La Pile et le Tas

Organisation de la mémoire

Au sein d'une application, la mémoire disponible est découpée en deux grandes zones :

- La pile (ou *stack* en anglais)
- Le tas (*heap*)

La pile est une zone mémoire gérée automatiquement par le compilateur. C'est la zone par défaut d'allocation des variables. Celle ci est souvent limitée en taille (quelques méga-octets sur les systèmes modernes). Elle est aussi utilisée en particulier pour empiler les arguments d'une fonction lors de son appel(d'où son nom).

A l'inverse, l'utilisation du tas doit être faite explicitement par le programmeur par l'utilisation de l'allocation dynamique. Cela peut être fait manuellement avec des pointeurs nus ou par des objets qui le feront pour nous. Cette zone est virtuellement illimitée : tant qu'il y a de la mémoire de libre, on peut créer des objets dedans.

ces deux zones mémoires sont le fruit d'une séparation logique de la mémoire du processus. Une mauvaise utilisation des pointeurs et de l'arithmétique associée peut amener à modifier le tas par accident. Ce genre d'erreur est difficile à traiter. Pour limiter ces risque, il est conseillé d'utiliser le moins possible des pointeurs nus.

Durée de vie et portée

Au moment de la création d'une variable, le compilateur va automatiquement réserver de la mémoire dans la pile pour cette dernière. La pile n'ayant pas une taille infinie, il est important qu'il puisse libérer

cette mémoire le plus tôt possible. Il peut le faire à partir du moment où le flux d'exécution sort du bloc où la variable a été déclarée. La destruction des variables allouées sur la pile se fera dans l'ordre inverse de leur déclaration.

Dans l'exemple ci dessous, le flux d'exécution commence à *main*. La première accolade ouvrante déclare un nouveau bloc qui se termine à l'accolade fermante correspondante 5 lignes plus bas. La première instruction de ce bloc est la création d'une variable *a*. Le flux d'exécution rentre alors dans un nouveau bloc, qui se termine 3 lignes plus bas. Dans ce bloc, on va créer les variables *b* et *c* avant d'en sortir. Au moment de la sortie du bloc, *c* puis *b*. Le flux rencontre ensuite la fin du premier bloc, où il va procéder alors à la destruction de *a*.

```
int main() {
   int a { 132 };
   {
       int b { 456 };
       int c = b + a;
   } // destruction de c puis b
} //destruction de a
```

La Pile par défaut

Utiliser le Tas

Allocation dynamique avec unique_ptr et make_unique

Allocation dynamique avec shared_ptr et

shared_unique

explication de la différence ? Intéret pratique ? Si on n'a pas encore vu les fonctions

bloc {}

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++