement des points, mode glisser-déposer, …).

- Les dessins sont constitués de formes géométriques élémentaires et colorées : lignes, polygones, cercles, ellipses.
- Les calculs du périmètre et de l'aire sont requis. L'aire d'un polygone quelconque peut être calculée par triangulation.
- Les formes géométriques peuvent se transformer par homothétie, se déplacer par translation, par rotation, par symétries centrale et axiale.
- Une image peut également contenir des images de plus petite taille (aire).
- On souhaite pouvoir ordonner les formes selon plusieurs critères: leur périmètre, leur aire et leur distance à l'origine. Les relations d'ordre devront utiliser la notation <=.
- Aucun doublon de forme géométrique ne peut exister.
- Le nombre total de chaque forme géométrique de la grande fresque murale doit pouvoir être déterminé afin de procéder à des statistiques (histogramme, fréquence). Il en est de même pour les couleurs.

Penser à mettre en place une **gestion d'exception**s lorsque cela s'avère nécessaire.

Vos tests devront être les plus exhaustifs possibles.

Dans le cadre du dialogue client-serveur, un format des images est à définir.

RENDU PROJET

Le rendu est sous la forme **NOM1_NOM2_NOM3.zip**, archive contenant le makefile, les .h et les .cpp sans oublier le rapport au format pdf. Les projets sont à déposer sur l'espace de rendu de l'ENT, ceux rendus uniquement par mail seront pénalisés. Merci de votre sens des responsabilités.

Afin de bien mettre en évidence les relations entre les classes de votre projet, vous pouvez utiliser un logiciel qui vous générera un diagramme de classes à partir de vos sources.
L'idéal est d'utiliser un logiciel libre permettant la rétro-conception.

StarUML : https://www.projet-plume.org/mots-cles-proposes-par-lauteur/retro-ingenierie

BOUML : https://www.projet-plume.org/fiche/bouml

Une documentation des sources doit être générée automatiquement (utiliser doxygen pour documenter les sources).
Utliser des outils pour générer des graphes d'appels :

http://odellconnie.blogspot.fr/2012/07/free-c-software-call-graph-generators.html

Un rapport doit fournir votre analyse et votre conception.

Les éléments suivants doivent notamment y figurer :

- une introduction exposant clairement les objectifs, limites, choix du projet
- un mode d'emploi de l'application
- des schémas décrivant l'architecture fonctionnelle
- des schémas décrivant les structures de données utilisées
- une explication en français et/ou pseudo-code et/ou langage d'implémentation choisi des principales fonctions (code à fournir et à commenter)
- une conclusion résumant le travail effectué et ouvrant des perspectives
- une bibliographie utilisée
- une table des matières

Les classes conçues doivent être détaillées aussi bien leur interface que leur implémentation.
Votre application doit être testée grâce à un code client assez représentatif. Un jeu d'essais (code client), le plus exhaustif possible, est donc à fournir. Des **tests unitaires** pourront être écrits.
Une bonne idée est de réaliser le rapport en même temps que le développement.

Options facultatives du projet :

1) analyseurs lexical et syntaxique : outils **flex** et **bison**
2) interface graphique : librairies **gtk, qt**

ANNEXES

I set

std::set<T,...>

La classe set permet de décrire un ensemble ordonné et sans doublons d'éléments de type T.
Le type T doit disposer d'un constructeur vide T().

Exemple :

```
#include <set>
#include <iostream>
using namespace std;

int main() {
    set<int> s; // équivaut à std::set<int, less<int> >
    s.insert(2); // s contient 2
    s.insert(5); // s contient 2 5
    s.insert(2); // le doublon n'est pas inséré
    s.insert(1); // s contient 1 2 5
    set<int>::const_iterator sit (s.begin());
    set<int>::const_iterator send(s.end());
    for( ; sit != send; ++sit)
        cout << *sit << ' ';
   cout << endl;
   return 0;
}
```

Attention : le fait de supprimer ou ajouter un élément dans un std::set rend invalide ses iterators. Il ne faut pas modifier un std::set dans une boucle for basée sur ses iterators

II Map

std::map<K,T,...>

Une map permet d'associer une clé (identifiant) à une donnée (table associative).

La map prend au moins deux paramètres templates :

1) le type de la clé K
2) le type de la donnée T

À l'image du std::set, le type K doit être ordonné. Le type T impose juste d'avoir un constructeur vide.

Attention : le fait de supprimer ou ajouter un élément dans un std::map rend invalide ses iterators. Il ne faut pas modifier un std::map dans une boucle for basée sur ses iterators.

Attention : le fait d'accéder à une clé via l'opérateur [ ] insère cette clé (avec la donnée T()) dans la map. Ainsi l'opérateur [ ] n'est pas adapté pour vérifier si une clé est présente dans la map, il faut utiliser la méthode find. De plus, il ne garantit pas la constance de la map (à cause des insertions potentielles) et ne peut donc pas être utilisé sur des const std::map.

Exemple :

```
#include <map>
#include <string>
#include <iostream>
using namespace std;
int main() {
   map<string, unsigned> map_mois_idx;
   map_mois_idx["janvier"] = 1;
   map_mois_idx["février"] = 2;
   //...
   map<string, unsigned>::const_iterator mit (map_mois_idx.begin());
   map<string,unsigned>::const_iterator mend(map_mois_idx.end());
   for( ;  mit != mend;  ++mit)
      cout << mit->first << '\t' << mit->second << endl;
   return 0;
}
```