heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Rappels sur l'allocation dynamique

Allocation / Construction / Destruction / Libération

- Cette présentation résume les différentes méthodes en C++ pour
  - allouer / libérer de la mémoire
  - de construire / détruire des objets dans cette mémoire allouée

- Vous avez le choix de la méthode d'allocation / construction

- Vous n'avez pas le choix de la méthode de libération / destruction. Il faut utiliser celle qui correspond à l'allocation / construction à cette adresse mémoire

- Pour les laboratoires d'ASD1
  - LinkedList utilise exclusivement l'approche p1
  - ResizableArray utiliser les approches p5, p7, p8 et p9

```
T* p0 = new T();          // alloue et construit par défaut
                          // un élément de type T

delete p0;                // détruit et libère la mémoire
```

```
T* p1 = new T(params); // alloue et construit un élément
                       // de type T avec les paramètres
                       // params

delete p1;             // détruit et libère la mémoire
```

```
T* p2 = new T(t);         // alloue et construit par copie de
                          // l'élément t

delete p2;                // détruit et libère la mémoire
```

```
T* p3 = new T[N];    // alloue un tableau de N éléments
                     // de type T

                     // Construit tous les éléments
                     // par défaut si le type T a un
                     // constructeur explicitement défini

                     // N'initialise rien si le type est
                     // POD (Plain Old Data), tel que int ou
                     // double

delete[] p3;         // détruit tous les éléments et libère
                     // la mémoire du tableau
```

```
T* p4 = new T[N](); // alloue un tableau de N éléments
                    // de type T.Construit tous les éléments
                    // par défaut quelque soit le type T

delete[] p4;        // détruit tous les éléments et libère
                    // la mémoire du tableau
```

```
T* p5 = (T*) ::operator new(N * sizeof(T));
                          // alloue un tableau de N éléments
                          // mais ne construit rien.

::operator delete(p5); // libère le tableau sans détruire
                          // les éléments
```

```
T* p6 = (T*) malloc(N * sizeof(T));
                        // équivalent en C de ::operator new.
                        // une différence: malloc retourne NULL
                        // au lieu de lancer std::bad_alloc en
                        // cas d'erreur

free(p6);               // équivalent C de ::operator delete
```

```
new(p7) T();      // construit un élément par défaut
                  // a l'emplacement mémoire spécifié
                  // par le pointer p67.

                  // Cet emplacement doit
                  //   1. avoir été alloué
                  //   2. ne pas avoir été construit

p7->~T();         // détruit l'élément pointé par p7, sans
                  // libérer la mémoire à l'emplacement p7
```

```
new(p8) T(t);     // construit un élément par copie de
                  // l'élément t à l'emplacement mémoire
                  // spécifié par le pointer p8.

                  // Cet emplacement doit
                  //   1. avoir été alloué
                  //   2. ne pas avoir été construit

p8->~T();         // détruit l'élément pointé par p8, sans
                  // libérer la mémoire à l'emplacement p8
```

```
*p9 = t;          // affecte une copie de t à l'élément
                  // pointé par p9 avec l'opérateur
                  // d'affectation par copie.

                  // Un élément doit avoir préalablement
                  // été construit à cette adresse pour
                  // que l'on puis y faire une affectation

// il n’y a rien à détruire ou libérer en plus suite à
// cette ligne de code
```