AP1 TP

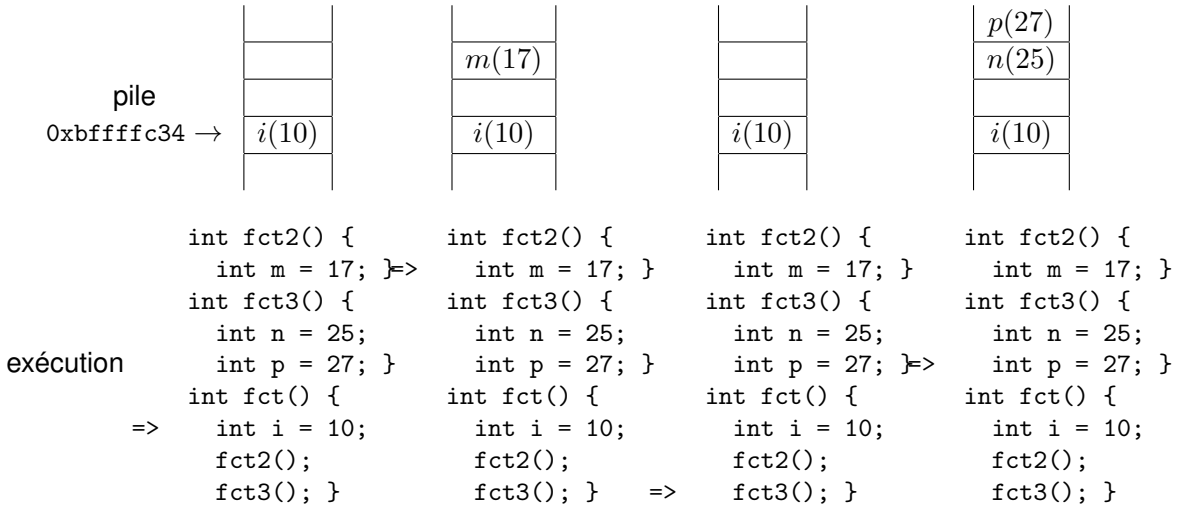
15 Semaine 15 : Pointeurs 2/2

Vous commencerez par récupérer les fichiers sources placés dans `/net/Bibliothque/AP1/TPSem15`.

Exercice 1 : Allocation statique : la pile

Regardez, compilez et exécutez le fichier `exAlloc.cc`. Vous constatez que les variables instanciées dans les fonctions ne sont pas allouées de manière complètement aléatoires, mais se suivent de façon décroissante. Ces variables sont allouées sur la *pile* d'exécution du processus. Lorsque le processeur rentre dans une fonction, de la mémoire est allouée sur la pile pour les variables locales à la fonction. Lorsque le processeur quitte une fonction, l'adresse de la pile remonte : les variables définies dans la fonction sont donc perdues. Les valeurs seront écrasées lors de l'appel d'une nouvelle fonction.

La figure ci-dessous illustre ce mécanisme d'allocation sur un exemple simple, où une fonction `fct` appelle successivement deux autres fonctions `fct2` et `fct3`. La flèche indique l'endroit où en est l'exécution. L'état de la pile est affiché au-dessus.



En fait, il est possible pour le compilateur de déterminer à l'avance, pendant la phase de compilation, à quelles positions respectives les variables vont être allouées sur la pile. C'est pourquoi on parle d'*allocation statique*. Un gros avantage de cette allocation est que la mémoire est désallouée naturellement et automatiquement lorsque vous n'en avez plus besoin. Un désavantage est que vous devez connaître à l'avance la taille des variables que vous voulez allouer sur la pile. C'est pourquoi vous devez fixer la taille d'un tableau par exemple.

Exercice 2 : Les paramètres sont aussi alloués sur la pile

Regardez, compilez et exécutez le fichier `exAllocbis.cc`. Vous pouvez constater que les paramètres passés par valeur sont des variables comme les autres, allouées sur la pile. Les paramètres passés par référence n'ont pas d'espace propre alloué.

Exercice 3 : Allocation dynamique, tas *(Opérateurs `new`, `delete`.)*

Dans cet exercice, vous allez allouer des données dynamiquement en utilisant les opérateurs `new` et `delete`.

1. Écrivez une fonction qui renvoie un réel aléatoire compris entre 0.0 et 1.0. Vous ferez appel à la fonction `rand()`. Avez-vous besoin de faire une allocation dynamique pour ce réel ?
2. Écrivez une fonction qui renvoie un pointeur sur un réel aléatoire compris entre 0.0 et 1.0. Avez-vous besoin de faire une allocation dynamique pour ce réel ?

3. Affichez le contenu de la variable pointeur renvoyée par la fonction précedente. Essayez ensuite de l'afficher après avoir désalloué la mémoire. Que se passe-t-il ?

Exercice 4 : Allocation dynamique de tableaux

(Opérateurs `new[]`, `delete[]`*.)*

On peut aussi allouer dynamiquement des tableaux de variables ou d'objets. L'opérateur est alors l'opérateur `new[]`. La taille voulue du tableau est passée en paramètre. Pour les désallouer, on utilise l'opérateur `delete[]`. Attention, le type renvoyé est simplement un pointeur sur le premier élément.

1. Écrivez une fonction qui renvoie un tableau de réels aléatoires compris entre 0.0 et 1.0, dont la taille sera donnée par l'utilisateur.
2. Écrivez une action qui affiche le contenu du tableau. Essayez d'afficher le contenu du tableau après avoir désalloué la mémoire. Que se passe-t-il ?

Exercice 5 : Pointeur de pointeurs

Ecrire une fonction à laquelle on passe un tableau de n pointeurs sur des `float` et qui retourne un nouveau tableau contenant ces n valeurs `float`.

Exercice 6 : Compressons

Ecrire une fonction qui reçoit comme paramètres un tableau d'entiers et sa taille, et retourne un tableau de pointeurs sur des entiers. Un élément du tableau de pointeurs pointera vers un entier égal à l'élément correspondant dans le tableau d'entiers. Si deux éléments du tableau d'entiers contiennent la même valeur, les deux pointeurs correspondants du tableau de pointeurs pointeront vers la même variable dynamique.

Exercice 7 : Tableau d'entiers non triés dynamique

L'idée est de manipuler un tableau dont la taille est égale au nombre exact d'éléments contenus dans le tableau. A chaque question, vous écrirez un programme de test.

1. Ecrire la fonction `saisir` qui demande à l'utilisateur un entier n, saisit n valeurs et renvoie ou transmet en sortie un pointeur sur le tableau des n éléments saisis.
2. écrire la fonction `afficher` qui étant donnés un pointeur p et une taille n affiche le tableau de taille n pointé par p.
3. écrire de deux façons différentes la fonction `enfiler` d'ajout d'un élément en fin du tableau.
4. écrire de deux façons différentes la fonction `defiler` de suppression du premier élément avec un décalage gauche.
5. écrire la fonction `premierFile` qui renvoie la valeur du premier élément du tableau.
6. écrire la fonction `fileVide` qui renvoie `true` si le tableau est vide et `false` sinon.
7. Uniquement à l'aide des fonctions `enfiler, defiler, premierFile, fileVide`, déterminer l'élément maximum du tableau.
8. Que pensez-vous de la complexité de l'implémentation proposée ? Dans la question précédente, combien de fois y a-t-il allocation et/ou désallocation d'un tableau ? Proposer des solutions afin d'améliorer la complexité. Ré-implémeter les primitives.
9. Toujours à l'aide des fonctions `enfiler, defiler, premierFile, fileVide`, écrire une fonction qui étant donné un tableau d'entiers de taille n sépare les entiers positifs des entiers négatifs en deux tableaux.
10. Toujours à l'aide des fonctions `enfiler, defiler, premierFile, fileVide`, écrire une fonction qui supprime les doublons dans un tableau.