Chapitre 7

ATTRIBUTION DYNAMIQUE D'ESPACE MÉMOIRE

POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Le type pointeur

- Une variable pointeur correspond à une variable qui contient l'adresse mémoire d'une donnée.
- Déclaration: *int* ptrEntier;*
- L'opérateur unaire & «adresse de» est utilisé pour connaître l'adresse d'une donnée.

```
int entier;

int* ptrEntier = &entier;
```

- L'identificateur d'une chaîne de caractères C contient l'adresse du premier caractère de la chaîne, il correspond donc à un pointeur.

étoile (*) vs éperluète (&)

(Note: « perluète » est le terme privilégié par l'Office québécois de la langue française, mais les termes « et commercial », « esperluette », « esperluète », et « éperluète » sont aussi communs)

Leur signification diffère selon leur emplacement:

Dans une déclaration: (i.e. « int* a », « int& a »)

- * définit un pointeur;
- & définit une référence a une variable, c'est-à-dire un nouveau nom permettant d'accéder à cette même variable; la référence doit être déclarée en paramètre ou être **absolument initialisée à la déclaration**

Dans les instructions, en opérateur unaire: (i.e. « *a », « &a »)

- * représente la déréférence, soit « le contenu à l'adresse pointée »
- & représente « l'adresse de »

Dans les instructions, en opérateur binaire: (i.e. « a * b », « a & b »)

- ce sont les opérations de multiplication (*) et de ET binaire (&).

Le type pointeur

```
    int x = 5; // x est une variable qui contient un entier.
    int* pointeur; // pointeur est une variable qui contient l'adresse mémoire
d'un entier.

    pointeur = &x; // Met l'adresse de x dans pointeur;
                   // x doit exister, mais n'a pas besoin d'avoir de valeur.

    cout << "L'adresse de x est " << &x << endl;
    cout << "Le contenu de x est " << x << endl;
    cout << "L'adresse de pointeur est " << &pointeur << endl;
    cout << "Le contenu de pointeur est " << pointeur << endl;
    cout << "Le contenu pointé par pointeur est " << *pointeur << endl;
    // Affichage:
    // L'adresse de x est 0013FF3C
    // Le contenu de x est 5
    // L'adresse de pointeur est 0013FF38
    // Le contenu de pointeur est 0013FF3C
    // Le contenu pointé par pointeur est 5
```

VII_type_pointeur.cpp

Le type pointeur

```
cout << "Changement du contenu de pointeur." << endl;
pointeur = new int;  // Adresse d'un emplacement vide pour un entier.
cout << "L'adresse de pointeur est " << &pointeur << endl;
cout << "Le contenu de pointeur est " << pointeur << endl;
cout << "Le contenu pointé par pointeur est " << *pointeur << endl;
cout << endl;
cout << "Changement du contenu pointé par pointeur." << endl;
// Un pointeur doit contenir une adresse valide avant de le déréférencer.
// Un pointeur non initialisé ne pointe pas vers une adresse valide.
*pointeur = 1234;
cout << "L'adresse de pointeur est " << &pointeur << endl;
cout << "Le contenu de pointeur est " << pointeur << endl;
cout << "Le contenu pointé par pointeur est " << *pointeur << endl;
delete pointeur; // Recycle l'emplacement alloué.
// Affichage:
// Changement du contenu de pointeur.
// L'adresse de pointeur est 0013FF38
// Le contenu de pointeur est 00032F30
// Le contenu pointé par pointeur est 0
//
// Changement du contenu pointé par pointeur.
// L'adresse de pointeur est 0013FF38
// Le contenu de pointeur est 00032F30
// Le contenu pointé par pointeur est 1234
```

VII_type_pointeur.cpp

Le type pointeur

```
int     ageAlex = 7, ageSandie = 27;
double  x = 1.2345, y = 32.14;

int* ptrEntier = &ageAlex;
*ptrEntier += ageSandie;
cout << "ageAlex est " << *ptrEntier << endl;

double* ptrReel = &x;
y += 5 * (*ptrReel);
cout << "Y contient la valeur " << y << endl;

char a[] = "A";
const char* car1 = a;  // La valeur *car1 ne peut être modifiée
char* const car2 = a;  // Le pointeur car2 ne peut être modifié
cout << car1 << car2 << endl;

// Affichage:
// ageAlex est 34
// Y contient la valeur 38.3125
// AA
```

VII_type_pointeur.cpp

Allocation dynamique

- Permet d'allouer de l'espace à une variable lors de l'exécution et de libérer cet espace lorsque la variable devient inutile.
- Allocation dynamique s'effectue à l'aide de l'opérateur *new* qui exige le type de la variable dynamique.

```
varPointeur = new type_donnée_pointée;
```

- Exemple:

```
int* ptrEntier = new int;
```

Allocation dynamique

- La libération de l'espace mémoire alloué dynamiquement s'effectue à l'aide de l'opérateur *delete* qui exige de préciser le pointeur correspondant

```
delete varPointeur;
```

- Exemple:

```
int* ptrEntier = new int;
*ptrEntier = 12;
delete ptrEntier;
```

New et delete

```
int main()
{
    int index, *ptrInt1, *ptrInt2;

    ptrInt1 = &index;
    *ptrInt1 = 77;
    ptrInt2 = new int;
    *ptrInt2 = 173;
    cout << "Les valeurs sont " << index << "  "
         << *ptrInt1 << "  " << *ptrInt2 << endl;
    ptrInt1 = new int;
    ptrInt2 = ptrInt1;
    // La mémoire originalement pointée par ptrInt2 est maintenant perdue.
    *ptrInt1 = 999;
    cout << "Les valeurs sont " << index << "  "
         << *ptrInt1 << "  " << *ptrInt2 << "\n";
    delete ptrInt1;
    ptrInt1 = 0;
    // delete ptrInt2; donne une erreur car ptrInt2 = ptrInt1 (même adresse pointée).
    delete ptrInt2; // N'affiche pas d'erreur avec VC 2008, mais c'est une erreur.
}
```

VII_new_delete_1.cpp

New et delete (suite)

```
int main()
{
    double *ptrReel1, *ptrReel2 = new double;

    ptrReel1 = new double;
    *ptrReel2 = 3.14159;
    *ptrReel1 = 2.4 * (*ptrReel2);
    delete ptrReel2;
    ptrReel2 = 0;
    delete ptrReel1;
    ptrReel1 = 0;
}
```

VII_new_delete_2.cpp

Tableau dynamique

```
// Déclaration d'un tableau d'entiers à une dimension.
int* vecteur;

// Attribution d'un tableau de 150 entiers.
vecteur = new int[150];

// Adresses du tableau et des cases.
cout << "Adresse du tableau: " << vecteur << endl;
for (int i=0; i<3; i++)
    cout << "&vecteur[" << i << "] est " << &vecteur[i] << "; ";
cout << endl;

// Initialise chaque valeur du tableau à 0.
for (int i=0; i<150; i++)
    vecteur[i] = 0;

// Remet en disponibilité l'espace mémoire du tableau d'entiers.
delete[] vecteur;
vecteur = 0;

// Affiche:
// Adresse du tableau: 00032F00
// &vecteur[0] est 00032F00; &vecteur[1] est 00032F04; &vecteur[2] est
00032F08;
```

VII_tableau_dynamique.cpp

Tableau dynamique

```
// Déclaration d'un tableau de réels à deux dimensions,
// pemettant des lignes de longueurs différentes:
double** matrice;

// Attribution de l'espace pour un tableau de 20 lignes x 30 colonnes:
matrice = new double* [20];           // Espace pour les 20 pointeurs de lignes.
for (int ligne=0; ligne<20; ligne++)  // Pour chaque ligne,
    matrice[ligne] = new double[30];  // espace pour 30 colonnes de la ligne.

// Initialise chaque valeur de la matrice à zéro:
for (int ligne=0; ligne<20; ligne++)
    for (int colonne=0; colonne<30; colonne++)
        matrice[ligne][colonne] = 0.0;

// Libération de l'espace mémoire:
for (int ligne=0; ligne<20; ligne++) // Pour chaque ligne,
    delete[] matrice[ligne];         // libérer l'espace pour la ligne.

delete[] matrice;  // Libérer l'espace mémoire des pointeurs de lignes.
matrice = 0;
```

VII_tableau_dynamique.cpp

Tableau dynamique (suite)

```
// Déclaration d'un tableau de réels à "deux dimensions",
// en utilisant un tableau une dimension:
double* matrice2;

// Attribution de l'espace pour un tableau de 20 lignes x 30 colonnes:
int largeurMatrice2 = 30;
int hauteurMatrice2 = 20;
matrice2 = new double [hauteurMatrice2 * largeurMatrice2];

// Initialise chaque valeur de la matrice à zéro:
for (int ligne=0; ligne<20; ligne++)
    for (int colonne=0; colonne<30; colonne++)
        matrice2[ligne*largeurMatrice2 + colonne] = 0.0;

// Libération de l'espace mémoire:
delete[] matrice2;
```

VII_tableau_dynamique.cpp

Chaîne de caractères C dynamique

```
char* ptrCar;              // Déclaration d'une chaîne de caractères dynamique.

ptrCar = new char[37];             // Attribution d'espace pour 37 caractères.
strcpy(ptrCar, "Premiere heure");  // Mémorisation d'une chaîne dans cet espace.
cout << ptrCar;            // Affiche la chaîne.

delete[] ptrCar;           // Remet en disponibilité l'espace mémoire.
ptrCar = 0;
```

VII_chaineC_dynamique.cpp

New et delete et enregistrement

```
struct Date {
    int mois;
    int jour;
    int annee;
};

int main()
{
    Date* ptrDate;
    ptrDate = new Date;
    ptrDate->mois = 10;
    ptrDate->jour = 18;
    ptrDate->annee = 1938;
    cout << ptrDate->mois << "/"  << ptrDate->jour << "/"
         << ptrDate->annee << endl;
    delete ptrDate;
    ptrDate = 0;
}
```

`ptrEnregistrement->champ` est équivalent à `(*ptrEnregistrement).champ` et permet d'accéder au champ de l'enregistrement pointé.

VII_new_delete_3.cpp