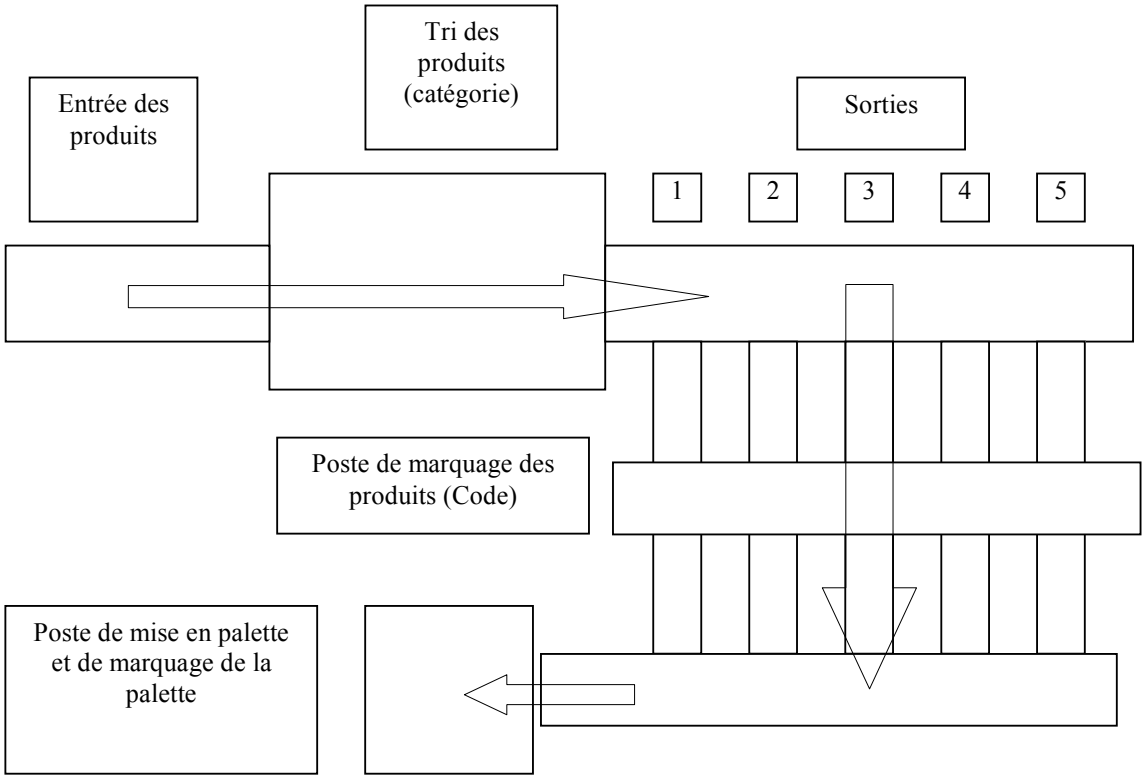
Projet C++ : modélisation d'une machine de tri

Présentation

Ce projet sous Linux propose la modélisation simple d'une machine permettant de trier des produits et de les regrouper par palette après leur avoir attribué un code unique.

Le projet se déroule sur 6 X 1h30 et fera l'objet d'une note de contrôle continu. Il est à réaliser en binôme et sera rendu sur *Moodle* le vendredi 22 février au plus tard.

Principe :

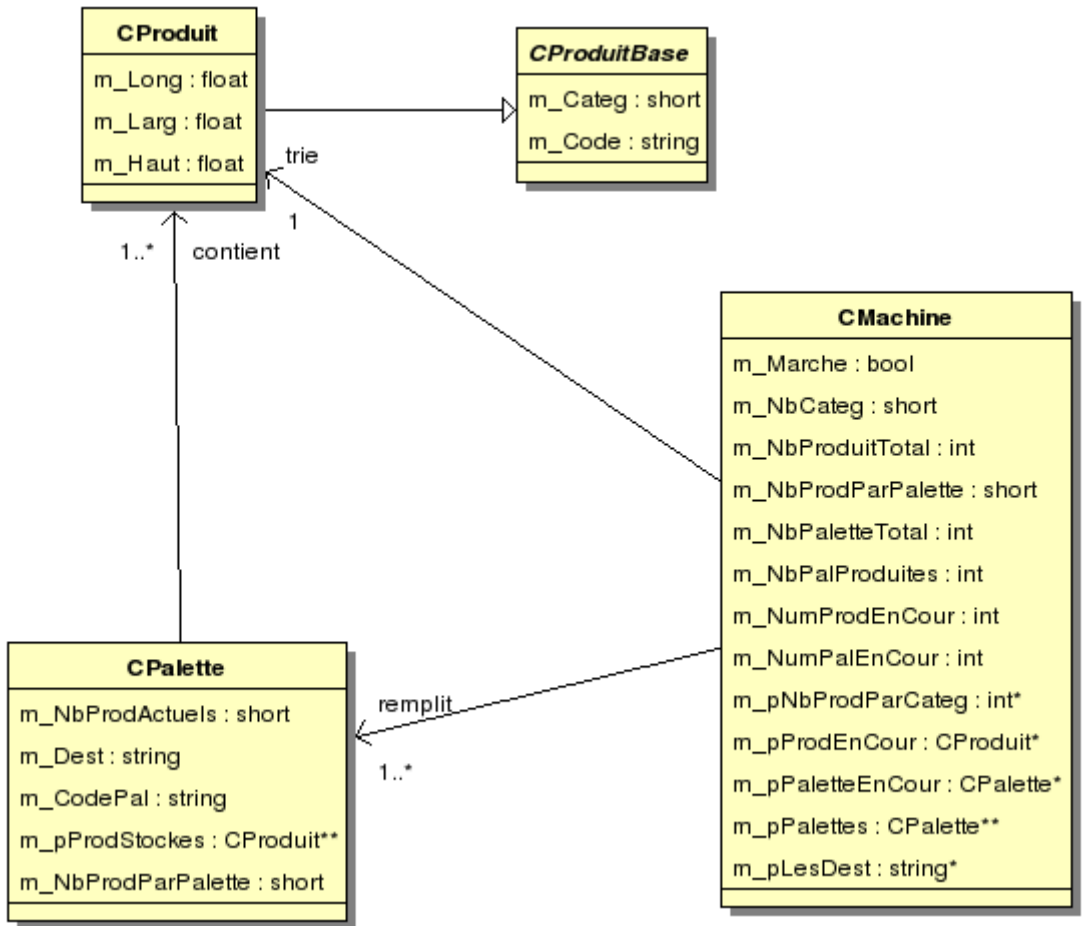


Travail demandé

Réaliser un programme en C++ permettant de modéliser le fonctionnement de la machine définie ci-dessus. Partant d'un diagramme de classes de conception de l'application (voir ci-dessous) et de quelques fichiers C++ utiles au codage (voir cours C++ sur Moodle), il est demandé de coder les classes une par une pour arriver à l'application finale. Dans la mesure du possible, la gestion des erreurs doit se faire à l'aide d'exceptions (de manière semblable à vos développements Java).

Pour chaque classe (concrète) codée (3 au total), il est également demandé d'y associer un fichier contenant un lanceur dont le rôle est de tester *complètement* la classe développée.

Le lanceur final de l'application (*tri.cpp/tri.h*) est fourni (voir fichiers C++ fournis). Ce lanceur nécessite l'utilisation de la classe *CKeyboard.cpp* qui est également fournie.



Ce projet ne nécessite aucun environnement de développement particulier, seuls un éditeur classique et un fichier *makefile*, obligatoire pour la compilation, sont nécessaires. Ce fichier *makefile* permettra de compiler séparément les différents lanceurs : d'une part les lanceurs qui testent chacune des 3 classes développées et d'autre part le lanceur final (*tri.cpp*) qui exécute l'application finale.

Détail de codage

La classe *CProduit*

Un produit est caractérisé par son volume (*m_Long, m_Larg, m_Haut*). La machine lui attribue une catégorie (un entier compris entre 1 et *NbCateg*). En fonction de sa catégorie et du numéro de palette sur laquelle il est stocké, la machine lui attribue un code (chaîne de caractères, *m_Code*) qui a la forme suivante :

< Produit « NumProd » - Cat « m_Categ » - Pal « NumPal » >

Le numéro du produit (*numProd* de type *int*), et le numéro de palette (*numPal* de type *int*) sont uniquement connus de la machine (voir classe *CMachine*) et ne sont donc pas attributs de la classe *CProduit*.

La classe *CProduit* hérite de la classe *CProduitBase*. La classe *CProduitBase* (abstraite) possède juste des accesseurs et modificateurs (pas de constructeurs) et une méthode abstraite *virtual void MakeCode (int NumProd, int NumPal) = 0* à redéfinir dans la classe dérivée *CProduit* (il s'agit donc du modificateur de l'attribut *m_Code*). Il faudra ajouter un destructeur *~CProduit()* contenant comme seule instruction l'affichage de l'adresse mémoire du produit qui est détruit.

Les méthodes de la classe *CProduit* sont les suivantes :

- un constructeur sans paramètres qui initialise les attributs *m_Long, m_Larg, m_Haut* : utiliser la fonction *rand()* (qui renvoie un entier aléatoire *i* telle que *0 <= i < RAND_MAX*, *RAND_MAX* étant une constante déjà déclarée et connue du compilateur, elle ne doit donc pas être déclarée) pour initialiser la longueur, la largeur et la hauteur aléatoirement. Pour améliorer la lisibilité, ramener chaque longueur à une valeur comprise entre 1 et *DIM* (*DIM* étant une constante à déclarer, 20 par exemple),
- accesseurs et modificateurs si nécessaire,
- une méthode *void MakeCode (int NumProd, int NumPal)* qui construit le code du produit (attribut *m_Code*) sous forme d'une chaîne de caractères *< Produit « NumProd » - Cat « m_Categ » - Pal « NumPal » >* (sachant que *NumProd, NumPal* et la catégorie *m_Categ* seront fixés et attribués par la machine),
- une fonction amie *friend ostream& operator << (ostream& os, CProduit& UnProd)* qui surcharge l'opérateur << de manière à afficher le contenu du produit sous la forme :

  Longueur :
  Largeur :
  Hauteur :
  Categorie :
  Code :

  Cette fonction amie permettra d'afficher les caractéristiques du produit grâce à la simple instruction *cout << unProd << endl;* (*unProd* étant un objet de type *CProduit*).

La classe *CPalette*

Une palette est caractérisée par le nombre actuel de produits stockés et par sa ville de destination. Une palette ne peut pas contenir plus qu'un nombre de produits déterminé (attribut *m_NbProdParPalette*) qui définit la taille du tableau de pointeurs sur chacun des produits stockés sur la palette.

La palette possède également un code (*m_CodePal*, chaîne de caractères) de la forme suivante :

< Palette N° « NumPal » - Destination : « m_Dest » >

Le numéro de la palette (*NumPal* de type *int*) est uniquement connu de la machine et n'est donc pas attribut de la classe *CPalette*.

Les méthodes de la classe *CPalette* sont les suivantes :

- un constructeur avec 2 paramètres : le nombre de produits par palette (type *short*) et la destination (type *const string*). A l'initialisation, il n'y a aucun produit stocké,
- un destructeur qui détruit la palette ET chacun des produits stocké (afficher dans le destructeur le code de la palette détruite),
- accesseurs et modificateurs si nécessaire,
- une méthode *void MakeCode (int NumPal)* qui construit le code de la palette (attribut *m_CodePal*),
- une méthode *void AddProduit (CProduit* pProduit)* qui ajoute au tableau des produits un pointeur sur un nouveau produit stocké,
- une méthode *CProduit* GetProduit (int IndexTab)* qui renvoie le pointeur sur le produit qui se trouve dans le tableau à l'index passé en paramètre.

La classe *CMachine*

La machine est la classe qui possède les différents traitements à effectuer sur le produit : insertion du produit dans la machine, attribution d'une catégorie, marquage du produit et stockage du produit sur la palette. Chaque fois qu'une palette est remplie (cf. *m_NbProdParPalette*) une nouvelle palette est créée avec un nouveau numéro.

C'est la machine qui décide des numéros à attribuer aux produits qui rentrent dans la machine et aux palettes sur lesquels sont stockés les produits. Chaque palette est remplie au fur et à mesure par les produits. La numérotation des produits et des palettes est complètement séquentielle et commence à zéro dans les deux cas (zéro pour le premier produit et la première palette puis incrémenter chaque fois de 1).

Pour faciliter le codage de la classe :
- le fichier *CMachine.h* vous est fourni : lire le fichier attentivement !
- le code de la méthode *void Affiche()* est donné (voir le fichier *CMachine.h*)

Séquencement et notation

Attention: lors de l'évaluation, la compilation de votre projet se fera exclusivement avec le compilateur g++ de Linux installé sur le réseau enseignement (le code *C/C++* n'est pas portable !).

Pour chaque classe codée, un lanceur de test doit y être associé. La compilation de chaque lanceur doit se faire obligatoirement à l'aide du fichier *makefile* dont un exemple est fourni. Dans la notation, la qualité du test de chaque classe indépendante sera évalué.

L'ordre dans lequel doit être développé l'application est le suivant :
- classes *CProduitBase* et *CProduit* + *testCProduit*
- classe *CPalette* + *testCPalette*
- classe *CMachine* + *testCMachine*
- test final avec le lanceur *tri.cpp* fourni (*tri.cpp* utilise un objet *CMachine* et doit s'exécuter parfaitement sans bugs)

La notation tiendra compte :

- de l'état d'avancement du projet
- de la conformité du code par rapport au cahier des charges
- de la qualité des commentaires, de la clarté de codage et du respect des règles de style C++
- de la qualité des tests

Rendu

Une archive *NomBinôme1_NomBinôme2.tgz* ou *NomBinôme1_NomBinôme2.zip* est à déposer sur *Moodle* pour le **vendredi 22 février à 23h55 au + tard**.

A partir du samedi 23 février, chaque jour de retard entraîne une pénalité supplémentaire de 2 points.

L'archive doit contenir :

- un répertoire */src* qui contient tous les fichiers **.cpp* et **.h* de l'application, y compris les fichiers fournis (*tri.cpp*, *tri.h*, *CKeyboard*) et les fichiers de test (*testCProduit* etc.),
- un fichier *makefile* qui contient l'ensemble de toutes les instructions de compilation du projet et qui place les fichiers compilés dans un répertoire */bin*,
- un fichier *readme* :
  - qui décrit l'état d'avancement du projet,
  - qui reprend clairement la liste des classes qui peuvent être compilées et testées grâce au *makefile* fourni.

Informations utiles

- Reste et quotient d'une division entière :

```
#include <stdlib.h>
div_t tmp;
tmp = div ( int numerator, int denominator );
tmp.rem;          // type « int », reste de la division entière
tmp.quot;         // type « int », quotient de la division entière
```

- De l'aide sur C++ est disponible sur les sites suivants :

http://www.cplusplus.com/ref

http://www.cppreference.com/

http://www.commentcamarche.net/cpp