Niveau : 2LMD. Année 2018-2019
Matière : **Programmation Orientée Objet**
Equipe de la matière : Dr. Kahloul, Dr. Zernadji, Mr. Guerrouf, Mr. Hamida, Mr. Alloui

TD3 : Pointeurs et Programmation Orientée Objet

Questions de révision générale :

1) C'est quoi un pointeur ? Quelle sont les opérations qu'on peut appliquer sur les pointeurs dans le C, C++ ? Que signifie la constante NULL ?
2) Donner une définition d'un objet.
3) Donner la définition d'une classe.
4) Dans un développement orienté objet, on commence par identifier les objets ou les classes.
5) Citer deux avantages de la programmation OO vis-à-vis celle procédurale ?

Exercice 1 : (Exemples sur les pointeurs)

Remarque :

1) En C++, on peut déclarer des pointeurs comme dans le C en utilisant l'opérateur * :
   double *pointeurA; //Un pointeur qui peut contenir l'adresse d'un nombre à virgule
   unsigned int *pointeurB; //Un pointeur qui peut contenir l'adresse d'un nombre entier positif
   string *pointeurC; //Un pointeur qui peut contenir l'adresse d'une chaîne de caractères
   int const *pointeurE; //Un pointeur qui peut contenir l'adresse d'un nombre entier constant
   Mm
2) Il est possible d'affecter l'adresse d'une variable dans un pointeur par l'opérateur **&**:
   int ageUtilisateur;
   int *ptr;
   ptr = **&**ageUtilisateur;
3) Il est possible d'afficher l'adresse contenu dans un pointeur, ainsi que la valeur pointée par ce pointeur :
   cout<< "La valeur est : "<<ptr<<**endl** ; // afficher l'adresse
   cout << "La valeur est : " << *ptr << **endl**; // afficher le contenu
4) On peut louer manuellement une adresse avec l'opérateur **new** :
   int *pointeur;// déclaration d'un pointeur de type int*
   pointeur = **new** int;//demande de l'adresse manuellement
5) On peut libérer manuellement une adresse avec l'opérateur **delete** :
   delete pointeur; //On libère la case mémoire
6) Si pointeur ptr pointe sur une structure str{type1 X ; type2 Y} alors pour accéder à X, on peut écrire :
   (*ptr).X ou bien **ptr->X**

Remarque très importante:

1) Il est toujours préférable d'initialiser le pointeur avec la valeur **0** ou **NULL** lors de sa déclaration;
   int* ptr=0;
   int *ptr(0);// ça donne le même résultat

2) Après la suppression d'un pointeur (**delete**), il est aussi nécessaire de mettre 0 dans ce pointeur;

Question :

Soit le programme C++ suivant, on vous demande de déduire les résultats de l'affichage tout en justifiant chaque résultat.

Programme 1 :

```
int x=3;
cout<<x<<endl;
int* p0;
cout<<p0<<endl;
p0=&x;
```

```
cout<<p0<<endl;
cout<<*p0<<endl;

int* p1=NULL;
cout<<p1<<endl;
p1=new int;
cout<<p1<<endl;
*p1=2;
cout<<*p1<<endl;
delete p1;
cout<<p1<<endl;
cout<<*p1<<endl;
```

Programme 2 :

```
int* p0;
int x0=5;
*p0=x0;
cout<<"contenu p0="<<*p0<<endl;
cout<<"adresse p0="<<p0<<endl;
delete p0;
cout<<"contenu p0="<<*p0<<endl;
cout<<"adresse p0="<<p0<<endl;
```

Programme 3 :

```
int* p=new int;
cout<<"contenu p="<<*p<<endl;
cout<<"adresse p="<<p<<endl;
int x=5;
*p=x;
cout<<"contenu p="<<*p<<endl;
cout<<"adresse p="<<p<<endl;
delete p;
cout<<"after delete"<<endl;
cout<<"contenu p="<<*p<<endl;
cout<<"adresse p="<<p<<endl;
```

Programme 4 :

```
int* p=new int;
cout<<"contenu p="<<*p<<endl;
cout<<"adresse p="<<p<<endl;
int x=5;
*p=x;
cout<<"contenu p="<<*p<<endl;
cout<<"adresse p="<<p<<endl;

int* p2;

cout<<"contenu p2="<<*p2<<endl;
cout<<"adresse p2="<<p2<<endl;

p2=p;

cout<<"contenu p2="<<*p2<<endl;
cout<<"adresse p2="<<p2<<endl;
cout<<"before delete"<<endl;

delete p2;

cout<<"after delete p2"<<endl;
cout<<"contenu p après delete p2="<<*p<<endl;
cout<<"adresse p après delete p2="<<p<<endl;
cout<<"contenu p2 après delete p2="<<*p2<<endl;
cout<<"adresse p2 après delete p2="<<p2<<endl;
```

Exercice 2 : (Exemples sur les classes en C++)

Soit le code suivant en C++

```
15 class A{
16     int x;
17 public:
18     int get_x();
19 };
20 int A::get_x(){
21 return x;
22 }
23 int main() {
24     A a;
25     cout<<"a.x="<<a.get_x()<<endl;
26     return 0;
27 }
```

Questions :

1) Expliquer ce qui va se passer en mémoire à l'exécution de la ligne 24 ;
2) Déduire le résultat affiché à l'exécution de la ligne 25

Soit le code suivant :

```
15 class A{
16     int x;
17 public:
18     A(int x0);
19     int get_x();
20 };
21 A::A(int x0){
22 x=x0;
23 }
24 int A::get_x(){
25 return x;
26 }
27 int main() {
28     A a;
29     cout<<"a.x="<<a.get_x()<<endl;
30     return 0;
31 }
```

Questions :

1) Expliquer ce qui va se passer en mémoire à l'exécution de la ligne 28 ;
2) Déduire le résultat affiché à l'exécution de la ligne 29
3) Justifier

Exercice 3 : (Exemples simples de l'OO)

1) Soit l'objet cercle dont les cordonnées du centre sont (x0, y0), et le rayon est r. On est intéressé à faire un ensemble d'opérations sur cet objet : création, changement de coordonnées et du rayon, calcul de la surface. Proposer une solution orientée objet pour ce problème ?
2) On veut écrire un programme OO pour la résolution d'une équation de deuxième degré : $Ax^2+Bx+C=0$. Proposer les classes nécessaires ? écrire le programme OO.

Exercice 4 : (nombres complexes)

On s'intéresse à réaliser la classe des nombres complexes. Un nombre complexe est composé d'une partie réelle, et d'une partie imaginaire. Les méthodes qu'on propose pour cette classe sont les suivantes :

- double **module** () : fournit le module du nombre complexe (le module de x+y*i est la racine de x^2+y^2).
- Complex **oppose** () : fournit l'opposé du nombre complexe (l'opposé de x+y*i est –x-y*i).
- Complex **conjugue** () : fournit le conjugué du nombre complexe (le conjugué de x+y*i est x-y*i)
- Complex **plus** (Complex z) : fournit l'addition du nombre complexe et de z.
- Complex **moins** (Complex z) : fournit la soustraction du nombre complexe et de z.
- Complex **multiplier** (Complex z) : fournit la multiplication du nombre complexe et de z.
- void **ecritC** () : écrit le nombre complexe sous la forme (pReel + pImag i).

Exercice 5 : (Objet Date)

Proposer une classe date permettant de faire les services suivants :

- **date** (int j, int m, int an) : constructeur créant un objet date.
- **string** toString () : chaîne de caractères correspondant à la date de l'objet.
- **bool** bissex () : fournit **true** si l'année est bissextile, false sinon.
- **int** nbJoursEcoules () : nombre de jours écoulés depuis le début de l'année.
- **int** nbJoursRestants () : nombre de jours restant dans l'année.

2) On veut avoir en plus des services précédents, les deux services suivants :

- **bool** bissex (int annee) : fournit **true** si l'année est bissextile, false sinon.
- **long** nbJoursEntre (Date date1, Date date2) : fournit le nombre de jours écoulés entre les deux dates date1 et date2.

Exercice 6 : (Pile, File, Liste, Arbre)

On veut réaliser un programme orienté objet pour implémenter les structures Pile, File, Liste, Arbre binaire.

- Pile : Les opérations possibles sont :

```
void Créer ();          // Créer une pile vide
bool pileVide ();       //tester si la pile est vide
void empiler (int valeur);   // empiler une valeur
void depiler ();        // dépiler une valeur
void viderPile ();      // vider la pile
void listerPile ();     // lister les éléments de la pile
```

- File : Les opérations possibles sont :

```
void Créer ();          // Créer une file vide
bool fileVide ();       //tester si la file est vide
void Enfiler (int valeur);   // empiler une valeur
int Defiler ();         // dépiler à l'adresse de valeur
void viderFile ();      // vider la file
void listerFiler ();    // lister les éléments de la file
```

- Liste : Les opérations possibles sont :

```
void Créer ();          // Créer une liste vide
bool ListeVide ();      //tester si la liste est vide
bool Existe(int valeur) ;    //chercher un élément
void Inserer (int valeur);   // empiler une valeur
int  Supprimer(int valeur);  // dépiler à l'adresse de valeur
void viderListe ();     // vider la liste
void listerListe ();    // lister les éléments de la liste
```

- Arbre Binaire : Les opérations possibles sont :

```
void Créer_1 (int racine);          // Créer une arbre vide
void Créer_2 (int racine, int fils_g, int fils_d);   // Créer une arbre à 3 éléments
bool arbreVide ();      //tester si l'arbre est vide
bool Existe(int valeur) ;  //chercher une valeur
void parcours_pré ();   // parcours pré-ordre
void parcours_post ();  // parcours post ordre
```

1) Proposer une première solution avec une représentation contigüe (table statique). (si c'est possible)
2) Proposer une deuxième solution avec une représentation chaînée (les pointeurs).