Langages 1
Recueil d'exercices

Nikita Veshchikov, Keno Merckx, Cédric Ternon
Une partie d'exercices est basé sur les versions fait par
Stephane FERNANDES MEDEIROS
remerciements à
Jérôme DOSSOGNE
Naïm QACHRI

Table des matières

Chapitre 1

C++

1.1 Compilation et exécution de programmes en C++

Exercice 1 Écrivez un programme qui affiche *"Hello !"*. Nommer le fichier source `hello.cpp`

Exercice 2 En ligne de commande lancer le compilateur `g++` avec l'option `--version` pour vérifier la version du compilateur. Le résultat peut ressembler à ça :

```
g++ (Ubuntu 4.8.2-19ubuntu1) 4.8.2
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Exercice 3 Pour compiler un fichier `.cpp` avec `g++` on peut utiliser la commande suivante :

```
g++ <filename.cpp> -o <output_executable_program>
```

Compilez le programme `hello.cpp` à l'aide du compilateur `g++`, appelez votre programme `hello`. Si le compilateur n'affiche pas de message d'erreur cela veut dire que tout s'est bien passé.

Lancez le programme à l'aide de la commande `./hello` en terminal.

Exercice 4 Modifiez votre programme `hello.cpp` pour qu'il affiche *"Hello, world."*, sauvegardez le fichier, retournez dans le terminal et relancez `hello`. Qu-est-ce que votre programme vient d'afficher ? Pourquoi ? Faites le necessaire pour qu'en lancant `./hello` en terminal on puisse voir le message *"Hello, world."*.

Exercice 5 Suprimez le fichier executable `hello`. Retournez dans le fichier source `hello.cpp` et supprimez le symbole `;` à la fin d'une de ligne. Essayez de recompiler le programme. Que affiche le compilateur ? Est-ce qu'on peut lancer le programme en terminal ? Corrigez l'erreur et recompiler le programme.

Exercice 6 Le compilateur `g++` permet d'activer une serie d'option. Recompilez le programme `hello.cpp` avec les options `-Wall` et `-Wextra`. Verifiez que votre programme s'execute correctement.

Supprimez le fichier `hello`.

Retournez dans le code du `hello.cpp` et declarez une variable `int x` au debut du `main`. Recompilez le programme `hello.cpp` avec les options `-Wall` et `-Wextra` encore une fois. Que veut dire le message affiché par le compilateur ? Est-ce que le fichier executable est créé ?

Exercice 7 Ecrivez un programme `sum.cpp` qui demande à l'utilisateur 2 entieres et affiche leur somme.

Exercice 8 Telechargez le fichier `error.cpp` sur UV, ce fichier contient des erreurs. Compilez ce programme à l'aide de la commande suivante : `g++ -Wall -Wextra error.cpp -o no_error`

Corrigez les erreurs dans le code source pour que le programme puisse être compilé sans warnings.

Executez ce programme avec des differentes valeurs d'entrée, e.g. 8 et 4, 16 et 3, 5 et 0. Expliquez les resultats.

Exercice 9 Pour ne pas devoir re-ecrire la ligne de compilation d'un programme et pour simplifier le processus de compilation des grandes programmes on utilise des Makefiles. Supprimez le fichier `hello`.Creez un fichier vide et appelez le `Makefile`.

Voici la structure d'un `Makefile` tres simple :

```
# commentaire
all: <nom du programme1> <nom du programme2>

<nom du programme1>: <fichier(s) necessaire(s) pour la compilation>
<TAB> <la ligne de compilation>

<nom du programme2> : ...
...
```

Creez le `Makefile` pour `hello.cpp` et compilez votre programme. Pour lancer un `Makefile` tapez `make` en ligne de commande. Executez `hello` pour verifier qu'il fonctionne. Relanez le `Makefile` encore une fois. Qu'est ce qui est affiché à l'ecran ?

Modifiez le programme `hello.cpp` pour qu'il affiche *"Hello, world !"*. Sauvegardez le fichier et retapez `make` en ligne de commande. que s'est il passé ?

Exercice 10 Modifier le `Makefile` pour qu'on puisse l'utiliser pour compiler `hello.cpp` et `sum.cpp`. Relancez le `Makefile`. Essayez de lancer le `Makefile` en modifiant un de deux programmes, en modifiant les deux en même temps.

1.2 Les conditions et les boucles

Exercice 11 Ecrire une fonction qui reçoit trois entiers en paramètre et les affiche de manière triée. Vous ne pouvez utiliser que les structures de contrôle if, else if et else. Appelez cette fonction à partir d'un main où vous demandez les trois nombres à l'utilisateur.

Exercice 12 Voici un algorithme que vous avez vu au premier trimestre en programmation, il est exprimé en Python, exprimez-le en C++.

```
n = int(input('entier positif : '))
```

```
while n != 1:
    print(n, end=' ')
    if n % 2 == 0:
       n = n//2
    else:
       n = n*3+1
```

Exercice 13 Voici un autre algorithme vu au cours de programmation, exprimez-le également en C++.

```
import math
EPSILON =10**-20

def almost_equal(x, y, EPS = 10 ** -7):
   return abs(y - x) < EPS

def square_root(a):
   """Calcul de la valeur approchée de la racine carrée de a par la méthode de Héron
   """
   x = 0.0
   y = 1.0
   while not almost_equal(x, y,EPSILON):
      # rq : pas almost_equal(a,y*y,epsilon) qui peut cycler
      x=y
      y = (y + a / y) / 2
   return y
```

Exercice 14 Toujours au cours de programmation, vous avez vu que l'approximation de e réalisée par :

$$1+\frac{1}{1!}+\frac{1}{2!}+\ldots+\frac{1}{n!}$$

diffère de e d'au plus deux fois le terme suivant dans la série, c'est à dire d'au plus $2/(n+1)!$

Réalisez en C++ une fonction recevant en paramètre un epsilon caractérisant la différence maximale entre l'approximation renvoyée par votre fonction et e.

1.3 Les ADT

Exercice 15 Lors du cours théorique de langages, vous avez observé deux implémentations d'une classe représentant les nombres complexes. L'une est implémentée sous forme cartésienne, l'autre sous forme polaire. Enrichissez chacune des classes avec les opérateurs de soustraction, division, multiplication.

Exercice 16 Ecrire une classe Personne permettant de décrire complètement une personne, sachant que l'on souhaite avoir autant d'informations que dans la phrase suivante :
"M. George Abitbol est né en 1993, il est célibataire."

Exercice 17 On voudrait gérer les étudiants d'une institution à l'aide d'une classe Etudiant définie par les attributs suivants :

— `nom` : nom d'un étudiant (max 50 char)
— `prénom` : prénom d'un étudiant (max 50 char)
— `tabnotes` : tableau contenant les notes d'un étudiant, sachant qu'un étudiant a au total 10 notes.

les méthodes suivantes :

— `void saisie ()`, permettant la saisie d'un étudiant
— `void affichage ()`, permettant l'affichage d'un étudiant
— `float moyenne ()`, retourne comme résultat la moyenne des notes d'un étudiant.
— `bool exae_quo (Etudiant E)`, retourne comme résultat la valeur `true`, si deux étudiants ont la même moyenne et la valeur `false`, sinon.

Exercice 18

1. Pour représenter un point entier dans le plan euclidien, construire une classe Point sachant qu'un point est défini par ses 2 coordonnées entières x1,y1. Ecrire le constructeur de cette classe et surcharger l'opérateur "+". On prendra 0 comme valeur par défaut pour x1 et y1.
2. Construire une classe Rectangle sachant qu'un rectangle entier est défini par 4 valeurs entières x1,y1,x2,y2 représentant les coordonnées des sommets d'une diagonale du rectangle. Ecrire le constructeur de cette classe. On prendra 0 comme valeur par défaut pour x1,y1,x2 et y2
3. Ajouter une méthode Ordonner qui permet d'ordonner les coordonnées x1,y1,x2,y2 de manière à ce que le point (x1,y1) corresponde au coin en haut à gauche du rectangle et le point (x2,y2) au coin en bas à droite.
4. Modifier le constructeur de manière à avoir un rectangle ordonné dès la construction de l'objet.
5. Ajouter un second constructeur qui permet de construire l'objet rectangle à partir de 2 objets de la classe Point.
6. Ajouter une methode Translation à la classe Rectangle qui effectue une translation du rectangle selon un vecteur OP défini par l'origine et un point P.

1.4 Les tableaux et les pointeurs

Exercice 19 Écrivez un programme qui demande à l'utilisateur d'entrer un entier et affiche la valeur entrée ainsi que l'adresse où cette valeur est stockée. Stockez la valeur de l'adresse dans une variable (vous allez avoir besoin d'un pointeur). Exécutez votre programme plusieurs fois. Est-ce que la même adresse est utilisée chaque fois ? Pourquoi ?

Exercice 20 Écrivez deux fonctions `swap` qui prennent deux paramètres et échangent leurs valeurs pour que la modification soit "visible" dans le main. Dans une des implémentations passez le pointeurs vers le variables comme paramètres, passez les valeurs par référence dans l'autre implémentation.

Testez les deux implémentations, écrivez un main avec seulement 2 variables, affichez leurs valeurs, appelez la fonction `swap` et affichez les valeurs de ces deux variables encore une fois pour vérifier que les contenus de ces deux valeurs sont échangées. Appelez l'autre fonction swap et affichez les deux variables encore une fois.

Exercice 21 Déclarez un tableau de 10 éléments. Remplissez ce tableau avec les nombres de Fibonacci. Affichez le contenu de ce tableau pour vérifier que les valeurs sont correctes. Modifiez la valeur 10 pour calculer les 20 premiers nombres de Fibonacci. Est-ce que vous avez du modifier plusieurs endroits dans votre programme ?

Exercice 22 Déclarez un vecteur de 5 éléments. Remplissez ce vecteur avec les nombres de 0 à 4. Affichez le contenu du vecteur, pour l'affichage utilisez les indices allant de 0 à 10000. Testez votre programme. Qu'est-ce qu'il affiche ? Pourquoi ?

Exercice 23 Déclarez un vecteur *text* de `char` de longueur 42. Demander à l'utilisateur d'entrer une phrase et affichez la en utilisant trois méthodes :

- utilisez simplement un seul `cout` (comme pour des entiers ou des reels)
- affichez chaque case du vecteur *text* séparément (`operateur []`),
- affichez le message sans passer par `operateur []`, utiliser le fait que `text` est un pointeur vers le début d'une suite de cases mémoire de type `char`.

Rappel : en `C++` la fin d'un string est signalée par le caractère `'\0'`.

Exercice 24 Écrivez une fonction qui prend en paramètre une matrice 4×4 et transpose cette matrice (en place, c-à-d vous ne pouvez pas déclarer encore une matrice). Testez votre fonction avec un petit programme, assignez les valeurs de cases de la matrice au moment de la déclaration.

Exercice 25 Écrivez un programme qui demande à l'utilisateur un nombre naturel n et alloue un vecteur (pour stocker des nombres réels) de cette taille ensuite. Remplissez ce vecteur avec les n premiers approximations du nombre d'or (utilisez les nombres de Fibonacci pour calculer les approximations successives). N'oubliez pas de désallouer l'espace avant de terminer le programme. Pour rappel, le nombre d'or peut être calculé en utilisant la formule suivante :

$$\varphi = \lim_{i \to \infty} \frac{F_{i+1}}{F_i}$$

Exercice 26 Dans l'exercice précédant, essayez de désallouer la mémoire une deuxième fois. Testez votre programme après cette modification.

Exercice 27 Écrivez un programme qui alloue des morceaux de mémoire sans les désallouer (en utilisant la même variable-pointeur) en boucle sans jamais s'arrêter. Lancer l'outil `System Monitor` pour pouvoir observer la quantité de la mémoire utilisée par les différents processus. Lancez ensuite votre programme pour observer ce qui se passe si on oublie de libérer (désallouer) la mémoire.

Exercice 28 Écrivez un programme qui contient une fonction `fctAdd(int a, int b)` qui fait l'addition des deux entier donnés a et b. Ceux-ci seront donnés par l'utilisateur. On vous demande ici de découper votre programme en trois fichier :

- `fctAdd.h` qui contiendra la déclaration de la fonction,
- `fctAdd.cpp` qui contiendra la définition de la fonction,
- `main.cpp`.

En guise de bonus : réalisez un Makefile.

1.5 Pointeurs, références, const et tableaux : aller plus loin

Séance créée par Jérôme Dossogne.

Rappels de différents types :

```
int i;                  // entier
const int i;            // entier constant
int const i;            // entier constant
const int *i;           // pointeur vers un entier constant
int *const i;           // pointeur constant vers un entier
const int *const i;     // pointeur constant vers un entier constant
int i[];                // tableau d'entiers
int *i[];               // tableau de pointeurs vers des entiers
int &i;                 // reference vers un entier
const int &i;           // reference vers un entier constant
```

Tableau de rappel des règles :

Expression	Assignation obligatoire à la déclaration ?	Réassignable ?	Assignable à un non-constant ?	Assignable à un constant ?
`int &i = j`	oui	oui	oui	non
`int *p`	non	oui	oui	non
`const int &i = j`	oui	non	oui (READ only)	oui
`const int *p`	non	oui	oui (READ only)	oui
`int *const i = &j`	oui	non	oui	non

Exercice 29 Parmi les instructions suivantes, lesquelles ne sont pas correctes ? Pourquoi ? Décrivez le comportement des instructions permises.

```
int main() {
        int i = 10;
        int *p;
        const int j = 20;
        i = j + 2;
        p = &i;
        j++;
        int &k;
        int &l = j;
        int &m = i;
        ++m;
        const int *q = &i;
```

```
        int n[] = { 1, 2, 12, 212, 12212, 21212212 };
        int *o[] = { &i, &k, &l};
        o[2] = 15;
        *&i = 28;
        *&*p = 30;
        *(n+3) = 14;
        int *const s;
        int *const r = &i;
        r = p;
        const int &z = j;
        return 0; }
```

Exercice 30 Déclarez :

1. un tableau t de pointeurs constants vers des entiers x, y et z
2. un pointeur vers un pointeur constant vers un entier pointant vers la troisième case de t
3. un pointeur vers une fonction qui renvoie un double et qui prend un pointeur vers un entier en paramètre

Exercice 31 Quelles sont les différences entre ces trois fonctions ? Sont-elles correctes ? Sinon pourquoi ?

```
void f(int i) { i = 10; }
void g(int &i) { i = 10; }
void h(const int &i) { i = 10; }
```

Exercice 32 Avec les mêmes fonctions que ci-dessus, parmi les appels suivants, lesquels ne sont pas valides ? Pourquoi ?

```
int main() {
        int a = 25;
        const int b = 38;
        f(3);
        g(3);
        h(3);
        f(a);
        g(a);
        h(a);
        f(b);
        g(b);
        h(b);
        return 0;
}
```

Exercice 33

Rappel :

```
Si un tableau est déclaré avec ints[],

&ints == &ints[0] == ints,

Par contre, si le tableau est déclaré *intp = new [] ou *intp = ints,

&intp != &intp[0]

Petit dessin :

ints
[ | | | | | | ]
^ |--- &ints == &int[0]

[ intp ]---------------->[ | | | | ]
^                        ^
|--- &intp               |--- &intp[0]
```

Question : Supposant que le vecteur ints se trouve à l'adresse 100, donnez la valeur des expressions suivantes :

```
int main()
{
        int ints[20] = { 10, 20, 30, 40, 50,60, 70, 80, 90, 100, 110, 120,
          130, 140, 150, 160, 170, 180, 190, 200 };
        int *intp = ints + 3;
        cout << endl;
        cout << ints << endl;
        cout << ints[4] << endl;
        cout << ints+4 << endl;
        cout << *ints+4 << endl;
        cout << *(ints+4) << endl;
        cout << ints[-2] << endl;
        cout << &ints << endl;
        cout << &ints[4] << endl;
        cout << &ints+4 << endl;
        cout << sizeof(ints) << endl;
        cout << (&ints)+4 << endl;
        cout << &ints[-2] << endl;
        cout << endl;
        cout << intp << endl;
        cout << intp[4] << endl;
        cout << intp+4 << endl;
```

```
        cout << *intp+4 << endl;
        cout << *(intp+4) << endl;
        cout << intp[-2] << endl;
        cout << &intp << endl;
        cout << &intp[4] << endl;
        cout << &intp+4 << endl;
        cout << &intp[-2] << endl;
        cout << endl;
        return 0;
}
```

Chapitre 2

ASM

2.1 Compilation et exécution de programmes en assembleur 80386

Exercice 34 Écrire le code assembleur qui place la valeur 14 dans le registre `EAX`, et qui ensuite appele la fonction qui affiche le contenu de `EAX`.

Exercice 35 Écrire le code assembleur qui remplace l'entier non signé se trouvant dans le registre `EAX` par son double.

Exercice 36 Écrire le code assembleur qui remplace l'entier non signé se trouvant dans le registre `EAX` par le résultat de l'opération `EAX` mod 32.

Exercice 37 Écrire le code assembleur qui remplace un entier non signé situé dans `EAX` par 1 si `EAX` est pair ou 0 si `EAX` est impair.

Exercice 38 Écrire le code assembleur qui inverse les 5 bits de poids fort et les 5 bits de poids faible d'un nombre codé sur 16 bits situé à l'adresse i.

2.2 Instructions arithmétiques

Exercice 39

Pour le reste des exercices de ce chapitre, nous allons supposer que :

- N1 est un entier signé sur 64 bits ;
- N2 est un entier signé sur 64 bits ;
- N3 est un entier signé sur 64 bits ;
- N4 est un entier non signé sur 128 bits ;
- N5 est un entier non signé sur 32 bits ;
- N6 est un entier non signé sur 64 bits.

On demande de donner le code assembleur permettant d'effectuer les opérations suivantes :

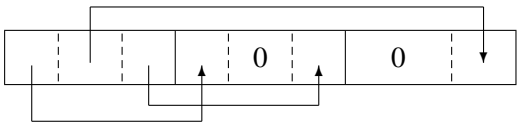
1. $\mathrm{N3} \leftarrow \mathrm{N1} + \mathrm{N2}$
2. $\mathrm{N3} \leftarrow \mathrm{N1} - \mathrm{N2}$
3. $\mathrm{N1} \leftarrow \mathrm{N1} * 2$
4. $\mathrm{N1} \leftarrow \mathrm{ROL}(\mathrm{N1}, 1)$

5. N3 $\leftarrow \lfloor \frac{N2}{2^{32}} \rfloor * 237 + N1$
6. N4 $\leftarrow N5 * N6$
7. N4 $\leftarrow N6^2$

Exercice 40 Soit un vecteur t de 3 composantes entières non signées sur 16 bits (déclaré par exemple en `C++` comme ceci : `short int t[3]`). Écrire le code assembleur qui place :

— les 5 bits de poids fort et les 5 bits de poids faible de $t[0]$ dans $t[1]$, le reste étant rempli par des 0 ;

— les bits 5 à 10 de $t[0]$ dans $t[2]$, tassés à droite.



2.3 Les choix et les boucles

Exercice 41

Étant donné un vecteur V de x entiers sur 16 bits, vérifier si ce vecteur ne contient que des éléments pairs. `EAX` sera mis à 1 si c'est le cas, et à 0 sinon. x est stocké à l'adresse n et fait 16 bits.

Exercice 42

Effectuer l'addition (tronquée à 16 bits) d'une matrice $A_{x\times y}$ avec une matrice $B_{x\times y}$, en plaçant le résultat dans une matrice $C_{x\times y}$. Les composantes de ces trois matrices sont des entiers sur 16 bits. x et y sont deux entiers sur 16 bits stockés respectivement aux adresses n et m.

Notons qu'il existe deux manières différentes de stocker une matrice en mémoire. La première consiste à parcourir la matrice colonne par colonne et à placer de manière consécutive en mémoire, par adresses croissantes, les éléments rencontrés. La deuxième diffère de la première dans le fait que ce parcours est fait ligne par ligne. Certains langages comme `FORTRAN`. `C++` le fait par lignes. Nous supposerons dorénavant, sauf si c'est précisé autrement, que les matrices à manipuler sont stockées par lignes.

Exercice 43

Calculer la trace (la somme des éléments de la diagonale) tronquée à 16 bits d'une matrice $A_{x\times x}$ d'entiers sur 16 bits, et placer le résultat dans le registre `EAX`. x est un entier sur 16 bits situé à l'adresse n.

Exercice 44

Soit x un entier non signé pair codé sur 16 bits, stocké à l'adresse n, et V un vecteur de x entiers non signés sur 16 bits. Chaque couple d'éléments $(V[2i], V[2i+1])$, $\forall i \in [0, x/2]$, contient une information (en $V[2i+1]$) et la position où devra se placer le couple (en $V[2i]$) après le traitement consistant à ordonner V. Ainsi, le vecteur :

3	10	5	15	4	17	1	9	2	19

deviendra, après traitement :

1	9	2	19	3	10	4	17	5	15

Exercice 45

Déterminer si une matrice $M_{x\times x}$ d'entiers sur 32 bits est symétrique. Le registre `EAX` devra valoir 1 si c'est le cas, 0 autrement. x est un entier sur 32 bits situé à l'adresse n.

2.4 Les procédures

Exercice 46

Écrire une fonction assembleur dont voici le prototype :

```
unsigned int moyenne(unsigned int, unsigned int);
```

qui renvoie la moyenne tronquée de 2 nombres entiers non signés de 32 bits passés en paramètres sur la pile.

Exercice 47

Soit V un vecteur de n entiers signés sur 32 bits. Écrire une fonction assembleur dont voici le prototype :

```
int somme(int[], unsigned int);
```

qui à 2 paramètres V et n passés sur la pile qui calcule

$$\texttt{EAX} = \sum_{i=1}^{n} V[i].$$

La gestion des dépassements de capacité peut être ignorée.

Exercice 48

Soient x et y deux entiers non signés codés sur 32 bits. Écrire une fonction assembleur

```
unsigned int exposant(unsigned int x, unsigned int y);
```

qui calcule x^y tronqué à 32 bits par l'algorithme suivant :

$v \leftarrow x$
$w \leftarrow y$
$res \leftarrow 1$
tant que $w > 0$ **faire**
    tant que w est pair **faire**
        $v \leftarrow v^2$
        $w \leftarrow w$ **div** 2
    ftant
    $w \leftarrow w - 1$
    $res \leftarrow res \times v$
ftant

Exercice 49

Écrire une fonction assembleur qui engendre et affiche toutes les permutations d'un vecteur V à n composantes sur 8 bits de façon récursive, par l'algorithme suivant :

permut(**caractère**[] v, **entier nonsigné** n)
  si $n = 0$ **alors**
    afficher(v)
  sinon
    pour i **allant de** 0 **jusqu'à** $n-1$ **faire**
      $v[i] \rightleftarrows v[n-1]$
      permut$(v, n-1)$
      $v[i] \leftrightarrows v[n-1]$
    fpour
  fsi
fin

2.5 Exercices supplémentaires

Exercice 50

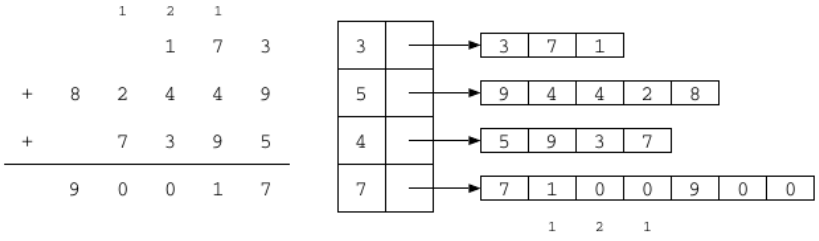
Écrire un programme effectuant le produit de deux matrices $A_{x\times y}$ et $B_{y\times z}$ d'entiers non signés sur 8 bits en plaçant le résultat dans $C_{x\times z}$. x, y et z sont trois entiers sur 16 bits stockés respectivement aux adresses m, n et p. Vous pouvez utiliser deux variables de travail de 16 bits situées aux adresses *ligne* et *colonne*.

Exercice 51

On peut modéliser l'addition écrite en base 10 de n termes à l'aide d'un vecteur à $n+1$ éléments. Ces derniers sont composés d'un pointeur vers un vecteur de chiffres décimaux sur 8 bits et d'un entier donnant le nombre d'éléments de ce dernier vecteur, tous deux codés sur 32 bits.

Les n termes à additionner sont encodés par ordre croissant de puissances de 10 dans les vecteurs vers lesquels pointent les n premières composantes du vecteur principal. Le résultat est placé dans le vecteur vers lequel pointe la $n+1^{\text{e}}$ composante du vecteur principal. On suppose que le $n+1^{\text{e}}$ nombre du vecteur principal est suffisamment grand pour pouvoir stocker le résultat et que les éléments du vecteur pointé sont initialisés à 0.

Voici un exemple, avec n égal à 3 :



On vous demande d'écrire un programme réalisant cette addition. On suppose la valeur de n et le

vecteur principal connus.

Exercice 52

Soit un A un tableau de n entiers signés sur 16 bits, strictement positifs. On vous demande d'écrire un programme qui compresse le tableau A en une séquence de bits constituée d'éléments composés de deux champs, le premier étant un champ sur 4 bits contenant la longueur du champ suivant, le champ de données. La fin de cette séquence sera marquée par un élément dont le champ longueur contient la valeur 0 (il n'y aura donc pas de données pour cet élément). La séquence obtenue devra être placée dans un tableau B, supposé suffisamment grand.

Les éléments de A resteront adjacents dans le tableau B. De plus, ils devront tous avoir la plus petite taille possible dans B, c'est-à-dire que les zéros inutiles présents en tête des éléments de A seront supprimés.

Par exemple, le tableau A suivant :

0000 0000 0000 1111	0000 0010 0101 1010	0000 0000 0000 0011	0111 0000 0000 0000

sera transformé en le tableau B suivant :

0100 1111	1010 10 0101 1010	0010 11	1111 111 0000 0000 0000	0000

Exercice 53

Soit M une matrice carrée $n \times n$ d'entiers sur 8 bits, où n est également codé sur 8 bits. Nous vous demandons d'écrire un algorithme qui réalise une rotation des éléments des lignes de cette matrice de la manière suivante : la ligne d'indice 0 restera inchangée, la ligne d'indice 1 subira une rotation vers la gauche d'une seule position, la ligne d'indice 2 subira une rotation vers la gauche d'amplitude 2, et ainsi de suite. Vous ne pouvez pas utiliser de structure de données de travail. De plus, vous pouvez supposer que M et n ont été correctement initialisées dans la section `.data` de votre programme.

Par exemple, dans le cas où n vaudrait 4, les éléments de la matrice se trouvant à gauche dans la figure suivante seraient placés, après transformation, tel qu'indiqué par la matrice de droite dans cette même figure.

$$\begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \xrightarrow{\text{est transformée en}} \begin{pmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ m_{1,1} & m_{1,2} & m_{1,3} & m_{1,0} \\ m_{2,2} & m_{2,3} & m_{2,0} & m_{2,1} \\ m_{3,3} & m_{3,0} & m_{3,1} & m_{3,2} \end{pmatrix}$$