**Contrôle Continu AP4A – Mardi 5 Avril 2022 - durée 1 heure  
(aucun document autorisé)**

**I.** (4 pts) Considérons le type Vecteur en C++ tel que réalisé en TP et des variables de type Vecteur. Donner la notation fonctionnelle équivalente des instructions suivantes :

Vecteur a, b, c, d, u, v;

... // les vecteurs sont initialises avec des contenus

a = a \* b + c;

c = (a + b) + c;

v[i] = v[j] = 2.0;

a = b = c = 2;

cout << “Vecteur u : “ << u << “Vecteur v : “ << v << endl;

((a = b) = c) = 2;

if (u != v) u = v;

++u = v++;

**II**. On considère ici la classe Liste telle que codée en récursif :

template <class T>

class Liste {

T val; // tete de liste = element

Liste\* reste; // suivi d'une liste

public:

… // interface étudiée en cours/TD/TP

}

1) (2 pts) Donner un schéma d’occupation mémoire (pile et tas) pour le programme suivant aux points d’exécution 1) et 2).

{

Liste<float> l;

l.ajouter(3.0, 0);// ajouter

l.ajouter(4.0, 0);

l.ajouter(6.0, l.longueur());

l.ajouter(7.0, 1);

Liste<float>\* pl = new Liste<float>;

pl->ajouter(3.0, 0);// ajouter

pl->ajouter(4.0, 0);

pl->ajouter(6.0, pl->longueur());

pl->ajouter(7.0, 1);

//1)

delete pl;

//2)

}

2) (2 pt) Considérons la version suivante de l’opérateur [] :

T **operator**[](int pos) {

if (reste == NULL || pos == 0)

return val;

else

return (\*reste)[pos-1];

}

Soit l une liste, peut-on écrire l[2] = 8; par exemple, justifier à l’aide d’un schéma mémoire. Comment corriger ?

3) (10 pts) On souhaite définir une sous-classe Iterateur (incluse à l’intérieur) de la classe Liste respectant l’interface standard des itérateurs de la STL (Standard Template Library) pour parcourir le contenu de la liste codée en récursif. Une ébauche de la classe est la suivante :

template <class T>

class Liste {

…

public:

class Iterateur {

friend class Liste;//a cause du constructeur private

... pl;

Iterateur(... l) : pl(...) {}

public:

bool **operator**!=(Iterateur it) {

return pl->reste != ...;

}

Iterateur& **operator**++() {

pl = ...;

return ...;

}

Iterateur **operator**++(int i) {

...;

pl = ...;

return ...;

}

... **operator**\*() {// operateur de dereferencement \*iter

return ...;

}

};

Iterateur **begin**() {

Iterateur it(...);

return it;

}

Iterateur **end**() {

Iterateur it(...);

return it;

}

} ;

pour l’utiliser de manière standard par exemple de la façon suivante :

{

Liste<int> l1 ;

...//ajout des elements dans la liste

cout << "Maniere STL Premier passage l1 " << endl;

for(Liste<int>::Iterateur it = l1.begin(); it != l1.end(); ++it){

cout << \*it << endl;

}

cout << "Maniere STL Deuxieme passage l1 " << endl;

for(Liste<int>::Iterateur it = l1.begin(); it != l1.end(); ++it){

\*it = 1;

}

cout << "Maniere STL Troisieme passage l1 " << endl;

for(Liste<int>::Iterateur it = l1.begin(); it != l1.end(); it++){

cout << \*it << endl;

}

}

Donner le codage complet (interface et implémentation) de la sous-classe Iterateur et des méthodes begin(), end() de la classe Liste en complétant là où il y a des « … ». Attention de bien énumérer exactement tous les éléments de la liste.

4) (2 pts) Nous n’avons pas écrit la forme de Coplien pour la classe Iterateur, faudrait-il l’ajouter ? en quoi consiste-t-elle ? pourquoi ne pas l’écrire ?