Examen Final C++

Structures de contrôle

1. La suite de **Tribonacci** est une suite (inspirée par la suite de Fibonacci) dont chaque terme est la somme des trois termes qui le précèdent. Les premiers termes de la suite Tribonacci sont : 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, …
Écrire une fonction récursive qui, étant donné un entier positif **n,** retourne le **n-**ième terme de la suite Tribonacci. **(1 point)**
Écrire une deuxième version non-récursive de la même fonction. **(1 point)**

Tableaux

2. Écrire une fonction qui prend comme paramètre un tableau d'entiers **A** ainsi que sa taille et retourne **true** s'il existe un entier **x** qui paraît au moins deux fois dans le tableau **A**. **(1 point)**
Exemple : Pour le tableau A=[1, 2, 3, 4, 2, 5] la fonction doit retourner **true**, puisque 2 paraît deux fois, alors que la fonction doit retourner **false** pour le tableau A=[2, 3, 1, 5].

3. Écrire une fonction qui prend comme paramètres deux tableaux d'entiers **A** et **B**, ainsi que leurs tailles respectives. Votre fonction doit décider si les tableaux **A,B** sont disjoints, c'est-à-dire, elle doit retourner **true** si les deux tableaux n'ont pas d'éléments en commun. **(1 point)**
Exemple : Pour les tableaux A= [1, 6, 5] et B = [2, 7, 4, 5, 6] votre fonction doit retourner **false**, puisque 5 est un élément en commun entre **A,B**. Pour les tableaux A= [5, 6] et B = [1, 2, 3] elle doit retourner **true**, puisque **A,B** sont disjoints.

Pointeurs et gestion de la mémoire

4. Qu'affiche le programme suivant ? **(1 point)**

```
int b[] = {1,0,2,0,1};
int *p[5];
int i;
for(i=0; i<5; i++)
        p[i] = b+i+b[i];
for(i=0; i<5; i++)
        cout << *(p[i]-i) << endl;
```

5. Considérons le programme suivant:

```
int **p;
int *q;
int i,j;
for(i=0; i<5; i++){
        q = new int[i+1];
        for(j=0;j<i+1;j++)
                q[j] = i*j;
        p[i] = q;
}
for(i=0; i<5; i++){
        for(j=0;j<i+1;j++)
                cout << p[i][j] << ", ";
        cout << endl;
}
```

Trouvez et corrigez toutes les erreurs du programme ci-dessus. Une fois corrigé, que va-t-il afficher ? **(2 points)**

Listes chaînées

6. Écrire une fonction qui, étant donné un pointeur sur la tête d'une liste chaînée, alloue une nouvelle liste avec les mêmes éléments que la liste donnée, mais dans l'ordre **inverse**. La fonction doit retourner un pointeur sur la tête de la nouvelle liste. **(2 points)**
Exemple : Étant donnée la liste 1,2,3,4,5 il faut retourner la liste 5,4,3,2,1.

Note: Utilisez la définition suivante pour la structure liste chaînée.

```
struct Node { int data; struct Node *next; };
```

7. Écrire une fonction qui, étant donné un pointeur sur la tête d'une liste chaînée, retourne **true** si la liste est un **palindrome**. Une liste est un palindrome si l'ordre de ses éléments reste le même qu'on les lise de gauche à droite ou de droite à gauche. **(3 points)**

Exemple : La liste 1,2,3,2,1 est un palindrome, la liste 1,2,2,1 est un palindrome, la liste 1,2,3,4,2,1 n'est pas un palindrome.

Classes

7. Écrire une classe **Time**. Chaque objet de type Time stockera un certain nombre d'heures, de minutes et de secondes (tous des entiers).

On vous demande de proposer : **(3 points)**
1. La définition de la classe Time
2. Un constructeur sans arguments et un constructeur qui prend trois entiers.

3. Un opérateur d'affichage qui permet d'afficher un objet Time.

4. Un opérateur d'addition qui permet d’additionner deux objets Time.

5. Un opérateur de multiplication qui permet de multiplier un objet Time avec un entier positif.

6. Un opérateur de comparaison entre deux objets de la classe Time.

Pour les questions 3-6, donnez des versions surchargées des opérateurs **< <, +, *, <**.

NB : Rappelons que les secondes et les minutes ne doivent jamais avoir une valeur >60.
Exemple : 75 secondes doit être convertie en 1 minute et 15 secondes.
NB : Cette information doit être prise en compte dans le constructeur et dans les opérateurs.

8. Considérons la classe suivante

```
class A{
int x;
public:
  A(int y) { x=y; cout << "Const " << x << endl;}
  A(A& a) { cout << "Copy const " << a.x << endl; x =a.x;}
  ~A() { cout << "Dest " << x << endl; }
  void operator=(A);
  friend A operator+(A a1, A a2);
};

A operator+(A a1, A a2)
{ cout << "+" << endl; return a1;}

void A::operator=(A a2)
{ cout << "=" << endl;  }
```

Qu’affiche le programme suivant? **(2 points)**

```
void f(A a){ cout << "f" << endl; }
int main()
{
   A a1(1),a2(2);
   a1+a2;
   a1 = a2;
   f(a1);
}
```

9. Considérons la classe suivante

```
class B{
int *x;
public:
  B() { x=new int; *x=0; cout << "Const "<< *x << endl;}
  B(int a) { x=new int;*x=a;cout<<"Const "<< *x << endl;}
  ~B() { cout << "Dest " << *x << endl; delete x; }
};
```

Qu'affiche le programme suivant? **(1 point)**

```
void f(B b1){ cout << "f" << endl; }
int main() { B b1(1), b2(2); f(b1); f(b1); f(b2); }
```

Héritage

10. Qu'affiche le programme suivant? **(1 point)**

```
class A {
protected:
int x;
public:
A(int x=0) { this->x = x; }
~A() { cout <<"~A:" << x << endl; }
void f() { cout << "A:" << x << endl; }
};
class B: public A{
int y;
public:
B(int x,int y):A(x) { this->y = y; }
B(int x=0) {this->y = x; }
~B() { cout << "~B:" << x << ", " << y << endl;}
void f() { cout << "B:" << x << ", " << y << endl; }
};
int main()
{
    A **p = new A*[5]; int i;
    for(i=0;i<5;i++){
            p[i] = i%2? new A(i): new B(i,i);
    }
    for(i=0;i<5;i++){ p[i]->f();  }
    for(i=0;i<5;i++) delete p[i];
    delete [] p;
}
```

11. Qu'affiche le programme ci-dessus si on déclare la méthode A::f() et la méthode A::~A() comme étant **virtual** ? **(1 point)**