Trimestre Eté, 2007

Mohamed Lokbani

# IFT1166 – Examen Final –

Inscrivez tout de suite votre nom et le code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : mardi 3 juillet 2007

Durée : 3 heures (de 17h30 à 20h30)

Local : N-515 ; Pavillon Roger Gaudry

## Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises et clairement présentées.**

1._____/20 (1.1, 1.2, 1.3, 1.4)

2._____/15 (2.1)

3._____/20 (3.1, 3.2)

4._____/15 (4.1)

5._____/30 (5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9)

Total:_____/100

**Directives officielles**

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remettra sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes. Un seul étudiant à la fois peut quitter la salle.

* Toute infraction relative à une fraude, un plagiat ou un copiage est signalée par le surveillant au directeur de département ou au professeur qui suspend l'évaluation.

F.A.S

**Exercice 1 (20 points)**

**1.1** Expliquez brièvement la notion de « fuite de mémoire ».

**1.2** Écrivez les instructions correspondantes pour libérer l'espace mémoire alloué pour chacune des instructions ci-dessous :

| | **Allocation** | **Libération** |
|---|---|---|
| A | `string *p = new string;` | |
| B | `string *p = new string[3];` | |
| C | `Temps *p = new Temps(12,0,0);` | |
| D | `int *p = new int(33);` | |
| E | `int *p = new int[33];` | |

**1.3** Identifiez et corrigez les erreurs dans le programme suivant, s'il y'en a.

```
#include <iostream>
using namespace std;
int main() {
    int *p1 = new int (5);
    int *p2 = p1;
    cout << "p1 pointe la valeur " << *p2 << endl;
    delete [] p1;
    delete p2;
    return 0;
}
```

**1.4** Le programme suivant compile correctement mais se plante lors de l'exécution avec un message « segmentation fault ». Il contient en effet 3 erreurs fondamentales reliées à la manière avec laquelle le programme gère l'espace mémoire alloué. Trouvez ces erreurs et corrigez-les.

```
 1      #include<iostream>
 2      int main() {
 3          using namespace std;
 4          double* d = new double;
 5          for(unsigned int i = 0; i < 3; i++) {
 6              d[i] = 1.5 + i;
 7          }
 8          for(unsigned int i = 2; i >= 0; i--) {
 9              cout << d[i] << endl;
10          }
11      }
```

**Exercice 2 (15 points)** Dites si les lignes mentionnées ci-dessous sont syntaxiquement correctes ou non. Pour chacune des lignes incorrectes, indiquez la raison.

```
#include <iostream>
using namespace std;
class C {
    private: int x;
    protected: int y;
    public: int z;
    C(){x=0;y=0;z=0;}
};
class C1: public C {
    private: int x1;
    public: int y1;
    C1(){x1=0;y1=0;}
};
class C2: private C {
    private: int x2;
    public: int y2;
    C2(){x2=0;y2=0;}
};
int main() {
    C1 o1;
    C2 o2;
```

| | |
|---|---|
| `cout << o1.x << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o2.y << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o1.z << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o1.x1 << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o1.y1 << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o2.z << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o2.x2 << endl;` | Correcte : Incorrecte :<br>Pourquoi : |
| `cout << o2.y2 << endl;` | Correcte : Incorrecte :<br>Pourquoi : |

```
    return 0;
}
```

**Exercice 3 (20 points)**

**3.1** Pour chacune des fonctions « f, m et g », justifiez si elle va s'exécuter correctement (la fonction termine) ou non. Si elle s'exécute correctement, dites ce qu'elle calcule.

```
// f est appelée qu'avec des valeurs de n >= 0 par exemple f(3)
int f(int n){
    if(n == 0)
        return 1;
    else
        return f(n+1);
}
```

Exécution correcte et affichage en sortie : Exécution Incorrecte :

Pourquoi :

```
// m est appelée qu'avec des valeurs de n < 0 par exemple m(-3)
int m(int n){
    if(n == 0)
        return 0;
    else {
        int result = m(n-1);
        result += n;
        return result;
    }
}
```

Exécution correcte et affichage en sortie : Exécution Incorrecte :

Pourquoi :

```
// g est appelée avec ("Il etait une fois un petit chaperon rouge",'t')
int g(char* p, char c){
    return (*p)?(*p==c) + g(p+1,c):0;
}
```

Exécution correcte et affichage en sortie : Exécution Incorrecte :

Pourquoi :

**3.2** Expliquez les erreurs dans la fonction récursive ci-dessous et proposez une version corrigée (vous devez donc coder une nouvelle version qu'en modifiant le corps de la fonction. L'en-tête doit rester la même) :

```
double plus_grande_valeur(int nb, double *valeurs) {
    int a, b;
    a = plus_grande_valeur(nb/2, valeurs);
    b = plus_grande_valeur(nb/2, valeurs + nb/2);
    if(a >= b) return a;
    else return b;
}
```

**Exercice 4 (15 points)** Soit une classe « Pile » déjà définie et qui contient les méthodes suivantes :

```
// Elle permet d'empiler les éléments du type "char"
void push(char);

// Elle permet de dépiler les éléments. Elle retourne l'élément
char pop(void);

// Elle vérifie si la pile est vide. Retourne « true » si c'est
// le cas, sinon « false ».
bool estVide(void);
```

En utilisant la classe Pile définie précédemment, écrivez la fonction « verifBalance » qui vérifie si un tableau de caractères contenant des parenthèses est bien « parenthésé ». Par exemple :

```
char par0[] = "";                          // ok
char par1[] = "(()())";                    // ok
char par2[] = "((((((()())()))()))())";    // ok
char par3[] = "(((()())";                  // incorrect. Manque )
char par4[] = "(()(()))))";                // incorrect. Manque (
char par5[] = ")(";                        // incorrect. Manque ( et )
```

La signature de la méthode est comme suit :

```
bool verifBalance(char * str)
```

La fonction doit retourner « true » si le tableau est correctement balancé sinon « false » dans le cas contraire.

**Exercice 5 (30 points)** Soit le fragment de code suivant :

```
//=================
// Classe Vehicule
//=================

class Vehicule {
public:
    Vehicule(int nbRoues = 0);
    virtual ~Vehicule();
    virtual void Affiche() const;
    int GetRoues() const;
    void SetRoues(int Roues );
private:
    int m_nbRoues;
};
Vehicule::Vehicule(int Roues){
    m_nbRoues = Roues;
}
int Vehicule::GetRoues() const{
    return m_nbRoues;
}
void Vehicule::SetRoues(int Roues){
    m_nbRoues = Roues;
}
void Vehicule::Affiche() const{
    cout << "Vehicule";
}
Vehicule::~Vehicule() {}


//=================
// Moto class
//=================

class Moto:public Vehicule{
public:
    Moto(int Roues = 2);
    ~Moto();
};

// Constructeur de Moto à compléter

Moto::~Moto (){}


//==================
// Classe Voiture
//==================
class Voiture:public Vehicule {
public:
    Voiture (int Roues = 4);
    void Affiche() const;
};

// Constructeur de Voiture à compléter

void Voiture::Affiche() const{
    cout << "Voiture";
}
```

```
//============================
// Classe Parc de stationnement
//============================

class Parc {
public:
    Parc(int maxVehicules = 20);
    ~Parc();
    int NbStationnes() const;
    void Garer(Vehicule& v );
    int TotalRoues() const;
    void Affiche() const;
    Parc(const Parc&);
    Parc& operator=(const Parc&);
private:
    int m_nbVehicules;
    Vehicule **m_Vehicules;
};

// Constructeur de recopie et opérateur= sont
// déjà codés.

// Constructeur de Parc à compléter

// Destructeur de Parc à compléter

int Parc::NbStationnes() const {
    return m_nbVehicules;
}

// Méthode Garer à compléter
// Elle stocke le véhicule (l'argument) dans
// le tableau m_Vehicules

// Méthode Affiche à compléter
// Elle affiche en sortie le type du véhicule
// et son nombre de roues
// Comme exemple de format d'affichage:
// Voiture:4

// Méthode ToTalRoues à compléter
// Elle retourne le nombre total de roues dans
// le parc de stationnement


//==================
// Fonction « main »
//==================

int main(){
    Parc Parking;
    Voiture chevy;
    Voiture camry;
    Moto harley(3);
    Moto honda;
    Parking.Garer(chevy);
    Parking.Garer(honda);
    Parking.Garer(harley);
    Parking.Garer(camry);
    Parking.Affiche( );
    cout << endl;
    cout << "Le parc contient : "\
         << Parking.TotalRoues() \
         << " Roues" << endl;
    return 0;
}
```

**5.1** Citez au moins deux instructions du programme qui forcent l'utilisation de la ligature dynamique par ce dernier.

**5.2** Le programme contient-il une classe abstraite ? Si oui, nommez-la.

**5.3** Nommez les classes qui dérivent de la classe « Vehicule ».

**5.4** La classe « Parc » peut-elle conceptuellement parlant devenir une classe de base pour les autres classes du programme ?

**5.5** Le compilateur va-t-il prévenir la classe « Voiture » pour redéfinir la méthode « GetRoues » ?

**5.6** Le compilateur va laisser la classe « Voiture » redéfinir la méthode « GetRoues », mais ce n'est pas une très bonne idée de le faire ? Est-ce vrai ?

**5.7** La méthode « SetRoues » ne peut être utilisée de manière polymorphique car ce n'est pas une méthode virtuelle (ou virtuelle pure). Est-ce vrai ?

**5.8** Écrivez le code des méthodes suivantes :

(a) Constructeur de la classe « Moto »

(b) Constructeur de la classe « Voiture »

(c) Constructeur de la classe « Parc »

(d) Destructeur de la classe « Parc »

(e) Méthode « Garer » de la classe « Parc »

(f) Méthode « Affiche » de la classe « Parc »

(g) Méthode « TotalRoue » de la classe « Parc »

**5.9** Tout en justifiant votre réponse que va afficher en sortie le précédent programme ?