Allocation et Libération d'espace

- Allocation d'espace nécessaire pour toute structure qui ne respecte pas les règles des variables locales des fonctions

- Et donc, libération aussi nécessaire; sinon une longue exécution du programme risque d'échouer, faute d'espace mémoire disponible

- Mais libérer trop ou trop tôt peut aussi être désastreux

- Gestion de mémoire peut être automatique ou semi-automatique ou artisanale selon le langage de programmation et l'application

Méthodes de libération automatiques

Deux types de méthodes ont été étudiés qui permettent de récupérer automatiquement l'espace qui ne sera plus utilisé par le programme
un programme peut aussi utiliser les mêmes méthodes de "ramassage de miettes"

Première méthode
Compter les pointeurs

- dans chaque bloc (struct etc.) maintenir un champ caché qui est le nombre de pointeurs au bloc

- mise à jour chaque fois un pointeur est ajouté ou enlevé

- quand le nombre tombe à 0, le bloc peut être libéré avec éventuellement d'autres libérations en réaction en chaîne

- ne réussit pas à libérer toutes les structures circulaires
  (le cas simple d'un pointeur d'un bloc vers lui-même peut être traité)

Libération automatique
marquer et balayer

- Les structures accessibles à un moment donné sont celles directement connues au programme (valeurs de variables et paramètres) plus celles accessibles indirectement de celles-ci par des chaînes de pointeurs

- maintenir un bit dans chaque bloc : est-il accessible ou non?

- de temps en temps remettre tous à 0

- procéder récursivement à partir de chaque bloc directement accessible mettant des bits à 1 en chaque bloc rencontré (et arrêtant quand un bloc trouvé est déjà marqué)

- enfin balayer toute la zone en récupérant tous les blocs non marqués

- éventuellement compactifiant tous les blocs survivants
  (mais ceci nécessite modification de tous les pointeurs!)

- problème : le programme doit s'arrêter pendant ce temps

Ramassage parallèle

Une famille d'algorithmes basés sur un premier de Dijkstra permettent de ramasser sans arrêter les autres processus

- algorithmes compliqués :
  plusieurs "démonstrations" qu'ils sont corrects, dont plusieurs fausses!

- distinguent trois types de bloc : non marqués (blancs), marqués mais ses fils ne sont pas forcément marqués (gris), marqués et ses fils aussi (noirs)

- quand un pointeur dans un bloc change, le bloc et le nouveau fils deviennent gris

- Méthode utilisée en java

Contraintes sur le langage de programmation pour qu'un système automatique soit possible

- Il faut que le système puisse savoir où sont les pointeurs dans chaque bloc
- Règles strictes de typage
- Dangers d'arithmétique sur les pointeurs
- Restrictions sur les types **union**

Système semi-automatique

Par exemple en C ou exalgo

- libération explicite par le programme (free() ou désallouer())
- parcours de listes ou d'arbres possibles pour libérer chaque noeud
- mais difficile avec structures circulaires
- danger subtile de parcours simple qui marque et libère!
- aussi structures partagées

Allocation “artisanale” avec blocs de la même taille

- Si un programme n’utilise que des blocs d’une taille (manipule un type de liste ou d’arbre, ...), il est facile de garder tous les blocs disponibles dans une liste

- allocation : prendre le premier bloc de la liste

- s’il y en a !

- libération : ajouter au début de la liste

Blocs de tailles différentes

- Plus compliqué : toujours une liste de blocs disponibles mais de tailles différentes et en ordre d'adresse

- allocation : prendre le premier (?) bloc suffisamment grand dans la liste

- enlever de la liste ou ajuster la taille

- libération : trouver la bonne position dans la liste

- insérer et éventuellement fusionner avec son prédécesseur et/ou successeur

- compactification possible mais complexe