
TD1 : Pointeurs en C++ et Initiation à la POO en C++

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;

```

solution : on aura une erreur car delete ne s'applique pas sur un pointeur qui n'était pas déjà allouée manuellement.

```

contenu p0=5
adresse p0=0x612c7950

```

6 [main] P2 8652 cygwin_exception::open_stackdumpfile: Dumping stack trace to P2.exe.stackdump

Programme 3 : ici il est possible de libérer l'adresse louée manuellement

```

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;

```

```

contenu p=1629734642
adresse p=0x80072010
contenu p=5
adresse p=0x80072010
after delete
contenu p=1630054688
adresse p=0x80072010

```

Programme 4 : le contenu de p est perdu, même si c'est un autre pointeur qui été libéré.

```

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;

```

```

contenu p=1629734642
adresse p=0x80072010
contenu p=5
adresse p=0x80072010
contenu p2=1432107587
adresse p2=0x28cc60
contenu p2=5
adresse p2=0x80072010
before delete
after delete p2
contenu p après delete p2=1630054688
adresse p après delete p2=0x80072010
contenu p2 après delete p2=1630054688
adresse p2 après delete p2=0x80072010

```

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 ;
Création d'un objet nommé a avec un seule attribut et donc uniquement l'espace mémoire nécessaire pour stocker x.
 - 2) Déduire le résultat affiché à l'exécution de la ligne 25
Il afficher la valeur aléatoire de x non initialisé.
- 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 ;
 Ici il va signaler une erreur. En effet, lors de cette déclaration le compilateur va chercher un constructeur implicite (par défaut), mais dans ce cas de déclaration il y'a déjà un constructeur !!!!
 Donc le constructeur implicite n'existe plus. Le problème c'est que A a ; signifie crée a avec un constructeur sans argument, alors que le seul constructeur qui existe dans A a un argument. Donc il y'a une erreur et le programme ne se compile pas.
- 2) Dédire le résultat affiché à l'exécution de la ligne 29
 Erreur
- 3) Justifier
 Justifié en réponse 1

Exercice 3 : (Exemples simples de l'OO)

- 1) Soit l'objet cercle dont les coordonnées 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 $x+yi$ est la racine de x^2+y^2).
- Complex **oppose** () : fournit l'opposé du nombre complexe (l'opposé de $x+yi$ est $-x-yi$).
- Complex **conjugue** () : fournit le conjugué du nombre complexe (le conjugué de $x+yi$ est $x-yi$)
- 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 + plmag 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 :

```
Pile ();           // 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
```

```
//représentation contigüe
#include <iostream>
using namespace std;

const int Max=20;
class Pile{
    int sommet;
    int max; // doit être inférieur à max
    int T[Max];
public:
    Pile (int maximum); // Créer une pile vide
    bool pileVide (); //tester si la pile est vide
    void empiler (int valeur); // empiler une valeur
    void depiler (); // dépiler à une valeur
    void viderPile (); // vider la pile
    void listerPile (); // lister les éléments de la pile
};

Pile::Pile(int maximum){
    max=maximum;
    sommet=(-1);
}

bool Pile::pileVide(){
    return sommet==(-1);
}

void Pile::empiler(int valeur){
    if (sommet<max-1) {
        sommet++;
        T[sommet]=valeur;
        //cout<<sommet<<endl;
        //cout<<"T["<<sommet<<"]="<<T[sommet]<<endl;
    }
}
```

```

    }
    else{
        cout<<"Pile Pleine"<<endl;
    }
}

void Pile::depiler () {
    if (not pileVide()){
        sommet--;
    }
    else cout<<"pile est vide"<<endl;
    // dépiler à une valeur
}

void Pile::viderPile () {
    sommet=(-1);
}

void Pile::listerPile () {
    for (int i=sommet; i>=0; i--){
        cout<<T[i]<<endl;
    }
}

Pile p= Pile(4);
p.empiler(4);
p.empiler(6);
p.empiler(9);
p.empiler(10);
p.empiler(14);
p.empiler(16);
cout<<"après ajout de 4, 6, 9, 10, 14, 16"<<endl<<endl;
p.listerPile();

p.depiler();

cout << "après dépiler" <<endl;
p.listerPile();
//cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!

p.viderPile();
cout<<"après vidage de la pile"<<endl;
p.listerPile();

```

chaine

```

#include <iostream>
using namespace std;

struct elem{
    int e;
    elem* suiv;
};

class Pile{
    elem* sommet;
    //int max;
    //int T[];
public:
    Pile (); // Créer une pile vide
    bool pileVide (); //tester si la pile est vide
    void empiler (int valeur); // empiler une valeur
    void depiler (); // dépiler à une valeur
    void viderPile (); // vider la pile
    void listerPile (); // lister les éléments de la pile
};

Pile::Pile () {
    sommet=NULL;
    //T=new int(maximum);
}

bool Pile::pileVide () {
    return (sommet==NULL);
}

```

```

void Pile::empiler(int valeur){
    if (pileVide()){
        cout<<"empiler:"<<valeur<<endl;
        sommet=new elem;
        sommet->e=valeur;
        sommet->suiv=NULL;
    }
    else{
        cout<<"empiler:"<<valeur<<endl;
        elem*ptr=new elem;
        ptr->e=valeur;
        ptr->suiv=sommet;
        sommet=ptr;
    }
}

void Pile::depiler () {
    if (not pileVide()){
        if (sommet->suiv==0){
            delete sommet;
            sommet=0;
        }
        else{
            elem*ptr=sommet;
            sommet=sommet->suiv;
            delete ptr;
            ptr=0;
        }
    }
    else cout<<"pile est vide"<<endl;
    // dépiler à une valeur
}

void Pile::viderPile () {
    while (not pileVide()){
        depiler();
    }
    /*
    elem* ptr=sommet;
    while (ptr!=NULL) {
        elem*ptr2=ptr;
        cout<<"élément supprimé:"<<ptr->e<<endl;
        ptr=ptr->suiv;
        delete ptr2;
    }
    sommet=0;
    */
}

void Pile::listerPile () {
    if (not pileVide()){
        elem* ptr=sommet;
        while (ptr!=NULL) {
            cout<<"valeur:"<<ptr->e<<endl;
            ptr=ptr->suiv;
        }
    }
}

}

int main() {
    Pile p= Pile();
    p.empiler(4);
    p.empiler(6);
    p.empiler(9);
    p.empiler(10);
    p.empiler(14);
    p.empiler(16);
    cout<<"après ajout de 4, 6, 9, 10, 14, 16"<<endl<<endl;
    p.listerPile();
    p.depiler();
    cout << "après dépiler" <<endl;
}

```

```

    p.listerFile();
    //cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    p.viderFile();
    cout<<"après vidage de la Pile"<<endl;
    p.listerFile();
    return 0;
}

```

- File : Les opérations possibles sont :

```

File ();           // 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

```

Contigüe

```

#include <iostream>
using namespace std;

const int Max=20;
class File{
    int tete;
    int queue;
    int max; // doit etre inférieur à max
    int T[Max];
public:
    File (int maximum); // Créer une file vide
    bool fileVide (); //tester si la file est vide
    void enfiler (int valeur); // enfiler une valeur
    void defiler (); // défiler à une valeur
    void viderFile (); // vider la file
    void listerFile (); // lister les éléments de la file
};

File::File(int maximum){
    max=maximum;
    tete=(-1);
    queue=(-1);
}

bool File::fileVide() {
    return queue==(-1);
}

void File::enfiler(int valeur){
    if (queue<max-1) {
        if (tete==(-1)){
            tete++;queue++;
            T[tete]=valeur;
        }
        else{
            queue++;
            T[queue]=valeur;
        }
        //cout<<sommet<<endl;
        //cout<<"T ["<<sommet<<"]="<<T[sommet]<<endl;
    }
    else{
        cout<<"File Pleine"<<endl;
    }
}

void File::defiler () {
    if (not fileVide()){
        queue--;
    }
}

```

```

    }
    else cout<<"file est vide"<<endl;
    // dépiler à une valeur
}

void File::viderFile() {
    tete=(-1); queue=(-1);
}

void File::listerFile() {
    if (not fileVide())
        for (int i=tete; i<=queue; i++){
            cout<<T[i]<<endl;
        }
}

}

int main() {
    File f= File(4);
    f.enfiler(4);
    f.enfiler(6);
    f.enfiler(9);
    f.enfiler(10);
    f.enfiler(14);
    f.enfiler(16);
    cout<<"après ajout de 4, 6, 9, 10, 14, 16"<<endl<<endl;
    f.listerFile();
    f.defiler();
    cout << "après défiler" <<endl;
    f.listerFile();
    //cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    f.viderFile();
    cout<<"après vidage de la file"<<endl;
    f.listerFile();
    return 0;
}

```

Chainée

```

#include <iostream>
using namespace std;

//const int Max=20;
struct elem{
    int e;
    elem* suiv;
};

class File{
    elem* tete;
    elem* queue;
    //int max; // doit etre inférieur à max
    //int T[Max];
public:
    File (); // Créer une file vide
    bool fileVide (); //tester si la file est vide
    void enfiler (int valeur); // enfiler une valeur
    void defiler (); // défiler à une valeur
    void viderFile (); // vider la file
    void listerFile (); // lister les éléments de la file
};

File::File() {
    //max=maximum;
    tete=0; //NULL
    queue=0; //NULL
}

bool File::fileVide() {
    return queue==0;
}

void File::enfiler(int valeur) {
    //if (queue<max-1) {
        if (tete==NULL) {
            tete=new elem;
            tete->e=valeur;

```

```

        tete->suiv=0;
        queue=tete;
        //T[tete]=valeur;
    }
    else{elem* ptr=new elem;
    ptr->e=valeur;
    ptr->suiv=NULL;
    queue->suiv=ptr;
    queue=ptr;
    //T[queue]=valeur;
    }
    //cout<<sommet<<endl;
    //cout<<"T["<<sommet<<"]="<<T[sommet]<<endl;
//}
//else{
    //cout<<"File Pleine"<<endl;
//}
}

void File::defiler (){
    if (not fileVide()){
        if (tete==queue){
            delete tete;
            tete=0;
            queue=0;
        }
        else {
            elem* ptr=tete;
            while(ptr->suiv!=queue) ptr=ptr->suiv;
            elem *ptr2=queue;
            queue=ptr;
            delete ptr2;
            ptr2=0;
            queue->suiv=0;
        }
    }
    else cout<<"file est vide"<<endl;
    // dépiler à une valeur
}

void File::viderFile () {
    //tete=(-1); queue=(-1);
    while (not fileVide()) defiler();
}

void File::listerFile(){
    if (not fileVide()){
        elem* ptr=tete;
        while (ptr!=NULL){
            cout<<"valeur="<<ptr->e<<endl;
            ptr=ptr->suiv;
        }
    }
}

int main() {
    File f= File();
    f.enfiler(4);
    f.enfiler(6);
    f.enfiler(9);
    f.enfiler(10);
    f.enfiler(14);
    f.enfiler(16);
    cout<<"après ajout de 4, 6, 9, 10, 14, 16"<<endl<<endl;
    f.listerFile();
    f.defiler();
    cout << "après défiler" <<endl;
    f.listerFile();
    //cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    f.viderFile();
    cout<<"après vidage de la file"<<endl;
    f.listerFile();
}

```

```
return 0;
}
```

- Liste : Les opérations possibles sont :

```
Liste ();          // Créer une liste vide
bool ListeVide (); //tester si la liste est vide
bool Existe(int valeur); //chercher un élément
void Insérer (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
```

```
#include <iostream>
using namespace std;

struct elem{
    int e;
    elem* suiv;
};

class Liste{
    elem* tete;
    elem* queue;

public:
    Liste ();          // Créer une liste vide
    bool ListeVide (); //tester si la liste est vide
    bool Existe(int valeur, elem* &adr) ; //chercher un élément
    void adr_prec(elem* adr, elem* &adr_pre);
    void Insérer_debut (int valeur); // insérer une valeur
    void Insérer_fin (int valeur); // insérer une valeur
    void Insérer_pos (int valeur, int pos); // empiler une valeur
    void Supprimer(int valeur); // dépiler à l'adresse de
    valeur
    int get_tete() ; //retourne la valeur de la tête
    int get_queue() ; //retourne la valeur du queue
    void viderListe (); // vider la liste
    void listerListe (); // lister les éléments de la liste
};

Liste::Liste() {
    tete=0; queue=0;
}

bool Liste::ListeVide() {
    return (queue==NULL);
}

bool Liste::Existe(int valeur, elem* &adr) {
    //chercher un élément
    bool existe=false; elem* ptr=tete;
    while(not existe and ptr!=NULL) {
        if (ptr->e==valeur) {
            existe=true;
            adr=ptr;
        }
        ptr=ptr->suiv;
    }
    return existe;
}

void Liste::Insérer_debut (int valeur) {
    // insérer une valeur
    elem* ptr=new elem;
    ptr->e=valeur;
    ptr->suiv=tete;
```

```

        tete=ptr;
        if (ListeVide()) queue=tete;
    }

void Liste::Inserer_fin (int valeur){
    // insérer une valeur
    elem* ptr=new elem;
    ptr->e=valeur;
    ptr->suiv=NULL;
    if (ListeVide()) {
        tete=ptr; queue=tete;
    }
    else {
        queue->suiv=ptr;
        queue=queue->suiv;
    }
}

void Liste::Inserer_pos (int valeur, int pos){
    // trouver la position, ensuite faire le chainage
}

void Liste::adr_prec(elem* adr, elem* &adr_pre){
    adr_pre=tete;
    while(adr_pre!=NULL){
        if(adr_pre->suiv==adr){
            break;
        }
        else adr_pre=adr_pre->suiv;
    }
}

void Liste::Supprimer(int valeur){
    // supprimer une valeur
    elem* adr=0;
    if (Existe(valeur, adr)){
        if(adr==tete){//suppression du début
            tete=tete->suiv;
            if (tete==NULL) queue=NULL;
            delete adr;
        }
        else{
            elem*adr_pre=0;
            adr_prec(adr, adr_pre);
            adr_pre->suiv=adr->suiv;
            delete adr;
        }
    }
    else{
        cout<<valeur<<" n'est pas dans la liste"<<endl;
    }
}

}

int Liste::get_tete(){
    //retourne la valeur de la tête
    return 0;
}

int Liste::get_queue() {
    //retourne la valeur du queue
    return 0;
}

void Liste::viderListe (){
    // vider la liste
    while (not ListeVide()){
        int val=tete->e;
        Supprimer(tete->e);
        cout<<val<<" est supprimé"<<endl;
    }
}

void Liste::listerListe (){
    if (not ListeVide()){
        elem*ptr=tete;
    }
}

```

```

        while (ptr!=NULL) {
            cout<<"valeur:"<<ptr->e<<endl;
            ptr=ptr->suiv;
        }
    }
    else cout<<"liste vide"<<endl;
}

```

- Arbre Binaire : Les opérations possibles sont :

```

arbre (int racine);           // Créer une arbre vide
arbre (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 Ajouter (int valeur);    // ajouter une valeur
void Supprimer (int valeur);  // supprimer une valeur valeur
void parcours_pre ();         // parcours pré-ordre

```

```

parcourir ( Arbre )
si Arbre ≠ ∅ alors
    Traiter-1 (info(Arbre.Racine)) ;
    parcourir ( Arbre.filsG ) ;
    parcourir ( Arbre.filsD ) ;
Fsi

```

```

void parcours_post ();        // parcours post ordre

```

```

parcourir ( Arbre )
si Arbre ≠ ∅ alors
    parcourir ( Arbre.filsG ) ;
    parcourir ( Arbre.filsD ) ;
    Traiter-3 (info(Arbre.Racine)) ;
Fsi

```

- 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).