Correction des exercices de préparation pour l'examen final

Exercice 1) Dites ce qui sera affiché par le programme suivant :

```
#include <iostream.h>
class point {
private:
   float x,y;
public :
   point(float abs=0.0,float ord=0.0) {
      x=abs; y=ord;
   }
   void affiche() {
      cout <<  "point ("<<x<< ","<< y<< ")\n ";
   }
   void deplace(float dx, float dy) {
      x=x+dx; y=y+dy;
   }
};
class pointcol :public point {
private:
  int cl;
public :
  pointcol(float abs=0.0,float ord=0.0,int coul=2) : point(abs,ord) {
    cl=coul;
  }
  void colore(int coul) {
    cl=coul;
  }
  void affiche() {
    point::affiche();
    cout<<"\t couleur :"<<cl<<"\n";
  }
};
main(){
  pointcol c(2,3,1);
  c.affiche();
  ((point)c).affiche();
  c.deplace(-1,-1);
  c.affiche();
}
```

point <2,3>
couleur: 1
point <2,3>
point <1,2>
couleur:1

Exercice 2)

Définir une classe **tableau** ayant les caractéristiques suivantes:

- Repose sur un tableau de dimension n, n étant passé au constructeur (allocation dynamique)
- Implémenter les 4 opérateurs +,-,*,:

On accède aux données à l'aide de l'opérateur []. En cas de débordement, cet opérateur imprime un message et ne fera rien de dangereux.

Voici une possible solution. Tout est conçu dans la classe tableau. La méthode copie évitera la répétition de code.

```
#include  <iostream.h>
#include <conio.h>
class tableau {

private:
  const int sz;
  int *A;
  int reserve;

  void copie (int src, int *dest, int s) {
    for (int i=0; i<s; i++)
         *(dest++)=src;
  }

void copie (int *src, int *dest, int s) {
    for (int i=0; i<s; i++)
         *(dest++)= *(src++);
}

public:
 tableau (int n): sz(n),reserve(0) {
    cerr << "constructeur sz = " << n<<endl;
    A = new int[sz];
    copie(0,A,sz);
  }

  tableau (const tableau & t): sz(t.sz),reserve(0) {
    cerr << "constructeur de copie" << endl;
    A = new int[sz];
    copie(t.A,A,sz);
  }
```

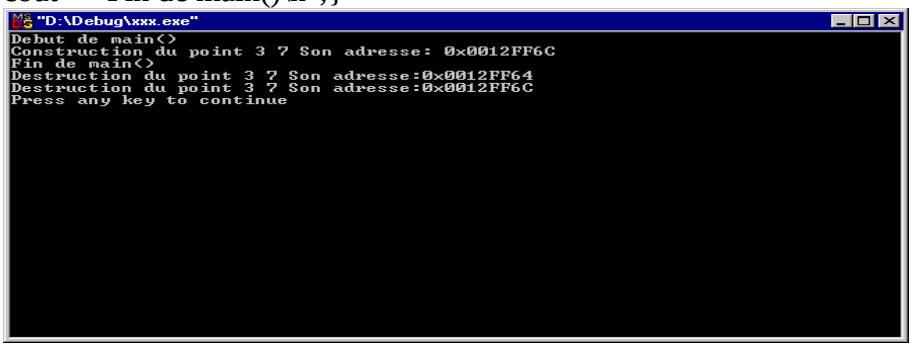
```
  tableau & operator=(const tableau &t) {
    cerr << "operateur =" << endl;
    if (sz != t.sz) {
      cerr << "Ne peut pas egaliser deux tableaux de tailles differentes" << endl;
    };
    copie(t.A,A,sz);
    return *this;
  }

  ~tableau() { delete [] A;}

  int taille() const { return sz;};

  int  operator[](int i) {
    if (i<0 || i>=sz) {
      cerr << "ATTENTION Debordement de tableau";
      return reserve;
    } else
          return *(A+i);
      }

  tableau & operator+=(const tableau & t) {
    if (sz != t.sz)
      cerr << "Ne peut pas ajouter deux tableaux de tailles differentes" << endl;
    else
      for (int i=0; i < sz; i++)
              A[i] += t.A[i];
      return *this;
  }

  void affiche () const  {
    for (int i=0; i < sz; i++)
      cout << *(A+i) << " ";
   cout << endl;
  }

tableau operator+(const tableau& t2) {
  tableau s(t2.taille());
  if (taille() != t2.taille())
    cerr << "Ne peut pas ajouter deux tableaux de tailles differentes" << endl;
  else {
    s = *this;
    s += t2;
  };
  return s;
}

};
```

Les opérateurs – et * seront définis de la même façon que +

Exercice 3)

Dites ce qui sera affiché par le programme suivant :

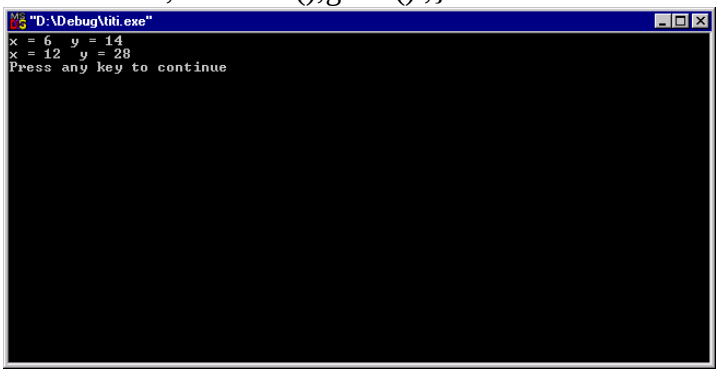
```
#include <iostream.h>
class point
{int x,y;
public:  point(int,int);
 ~point();};

point::point(int abs,int ord)
{x = abs; y = ord;cout<<"Construction du point "<<x<<" "<<y;
cout<<" Son adresse: "<<this<<"\n";
}

point::~point()
{cout<<"Destruction du point "<<x<<" "<<y<<" Son adresse:"<<this<<"\n";}

void main()
{cout<<"Debut de main()\n";
point a(3,7);
point b=a;
cout<<"Fin de main()\n";}
```



Exercice 4)

```
#include <iostream.h>
#include <conio.h>
                                // Classe vecteur
// Surdefinition de l'operateur + par une fonction AMIE
class vecteur
{float x,y;
public:

        vecteur(float,float);
        void affiche();
        friend vecteur operator+(vecteur, vecteur);
};
```

```
vecteur::vecteur(float abs =0,float ord = 0)
{x=abs;y=ord;}

void vecteur::affiche()
{cout<<"x = "<<x<<"  y = "<<y<<"\n";}

vecteur operator+(vecteur v, vecteur w)
{ vecteur res(0,0);
res.x = v.x + w.x;
res.y = v.y + w.y;
return res;}

void main()
{vecteur a(2,6),b(4,8),c(0,0),d(0,0);
c = a + b; c.affiche();
d = a + b + c; d.affiche();getch() ;}
```



Exercice 5: Écrire et tester un programme qui instancie un patron de fonction implémentant une recherche dichotomique sur un tableau d'objets triés. La fonction recoit le tableau v, élément recherché elem et les bornes de recherche inf et sup (inférieure, supérieure). Si trouvé on retourne son indice sinon on retourne –1.

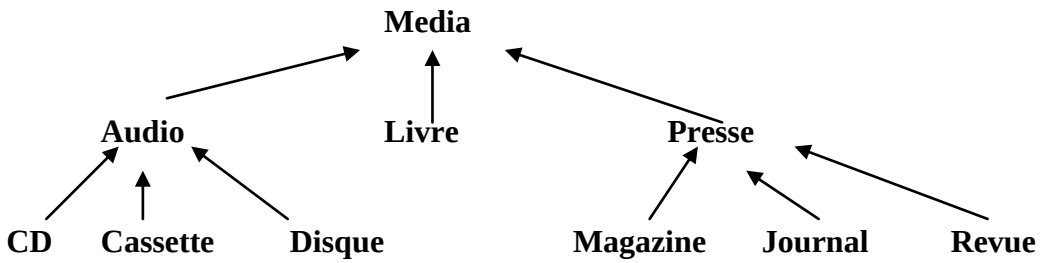
```
template<class T>
int dicho(T v[], T elem, int inf, int sup) {
        int mil;
        while (inf<=sup) {
            mil=(inf+sup)/2;
            if (elem < v[mil])
                sup=mil-1;
            else
                if (elem > v[mil])
                    inf=mil+1;
                else
                    return mil;
        }
        return –1;
}
```

Exercice 6 : Implémenter et tester un patron qui engendre des classes file attente (FileAttente). Une file attente ressemble à une pile mais on fait les insertions d'un côté tandis que les extractions sont faites de l'Autre. C'est ce qui simule classiquement une file attente à un péage d'autoroute ou à une caisse de supermarché. Vous devez avoir les méthodes :

inserer	insérer un élément
enlever	enlever un élément
est_vide	tester si file d'attente est vide
est_pleine	tester si file d'attente est pleine

```
template<class T> class FileAttente {
        public:
                FileAttente(int t=100) : taille(t+1), devant(0), arriere(0)
                { adrF= new T[taille];}
                ~FileAttente() {delete [] adrF;}
                void inserer(const T &x) { adrF[arriere++ % taille] = x;}
                T enlever() { return adrF[devant++  % taille];}
                int est_vide() const {return devant == arriere;}
                int est_pleine() const {return (arriere + 1) % taille == devant;}
        private:
                int taille, devant, arriere;
                T *adrF;
};
```

Exercice 7 : En utilisant la notion de **virtual** donner une possible déclaration de l'hiérarchie de classes suivante :




```
Classe Media {
        public:
                virtual void imprime();
                virtual char *id();
        protected:
                String titre;
};

class Livre: public Media {
        public:
```

```
        Livre(….) {}
    private:
        String auteur, editeur, isbn;
};

…..
```

le reste est similaire.

Exercice 8 : Soit le programme suivant, dites ce qui sera affiché :

```
class X {
    public:
        void f() {cout << "Exécution de X::f()"<<endl;}
};


class Y : public X {
    public:
        void f() {cout << "Exécution de Y::f()"<<endl;}
};

main() {
X x;
Y y;
X *p=&x;
p->f();
p = &y;
p->f();
}
```

```
Exécution de X::f()
Exécution de X::f()
```

Si on remplace la class X par :

```
class X {
    public:
        virtual void f() {cout << "Exécution de X::f()"<<endl;}
};
```

Que serait affiché ?

```
Exécution de X::f()
Exécution de Y::f()
```

Exercice 9 : Étudier ce programme et dites ce qu'il affichera.

```
#include <iostream.h>
#include <list.h>

using namespace std;
typedef list<int> LISTINT;

void main(void)
{
	LISTINT liste;
	LISTINT::iterator i;

	//ajout de données dans la liste

	liste.push_front (2);
	liste.push_front (1);
	liste.push_back (3);

	for(i=liste.begin(); i!=liste.end(); ++i)
			cout<<*i<<" ";
	cout<<endl;

	for(i=liste.end(); i!=liste.begin(); --i)
			cout<<*i<<" ";
	cout<<endl;

}
```

```
1 2 3
0 3 2
```

(En Borlando C++ en Visual C++ peut-être au lieu de 0 il affichera undefined. Ceci pour le cas de liste.end();)

FIN DU SOLUTIONNAIRE DES EXERCICES