CPPLI : TD 4 : C : Allocation dynamique et structure

Romain Absil Jonas Beleho Pierre Bettens
David Hauweele Pierre Hauweele
Nicolas Vansteenkiste * (Esi – He2b)

Année académique 2021 – 2022

Ce TD [1] aborde l'allocation dynamique de mémoire et les structures en langage C.

1 Allocation dynamique de la mémoire

Ex. 4.1 Écrivez la fonction de prototype :

```
unsigned * primeFactorsA(unsigned * count, unsigned number);
```

Elle crée et retourne une zone dynamiquement allouée [2] d'**unsigned** dont le contenu, un tableau [3] d'**unsigned**, est l'ensemble des facteurs premiers [4] du paramètre `number`.

Le nombre d'**unsigned** formant la zone retournée est stocké dans l'entier dont l'adresse est fournie via le pointeur `count`.

---

*Et aussi, lors des années passées : Monica Bastreghi, Stéphan Monbaliu, Anne Rousseau et Moussa Wahid.

1. https://poesi.esi-bru.be/pluginfile.php/7311/mod_folder/content/0/td04_c/td04_c.pdf (consulté le 13 septembre 2021).
2. https://openclassrooms.com/courses/apprenez-a-programmer-en-c/l-allocation-dynamique (consulté le 13 septembre 2021).
3. https://en.wikipedia.org/wiki/C_(programming_language)#Arrays (consulté le 13 septembre 2021).
4. https://en.wikipedia.org/wiki/Prime_factor (consulté le 13 septembre 2021).

Les facteurs premiers sont rangés dans l'ordre croissant, c'est-à-dire que le plus petit est à l'indice 0 du tableau retourné. Si un nombre apparaît plusieurs fois dans la décomposition en facteurs premiers de `number`, il apparaît autant de fois dans le tableau retourné.

Voici un extrait de code invoquant `primeFactorsA` :

```
1 unsigned nbElem = 0;
2 unsigned * decomposition = primeFactorsA(&nbElem, 84);
```

Après cet appel :

— `nbElem` contient la valeur 4 ;

— le contenu du tableau d'`unsigned` dont l'adresse du premier élément est dans `decomposition` est : $\{2, 2, 3, 7\}$.

Testez votre fonction en affichant le contenu des tableaux retournés pour des `number` compris entre 0 et 100 et vérifiant que le produit des éléments de chaque tableau est bien égal à `number`. N'oubliez pas de libérer les ressources [5] allouées à chaque appel de `primeFactorsA` !

Ex. 4.2 Écrivez la fonction de prototype :

```
unsigned primeFactorsB(unsigned * * factor,
                       unsigned * * multiplicity,
                       unsigned number);
```

Elle décompose `number` en produit de facteurs premiers. Plus précisément, elle stocke les facteurs premiers de `number`, rangés dans l'ordre croissant, dans une première zone qu'elle alloue dynamiquement, tandis que leurs multiplicités sont rangées dans une seconde zone qu'elle alloue également dynamiquement. Ces deux zones dynamiques sont des tableaux dynamiques d'`unsigned`. Les adresses de ces deux tableaux sont écrites par la fonction `primeFactorsB` dans les deux pointeurs d'`unsigned` dont les adresses sont fournies via les paramètres `factor` et `multiplicity`. La valeur retournée est le nombre d'éléments de chacun de ces tableaux.

Voici un extrait de code invoquant `primeFactorsB` :

```
1 unsigned * facteurs = NULL;
2 unsigned * multiplicites = NULL;
3 unsigned nbElem = primeFactorsB(&facteurs, &multiplicites, 84);
```

Après cet appel :

— `nbElem` contient la valeur 3 ;

— le contenu du tableau d'`unsigned` dont le premier élément est pointé par `facteurs` est : $\{2, 3, 7\}$;

5. https://en.cppreference.com/w/c/memory/free (consulté le 13 septembre 2021).

— le contenu du tableau d'`unsigned` dont l'adresse du premier élément est stockée dans le pointeur `multiplicites` est : $\{2, 1, 1\}$.

Testez votre fonction en affichant les contenus des tableaux dynamiques créés pour des `number` compris entre 0 et 100 et vérifiant que le produit des facteurs premiers obtenus est bien égal à `number`. N'oubliez pas de détruire les deux tableaux créés à chaque appel de `primeFactorsB` !

2 Structure

Ex. 4.3 Écrivez la fonction de prototype :

```
struct PrimeFactor * primeFactorsC(unsigned * count,
                                   unsigned number);
```

La structure [6] `struct PrimeFactor` est définie comme :

```
struct PrimeFactor
{
    unsigned value;
    unsigned multiplicity;
};
```

La fonction `primeFactorsC` alloue une zone mémoire dynamique où elle range de manière contiguë des `struct PrimeFactor`. Elle retourne l'adresse du premier élément de ce tableau dynamique de `struct PrimeFactor`. Le contenu du tableau dynamique est l'ensemble des facteurs premiers du paramètre `number`. À chaque facteur premier correspond un élément du tableau : son champ `value` renferme sa valeur, son champ `multiplicity` sa multiplicité. Le nombre d'éléments de ce tableau, c'est-à-dire le nombre de facteurs premiers distincts, est stocké dans l'entier non signé dont l'adresse est fournie à la fonction via le paramètre `count`.

Voici un extrait de code invoquant `primeFactorsC` :

```
1 unsigned nbElem = 0;
2 struct PrimeFactor * decomposition = primeFactorsC(&nbElem, 84);
```

Après cet appel :

— `nbElem` contient la valeur 3 ;

— le contenu du tableau de `struct PrimeFactor` dont le premier élément est pointé par `decomposition` est : $\{\{2, 2\}, \{3, 1\}, \{7, 1\}\}$.

6. https://en.wikipedia.org/wiki/Struct_(C_programming_language) (consulté le 13 septembre 2021).

Testez votre fonction en affichant le contenu des tableaux retournés pour des `number` compris entre 0 et 100 et vérifiant que le produit des facteurs premiers obtenus est bien égal à `number`. N'oubliez pas de libérer les ressources allouées à chaque appel de `primeFactorsC` !

Ex. 4.4 Écrivez la fonction de prototype :

```
void primeFactorsD(struct PrimeFactorization * pf);
```

La structure `struct PrimeFactorization` est définie comme :

```
struct PrimeFactorization
{
    unsigned number;
    unsigned count;
    struct PrimeFactor * primeFactors;
};
```

où la structure `struct PrimeFactor` est définie dans l'Ex. 4.3.

La fonction `primeFactorsD` décompose en facteurs premiers le champ `number` de la structure pointée par son paramètre `pf`. Elle stocke dans son champ `count` le nombre de facteurs premiers distincts. Elle alloue un tableau dynamique de `struct PrimeFactor` dont elle stocke l'adresse du premier élément dans le champ `primeFactors` de la `struct PrimeFactorization` pointée par `pf`. Le contenu de ce tableau dynamique est l'ensemble des facteurs premiers distincts du nombre `pf->number`, similairement à l'Ex. 4.3.

Voici un extrait de code invoquant `primeFactorsD` :

```
1 struct PrimeFactorization f = { 84, 0, NULL };
2 primeFactorsD(&f);
```

Après cet appel, le contenu de `f` est : $\{84, 3, \{\{2, 2\}, \{3, 1\}, \{7, 1\}\}\}$.

Testez votre fonction en affichant le contenu des tableaux retournés pour des `number` compris entre 0 et 100 et vérifiant que le produit des facteurs premiers obtenus est bien égal à `number`. N'oubliez pas de libérer les ressources allouées à chaque appel de `primeFactorsD` !