CYCLE DE VIE DES OBJETS (1)

CONSTRUCTION, INITIALISATION, AFFECTATION

➢ Cycle de vie d'un objet

Ensemble des états par lequel il passe au cours de l'exécution d'un programme :
- depuis sa **création**
- jusqu'à sa **destruction**,
- en passant par son **initialisation**, sa modification, sa copie, etc.

Bien maîtriser ce qui se passe durant la vie d'un objet :
- Creation et initialization d'un objet
- Copie d'un objet dans un autre objet de même type
- Affectation d'un objet dans un autre objet de même type (avec =)
- Destruction d'un objet, etc.

➢ Rappel : constructeur et destructeur

Constructeur : fonction membre appelée automatiquement lors de la création d'un objet :
- par une définition : `chaine a("azert");`
- par une allocation dynamique :

```
chaine *pa= new chaine ("azert");
```

Destructeur : fonction membre appelée automatiquement lorsqu'un objet est détruit :
- fin d'un bloc { X x ; } pour une variable locale
- fin du programme pour les variables globales
- appel explicite de **delete** pour un objet alloué dynamiquement avec **new**

➢ Rappel : constructeur par défaut du C++

Le C++ fournit un constructeur par défaut, sans paramètre.

- initialise tous les attributs à leurs valeurs par défaut
  - (entier et flottant à 0, pointeurs à NULL, objets avec leur constructeur par défaut...)
- n'existe plus dès qu'on définit un constructeur pour la classe
- peut être redéfini.

CYCLE DE VIE DES OBJETS (2)

CONSTRUCTION, INITIALISATION, AFFECTATION

➢ Rappel : constructeur par copie : `X(const X&)`

Constructeur qui réalise une copie d'un objet de classe X dans un autre objet de classe X.

Appelé :

- définition :

```
chaine a("azert"), b(a);
```

- passage de paramètres par valeur :

```
void f (X a) { /* ••• */ };
main() { ••• f(x); •••} // Copie de x dans a
```

- valeur de retour d'une fonction :

```
X f () { X a; /* ••• */ return a; };
main() { X x; •••x=f(); •••}
                    // Copie de a dans x
```

Constructeur de copie par défaut:

appelle constructeur de copie sur tous les attributs de l'objet copié.

Pour les attributs scalaires, copie bit à bit

Pour les attributs objets, execute le constructeur par copie

➢ Opérateur d'affectation : `X& operator=(const X&) :`

Opérateur membre qui affecte l'objet passé en paramètre à ***this**.

Fonction membre exécutée lors de l'utilisation de **=**, dans un code:

```
main() { X a=2,b=3   ;  a=b;}
```

Opérateur d'affectation par defaut

appelle l'opérateur d'affectation pour chacun des attributs de l'objet affecté.

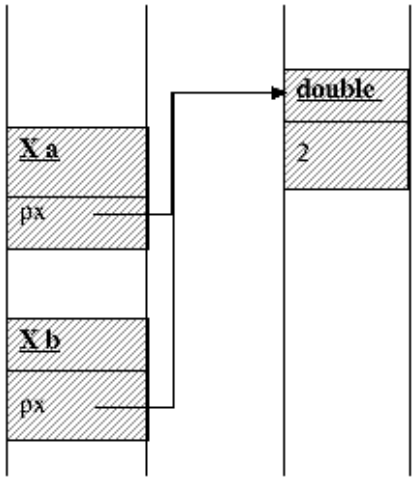
CYCLE DE VIE DES OBJETS ET ALLOCATION DYNAMIQUE (1)

Quelles sont les erreurs commises dans cette classe X ?

```
class X {
    friend X operator+(const X & a, const X & b);
    double * px;
    X()             { px = NULL } // privé
public :
    X(double x)     { px = new double (x); }
    ~X()            { delete px; }
    void setDbl(double r)   { *px = r ; }
    void aff() const        { cout << *px << endl ; }
};
X operator+(const X & a, const X & b) {
    X r;
    // Concaténation de 2 X
    r.px = new double( *a.px + *b.px ) ;
    return r;
}
void affX(X x) { cout << * x.px<<endl; }
```

➢ Exemple 1

```
main() {
    X a(2);
    X b(a); // a.px == b.px !
    b.setDbl(4);
    a.aff() ; // affiche 4 !
}   // destruction de a (libère a.px)
    // et de b (libère b.px : ERREUR)
```



➢ Exemple 2

```
main() {
    X a(2);
    X b = a;   // b.px = a.px : partage de pointeurs !
 // Toute modification de *a.px ou *b.px modifie l'autre

}   // destruction     de b (libère b.px)
    //                 de a (libère a.px : ERREUR)
```

CYCLE DE VIE DES OBJETS ET ALLOCATION DYNAMIQUE (2)

➢ Exemple 3

```
main() {
    X a(2);
    affX(a);   // Copie de a dans x : x.p=a.p
    // Après exécution : libère x.p !
    •••
    // Impossible d'utiliser a.p car il est déjà libéré !
}   // destruction     de a (libère a.p) :  ERREUR
```

➢ Exemple 4

```
main() {
    X a(1);    X b(2);
    X c = b+a; // Copie de r dans un objet temporaire tmp
    // Copie de tmp dans c
    // Libère r.px car locale à operator+()
    // Impossible d'utiliser c.px
    // car il est déjà libéré à travers r !
}   // destruction de c (libère c.p : ERREUR)
```

➢ Conclusion

Encapsulation des allocation/libération mémoire : toute classe qui alloue de la mémoire doit se préoccuper :

* de la destruction de la mémoire allouée
* de ce qui se passe quand des instances sont copiées ou affectées

En particulier une classe avec allocation dynamique doit comporter les fonctions suivantes :

- **constructeur par défaut**
- **constructeur par copie (appel lors du passage de valeur et retour de fonction)**
- **destructeur**
- **opérateur d'affectation**

CYCLE DE VIE DES OBJETS ET ALLOCATION DYNAMIQUE (3)

Reprenons pour conclure notre classe X :

```
class X {

    friend X operator+(const X & a, const X & b);

    double * px;
    X()             { px = NULL } // privé

public :

    X(double x)     { px = new double (x); }

    // Constructeur de copie
    X( const X & other)     {
      px = NULL ;
      if(other.px()
          px = new double ( * other.px );
    }

    // Destructeur
    ~X()            { delete px; }

    // Opérateur d'affectation. Affecte y à *this
    X& X::operator=(const X& y){
      if (this == &y) return *this;
      if (px) delete px;
      px = NULL ;
      if( y.px )
          px = new double ( * y.px );
      return(*this);
    }

    void setDbl(double r)   { *px = r ; }
    void aff() const        { cout << *px << endl ; }
};
```

SURCHARGE DE L'OPERATEUR = ET *SELF ASSIGNEMENT*

➢ Le problème du self-assignement

Reprrenons la classe X et considérons l'opérateur d'affectation:

```
X& X::operator=(const X & y){
     if (px) delete px;
     px = NULL ;
     if( y.px )
          px = new double ( * y.px );
     return(*this);
   }
main() {
     X x;
     x = x;
}
```

Que se passe-t-il ?

➢ Deux canevas pour l'opérateur d'affectation =

Canevas 1 : verifier que y n'est pas *this:

```
X& X::operator=(const X& y){
   if (this == &y) return *this;
   if (px) delete px;
   px = NULL ;
   if( y.px )
        px = new double ( * y.px );
   return(*this);
  }
```

Canevas 2 : copier d'abord les attributs

```
X& X::operator=(const X &y) {
   double *tmp = NULL; // temporaire
   if( y.px) tmp = new double ( * y.px ) ;
   if (px) delete px; // delete des attributs
   px=tmp;
   return *this;
  }
```

EXEMPLE : UNE CLASSE CHAINE DE CARACTERES (1)

```
class chaine {
    int t;          // Le nb de char alloues, y compris '\0'
    char *p;        // La chaîne stockée
public:
// constructeurs, destructeur, opérateur =
    chaine();                       // chaine x; (vide)
    chaine(const char *s);          //chaine x="ab" & chaine("ab")
    chaine(const chaine & y);  //chaine x(y), return(x)...
    chaine (char c, int n = 1);// n caractères c
    ~chaine ();                     // Destructeur
    chaine& operator=(const chaine&);// x=y;
    chaine& operator=(const char *); // x="abc" :

// accesseurs et modifieurs
    bool isNull() const { return t == 0; }
    // nb de caractères sans '\0' ; -1 si isNull()
    int length() const { return t-1 ;}
    // => isNull() devient true
    void clear() { delete [ ] p; p = NULL; t = 0; }
    //opérateurs []
    char  operator[](int i) const;// cout << x[i] ;
    char& operator[](int i);   // x[i]='a'; cin >> x[i]
    // conversion chaîne -> en const char *. A eviter
    operator const char* () const { return p; }

    // sous chaine débutant a pos et de taille len
    chaine substr (int pos = 0, int len = -1) const;
    // recherche de chaine
    int find (const chaine& s, int pos = 0) const;
    // concaténation "en place"
    chaine& operator+= (const chaine& str);

// Fonctions amies
    // égalité de 2 chaînes
    friend int operator==(const chaine& ,const chaine& );
    // concaténation
    friend chaine operator+(const chaine& ,const chaine& );
    // E/S
    friend ostream& operator<<(ostream&, const chaine &);
    friend istream& operator>>(istream&, chaine &);
};
```

EXEMPLE : UNE CLASSE CHAINE DE CARACTERES (2)

```
//******* Constructeurs, destructeur, opérateur =
chaine::chaine (): t(0), p(NULL) { }

chaine::chaine (const char *s): t(0), p(NULL) {
    if(s) { strcpy(p=new char[ t=strlen(s)+1 ], s); }
}

chaine::chaine (const chaine & y): t(0), p(NULL) {
    if(y.p) { strcpy(p=new char[ t=y.t ], y.p); }
}

chaine::chaine (char c, int n): t(n), p(NULL) {
    if ( n < 0 )  { cerr<<"Erreur A"; exit(1); }
    p = new char[ n + 1 ];
    memset(p, c, n);        p[n] = '\0' ;
}

chaine::~chaine () { delete [ ] p; }

chaine& chaine::operator=(const chaine& y) {
    if(this == &y) { *this; }
    if (t) delete [] p;
    if(y.t>0) { strcpy(p=new char[ y.t ], y.p); }
    else { t = 0; p = NULL ; }
    return *this ;
}

chaine& chaine::operator=(const char  *s) {
    if (t) delete [] p;
    if(s) { strcpy(p=new char[ t=strlen(s)+1 ], s); }
    else { t = 0; p = NULL ; }
    return *this ;
}

//******* accesseurs et modifieurs
char & chaine::operator[](int i) {
    if (i<0 || i>t) { cerr<<"Erreur B"; exit(1); }
    return p[i];
}
char chaine::operator[](int i) const {
    if (i<0 || i>t) { cerr<<"Erreur C"; exit(1); }
    return p[i];
}
```

EXEMPLE : UNE CLASSE CHAINE DE CARACTERES (3)

```
// retourne sous chaine commencant a pos et de taille len
chaine chaine::substr (int pos, int len) const {
    if ( pos < 0 || pos >= t) {
        cerr<<"Erreur D"; exit(1);
    }
    // correction de len
    if(len < 0) len = t;
    if(pos+len > t) { len = t-pos; }
    if(len <= 0) return chaine();
    chaine s;
    s.t=len;
    strncpy( s.p = new char[ len +1 ], p + pos, len) ;
    s.p[len] = '\0'; // null-terminated
    return s;
}

// recherche de chaine. Retourne -1 si pas trouvée
int chaine::find (const chaine& s, int pos) const {
    if(pos < 0)   { cerr<<"Erreur E"; exit(1); }
    if(pos >= t) { return -1; }
    char * found =  strstr( p+pos, s.p) ;
    if(found == NULL) return -1;
    return found - p ;
}


chaine& chaine::operator+= (const chaine& y) {
    if(y.p == NULL || y.p[0] == '\0') { return *this ; }
    int newT = t + strlen(y.p) ;
    char * tmp = new char[ newT ] ;
    if(t) {
        strcpy( tmp , p) ;
        strcpy( tmp + t-1, y.p);
    } else {
        strcpy( tmp , y.p) ;
    }
    t =  newT;
    delete [ ] p;
    p = tmp;
    return *this;
}
```

EXEMPLE : UNE CLASSE CHAINE DE CARACTERES (4)

```
//******* Fonctions et opérateurs amis
ostream& operator<<(ostream& s, const chaine & x) {
    if (x.p) { s << x.p; }
    return s;
}

istream& operator>>(istream& s, chaine & x) {
     // Attention : pas de test de la longueur
    char buf[256];
    s >> buf;
    x=buf;
    return s;
}

int operator==(const chaine& x,const chaine& y) {
    if( x.p == NULL && y.p == NULL) return 1;
    if( x.p == NULL || y.p == NULL) return 0;
    return strcmp(x.p, y.p)==0;
}

chaine operator+(const chaine& a,const chaine& b) {
    chaine r;
    r.t = a.t + b.t -1;         // -1 : un seul '\0'
    if(r.t<0) r.t = 0;// cas a et b null
    if(r.t) {
     r.p = new char[ r.t ] ;
     if(a.p) {
          strcpy( r.p , a.p) ;
     }
     if(b.p) {
          if(a.t>0) {
               strcpy( r.p + a.t-1, b.p);
          } else {
               strcpy( r.p, b.p);
          }
     }
    }
    return r;
}
```

EXEMPLE : UNE CLASSE CHAINE DE CARACTERES (5)

```
//******* Test
chaine f(chaine x, chaine y) { r = x+y; return r; }

main() {
   chaine a("chaine 1");                  // Constructeur 2
   cout << "a = "<< a << endl;      //a = chaine 1
   printf("a = %s\n", (const char*) a );    //a = chaine 1

   chaine x[100];           // Tab 100 chaînes "null"
   cin >> x[0];         // operateur >>
   x[1] = x[0];         // operateur affectation
   x[1] = x[1];         // self assignement
   x[2] = chaine('a', 12);// Constructeur 3
   x[2][3] = 'B';        // operateur [] non const
   x[3]=f(x[0],x[1]); // Concaténation, passage par valeur
   x[4]="salut"   ;     // operateur =(const char *)
   x[4]+= " veut dire bonjour" ;   // op += et conversion
   x[5] = x[4].substr(6, 4);        // substr()

   for(int i = 0 ; i < 5 ; i ++ ) {
     cout << "x["<<i<<"] = " << x[i] << endl;
   }

   cout << "x[3].length() = " << x[3].length() << endl;
   cout << " isNull() ? " << chaine("").isNull() << endl;
   cout << "find " << x[4].find("dire", 5) << endl;
   cout << "a == a "<< (a == a) <<endl;
   cout << "vide==null ? : "<<(chaine("")==chaine())<<endl;
} // Appel des destructeurs ici
```

➢ Sortie

On suppose que l'utilisateur tape 'A'

```
a = chaine 1                    x[4] = salut veut dire bonjour
a = chaine 1                    x[5] = veut
A                               x[3].length() = 2
x[0] = A                        isNull() ? 0
x[1] = A                        find 11
x[2] = aaaBaaaaaaaa             a == a 1
x[3] = AA                       vide==null ? : 0
```