Final

lundi 24 juin 2013, de 8h00 à 10h00

Note : Documents cours, TD et TP papier autorisés. La copie du voisin même papier, ne l'est pas.

1 Le trait fatal (5pts)

common_type est une classe trait de boost, utilisée pour déduire un type commun à plusieurs types, utilisé comme type de retour d'une fonction travaillant sur plusieurs types comme les expressions arithmétiques. Elle est définie dans le fichier "*boost/type_traits/common_type.hpp*".

Sa définition peut être :

```
1   template <class ...T>
    struct common_type;
2   template <class T, class U, class ...V>
    struct common_type<T,U,...V> {
        typedef typename common_type<typename common_type<T, U>::type, V...>::type type;
    };
3   template <class T>
    struct common_type<T> {
        typedef T type;
    };
4   template <class T, class U>
    struct common_type<T, U> {
        typedef decltype(declval<bool>() ? declval<T>() : declval<U>()) type;
    };
```

Notes : Le rôle de la fonction template *declval()* est une transformation du type template en valeur sans utilisation ou évaluation de la fonction. L'opérateur *decltype*() a pour but de donner le type d'une expression.
La norme c++11 a apporté une solution à cette question avec le mot clé ***auto***.

1.1 Expliquez les différentes déclarations de cette classe

1.2 Ecrivez une fonction générique permettant de déterminer la valeur médiane entre 3 valeurs compatibles passées en paramètre. Ceci doit se faire sans perte d'information (par exemple : retour d'un entier au lieu d'un réel). **Vous n'utiliserez pas *auto*.**

2 Le grain d'esprit attendu (5pts)

Voici un programme à compléter. Le but est d'analyser un nom de fichier. Vous devez décomposer le nom fourni en parties : le(s) répertoire(s), le nom à proprement parlé et d'extension. Le résultat sera retourné dans un vecteur.

Attention, le répertoire peut être vide, ainsi que l'extension !

L'extension ne pourra pas avoir plus de 4 caractères. Utilisez au choix le séparateur '/' ou '\'. Comment feriez vous pour ne retourner que 3 parties (chemin, nom et extension), ceci quelque soit la taille du chemin.

```
#include <boost/config/warning_disable.hpp>
#include <iostream>
#include <boost/spirit/include/qi.hpp>
```

```
#include <boost/spirit/include/phoenix_operator.hpp>
namespace qi = boost::spirit::qi;
namespace ascii = boost::spirit::ascii;

struct extension : qi::grammar<std::string:: iterator, … , ascii::space_type>
{
      extension() : extension::base_type(start) {
        …
      }
      qi::rule …
};

int main(){
      std::string test("/usr/machin/test.ext");
      extension grammaire;
      … resultat;

      std::string::iterator it = test.begin();
      std::string::iterator end = test.end();
      qi::phrase_parse(it, end, grammaire, ascii::space, resultat);
      // vérification analyse
      // affichage résultat
        return 0;
}
```

3 Traitement en série ! (10 pts)

3.1 Résultats

Nous disposons d'un vecteur contenant des notes, dont les indices correspondent à un matricule d'élève précis.
Nous voulons obtenir les résultats en termes de réussite ou échec dans la matière enseignée.

- Développez un foncteur "*Reussite*" qui permet d'obtenir un booléen vrai ou faux en fonction de la valeur de la note supérieure ou égale à 10.
- Utilisez ce foncteur pour obtenir un vecteur de booléens contenant le résultat de chaque élève.

Attention, vous devez utiliser impérativement un algorithme de la STL qui évite d'écrire un parcours explicite du vecteur !

3.2 Adaptation

- Pour adapter le résultat en fonction de la difficulté de l'examen on souhaite changer aisément la limite de 10 définie. Modifiez votre foncteur pour introduire un paramètre template réel qui servira de limite.
- La souplesse introduite n'est pas suffisante. Modifiez le foncteur afin de permettre une définition de limite à l'exécution. La limite sera définie à la création de l'objet foncteur et par défaut le 10 sera utilisé.

3.3 Moyenne

Nous disposons de deux vecteurs contenant des notes.

- Nous souhaitons créer un vecteur des moyennes de chaque élève. Pour ceci, créez une fonction "*lesMoyennes(std::vector<double> v1, std::vector<double> v2, std::vector<double> vm)*" qui parcourt les vecteurs et établit la moyenne pour chaque élève puis la stocke dans un troisième vecteur.

- Modifiez votre fonction en *lesMoyennesCont* pour accepter tout type de conteneur.
- Afin de généraliser encore (travail avec tout type de regroupement de données accessibles par iterateur), utilisez la logique de la STL comme les algorithmes, pour écrire *lesMoyennesStl*.

3.4 Adaptation du calcul

- Ecrivez une classe *Moyenne* qui fournit une ***méthode de classe*** *operation* permettant de calculer la moyenne entre deux valeurs reçues en paramètre.
- Sur la base de votre fonction *lesMoyennesStl* écrivez *lesMoyennesPol* pour intégrer une classe politique dans les paramètres templates. Celle-ci devra fournir la méthode *operation* qui sera utilisée pour le calcul de moyenne.
- Ecrivez le code nécessaire pour initialiser le vecteur des moyennes avec une moyenne pondérée : médian 45% (première note) et final 55% (seconde note). On suppose que vous disposez des vecteurs de notes v1 et v2 et que le vecteur de moyennes vm est déclaré.

3.5 Correction de bug

Dans le code qui suit :

- Corrigez les erreurs éventuelles. (*On suppose par la suite que la compilation réussit*)
- Expliquez l'objectif d'un appel à *transform* et s'il y en a une, la différence entre les deux appels à cet algorithme.
- L'appel à *transform* provoque une erreur à l'exécution. Expliquez pourquoi et proposez une solution qui ne modifie pas l'appel, puis une solution intervenant sur le type de paramètre à l'appel. Dans les deux cas, vous motiverez votre proposition.

```
#include <vector>
#include <iterator>
#include <algorithm>
#include <functional>
vector< double > v1, v2, v3;
double *tab={12, 4, 16, 7, 9, 14, 17, 9, 8, 13, 10};
v2::iterator it2=v2.begin();
for(int i=0; i<10; ++i){     v1.push_back(i);}
transform(tab, tab+9, it2, bind2nd(plus<double>,3));
transform(tab, tab+9, it2, bind(plus<double>,3,placeholders::_1));
copy(v1.begin(), v1.end(), std::inserter(v3, v3.begin()));
```

3.6 Regroupement

- Nous souhaitons créer un vecteur regroupant les notes de chaque élève. Pour ceci, utilisez un appel à *transform* sur les 2 plages contenant les notes et pour créer les paires avec la fonction *make_pair*

Pour information, les prototypes pour l'algorithme *transform* :

```
template <class InputIterator, class OutputIterator, class UnaryOperation>
  OutputIterator transform (InputIterator first1, InputIterator last1,
```

```
                    OutputIterator result, UnaryOperation op);

template <class InputIterator1, class InputIterator2,
        class OutputIterator, class BinaryOperation>
  OutputIterator transform (InputIterator1 first1, InputIterator1 last1,
                    InputIterator2 first2, OutputIterator result,
                    BinaryOperation binary_op);
```