IFT339 Structures de données (Été 2001) 29 juin 2001
Examen intra−trimestriel 09h00−11h00
Daniel Lemire, Ph.D. D7−2023 et D7−3021
SOLUTIONAIRE

Nom :_____

Prénom :_____

Matricule :_____

L'examen est sur 100 points et comprend 4 questions. Vous devez répondre sur le questionnaire. Un résumé du C++ est inclus sur cette page. Aucune autre documentation n'est permise.

Les opérateurs C++ (en ordre décroissant de priorité)

Catégorie d'opérateurs	*Opérateurs*
fonction, tableau, membre de structure, pointeur sur un membre de structure	() [] . −>
opérateurs unaires	− ++ −− ! ~ * & sizeof (type)
multiplication, division, modulo	* / %
addition, soustraction	+ −
opérateurs binaires de décalage	<< >>
opérateurs relationnels	< <= > >=
opérateurs de comparaison	== !=
et binaire	&
ou exclusif binaire	^
ou binaire	|
et logique	&&
ou logique	||
opérateur conditionnel	?:
opérateurs d'affectation	= += −= *= /= %= &= ^= |= <<= >>=
opérateur virgule	,

Déclaration d'une classe

```
class MaClasseDerivee: public MaClasseDeBase {
public:
        MaClasseDerivee();
        MaClasseDerivee(const MaClasseDerivee&);
        ~MaClasseDerivee();
        const  MaClasseDerivee& operator=(const MaClasseDerivee&);
};
```

Flots

```
ofstream, ifstream, cin, cerr, cout
cin >> P;//lecture
cout << P; // écriture
```

Question 1. Technique (25 points)

A. Expliquez aussi clairement que possible mais brièvement ce qu'est X dans les expressions suivantes. Vous ne devriez pas utiliser plus d'une phrase par expression. (20 points)

double X; (2 points)
variable de type double

double * X; (2 points)
pointeur sur des « double »

double * X[10]; (2 points)
un tableau de 10 pointeurs sur des « double »

double (* X)[10]; (2 points)
un pointeur sur des tableaux de 10 « double » (double[10])

double X(10); (2 points)
variable de type double contenant la valeur 10 ou 10.0

double X[10]; (2 points)
un tableau de 10 « double »

double& X; (2 points)
une référence à une valeur de type « double »

double * & X; (2 points)
une référence à un pointeur sur des « double »

```
double p1 = 1.0;
double * p2 = & p1;
double * & X = p2; (2 points)
```
une référence à un pointeur sur la variable p1 (valeur 1.0)

```
MaClasse Y;
MaClasse & X = Y; (2 points)
```
une référence à l'objet MaClasse (constructeur par défaut)

B. Qu'est−ce qui s'affichera à l'écran après l'exécution de ce code? (5 points)

```
int b; for (b = 2; b <= 2; b++) cout << b++; cout << b;
```

On devrait voir « 24 » ou l'équivalent (affichage de 2 puis de 4).

Question 2. Utilisation d'un tableau (25 points)

Il n'est pas rare en C++ qu'on veuille sécuriser l'accès à un pointeur. En voici un exemple:

```
template <class TYPE>
class MonTableau {
public:
        MonTableau(const int N) {
                tableau = new TYPE[N];
        }

        ~MonTableau() {
        }

        const TYPE operator[](const int indice) {
                return tableau[indice];
        }

protected:
        TYPE * tableau;
        MonTableau(const MonTableau&);
};

void main() {
        MonTableau mt(10);
        mt[2] = 3;
}
```

A. Trouvez deux erreurs dans ce code qui empêcheront la compilation et corrigez−les (dans le code).
(10 points) *1. remplacer* *MonTableau mt(10); par MonTableau<int> mt(10);*
2. remplacer *const TYPE operator[] par TYPE& operator[]*

B. Un programmeur vient vous consulter parce qu'après avoir corrigé les deux erreurs précédentes, le code suivant pose toujours un problème. (Expliquez en une phrase ce qu'est le problème et comment le corriger.) (5 points)

```
const int N =  1024*1024*10;
for(int k = 0; k < 1000; k++) {
        MonTableau mt(N);
}
```

Il y a au moins trois bonnes réponses:

Il y a une fuite de mémoire. On devrait ajouter « delete[] tableau » au destructeur.
Il y a une possibilité de débordement de l'entier N
Il y a une erreur de syntaxe: MonTableau<int>

C. Est−ce que cette classe est sécuritaire? Proposez au moins une modification qui pourrait rendre cette classe plus sécuritaire. On peut supposer que l'opérateur « = » et le copieur ne sont pas utilisés. On dit qu'une classe est sécuritaire si elle ne peut pas causer de plantages. (10 points)

La possibilité la plus évidente est de remplacer la méthode

```
TYPE& operator[](const int indice) {
    return tableau[indice];
}
```

par

```
TYPE& operator[](const int indice) {
    assert(indice>=0);
    assert(indice < taille);
    return tableau[indice];
}
```

et d'ajouter la ligne « taille = N » dans le constructeur et la déclaration « int taille; » dans la section « private » de la classe. Il y a d'autres façons de faire et d'autres corrections possibles.

Question 3. Utilisation d'une Pile (25 points)

D'une part, on désire dériver la classe Pile ci−après de telle manière que l'on puisse connaître en tout temps le nombre d'éléments présents dans la pile en un temps $O(1)$ et sans faire de supposition particulière concernant l'*implémentation* de la Pile. D'autre part, on désire aussi faire en sorte que la taille de la Pile ne dépasse jamais 1 000 000 éléments et qu'en cas de débordement, l'ajout d'un élément remplace simplement l'élément au−dessus de la pile.

```
template <class TYPE>
class Pile {
public:
        Pile();
        virtual ~Pile();
        virtual void empiler(const TYPE&);
        virtual TYPE depiler();
        virtual void vider();
        bool estVide() const;

protected:      /*...*/
};
```

Écrivez votre classe « MaPile » ici :

```
template <class TYPE>
class MaPile: public Pile<TYPE> {
public:
        MaPile() {//mettre votre code ici
                Taille = 0;
        }
        void empiler(const TYPE& t) { //mettre votre code ici
                if( Taille == 1000000 ) {
                        Pile::depiler();
                        Pile::empiler(t);
                } else {
                        Pile::empiler(t);
                        Taille++;
                }
        }
        TYPE depiler() {//mettre votre code ici
                if(Taille > 0) {
                        Taille−−;
                        Pile::depiler();
                }
        }
        void vider() {//mettre votre code ici
                Pile::vider();
                Taille = 0;
        }
        int nombreDElements() {//mettre votre code ici
                return Taille;
        }
private://mettre votre code ici
        int Taille;
};
```

Question 4. Utilisation d'une file (25 points)

La norme vidéo H.261 utilise une file afin de respecter les limites de la bande passante disponible (un multiple de 64 kbps). Vous devez coder une classe dont la méthode « tick » sera appelée 1000 fois par secondes (aux millisecondes), la file contient des objets « Image » munis d'une méthode « int taille() » qui donne la taille en octets de chaque image. La méthode tick doit défiler la file de façon sélective : elle doit donner soit un pointeur « NULL » soit un pointeur sur une image. Il faut qu'en moyenne, vous ne dépassiez pas une utilisation de la bande passante de 240 octets par seconde. Par exemple, si la méthode tick donne une image ayant un poids de 2400 octets au temps $t = 0$, il faudra que laisser s'écouler 10 secondes avant d'envoyer une nouvelle image (le temps que l'image soit transmise). On retourne « NULL » pour indiquer qu'on ne peut pas encore donner l'image suivante. Attention! Vous devez tenir compte du cas où la file serait vide!!! (Voir Fig. 1.)

La classe file est une classe générique munie des opérations suivantes :

```
void enfiler(const TYPE&);
TYPE defiler();
bool estVide() const;
void vider();
```

```
class Video {
        Video(file<Image> * f) {// mettre votre code ici
                mMaFile = f;
                mMilliOctets = 0;

        }
        Image * tick() {// mettre votre code ici
                if (mMilliOctets > 240)  mMilliOctets −= 240; else  mMilliOctets = 0;
                if (mMaFile−>estVide() || (mMilliOctets > 0)) return NULL;
                mImageCourante = mMaFile−>defiler();
                mMilliOctets = mImageCourante.taille() * 1000;
                return &mImageCourante;
        }
        private:// mettre votre code ici
                file<image> * mMaFile;
                image mImageCourante;
                int mMilliOctets;
};
```

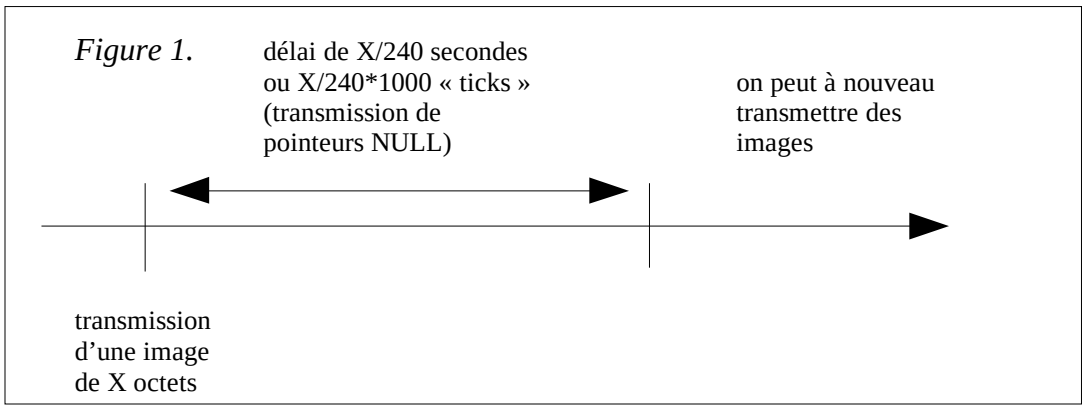

Figure 1.