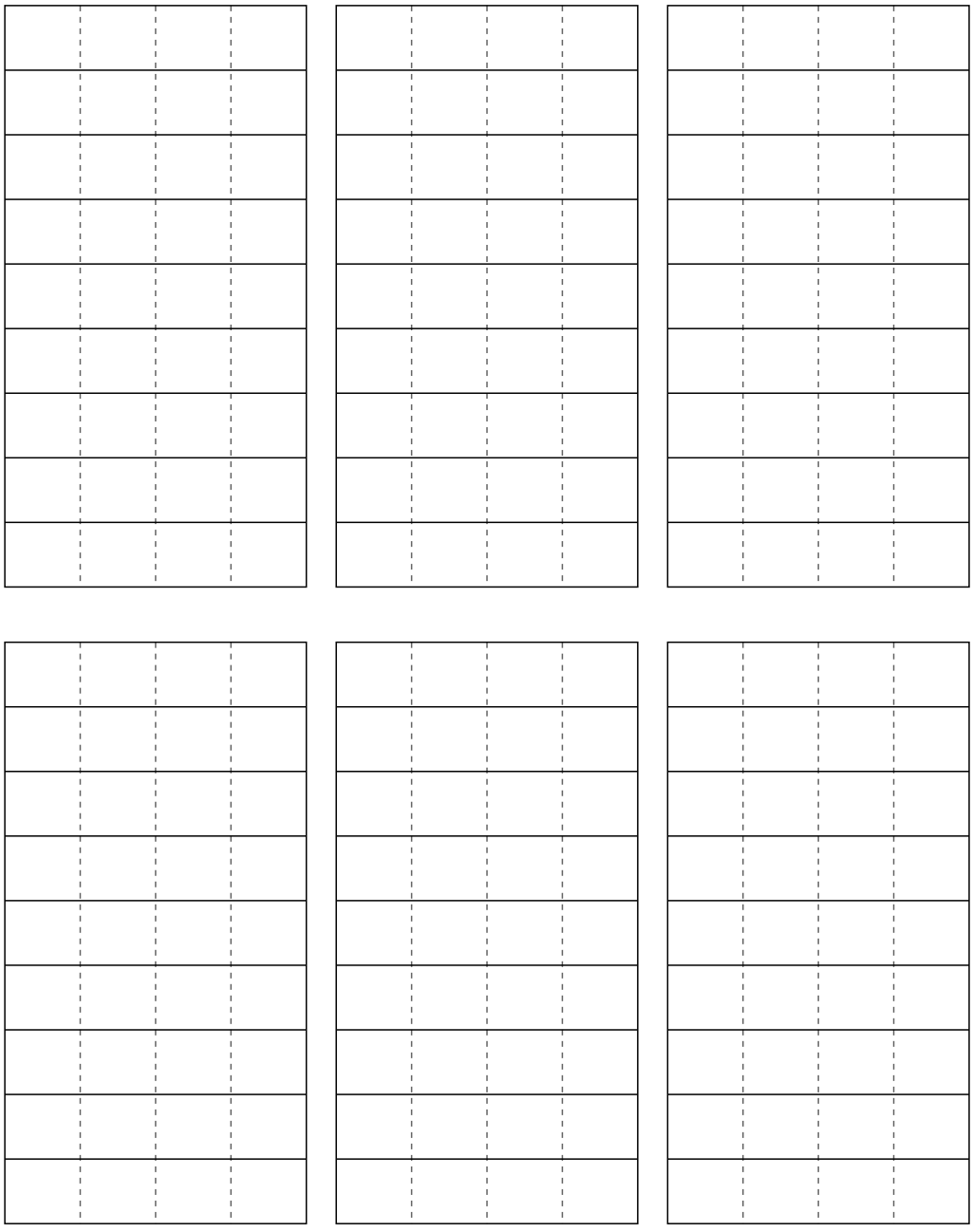
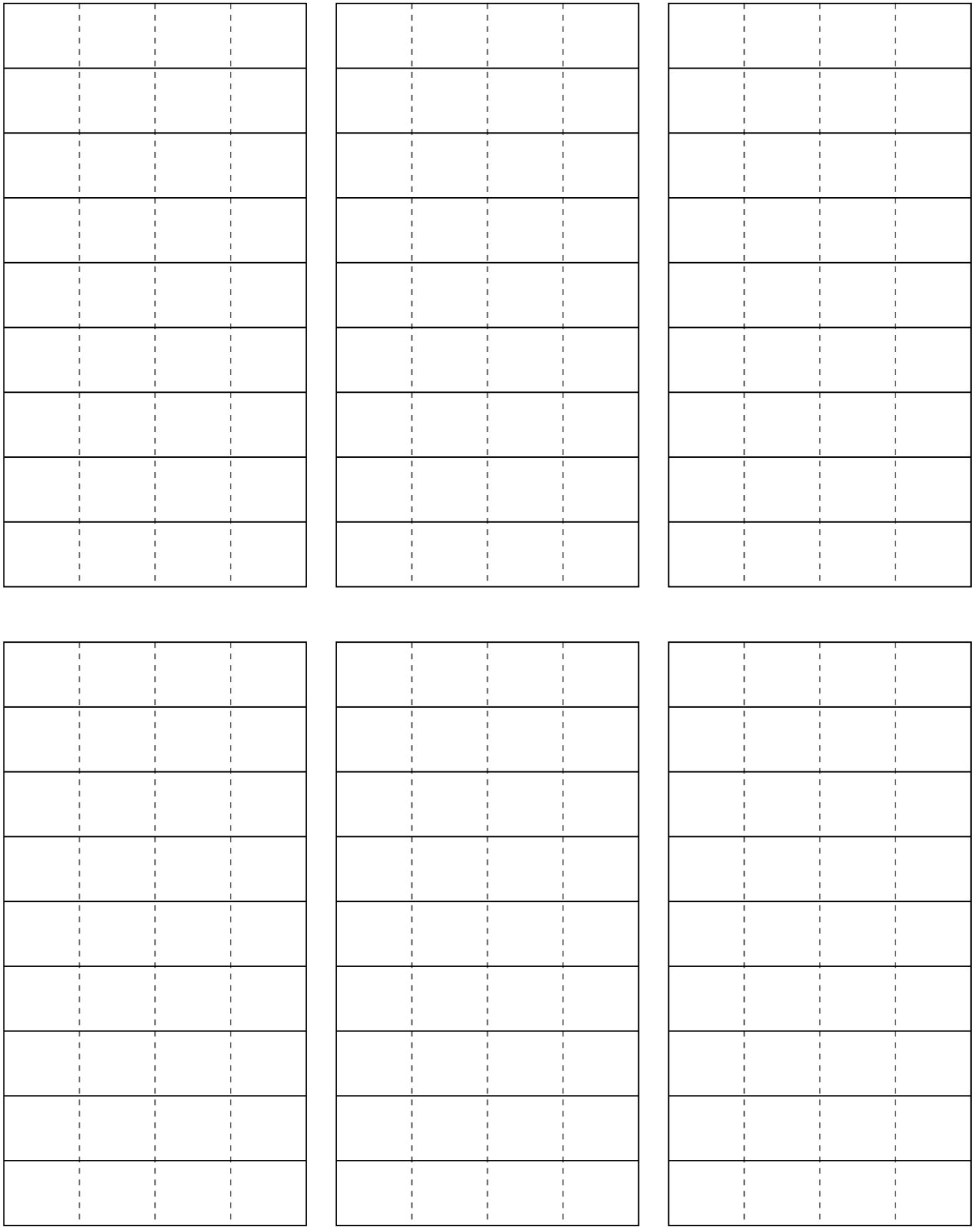
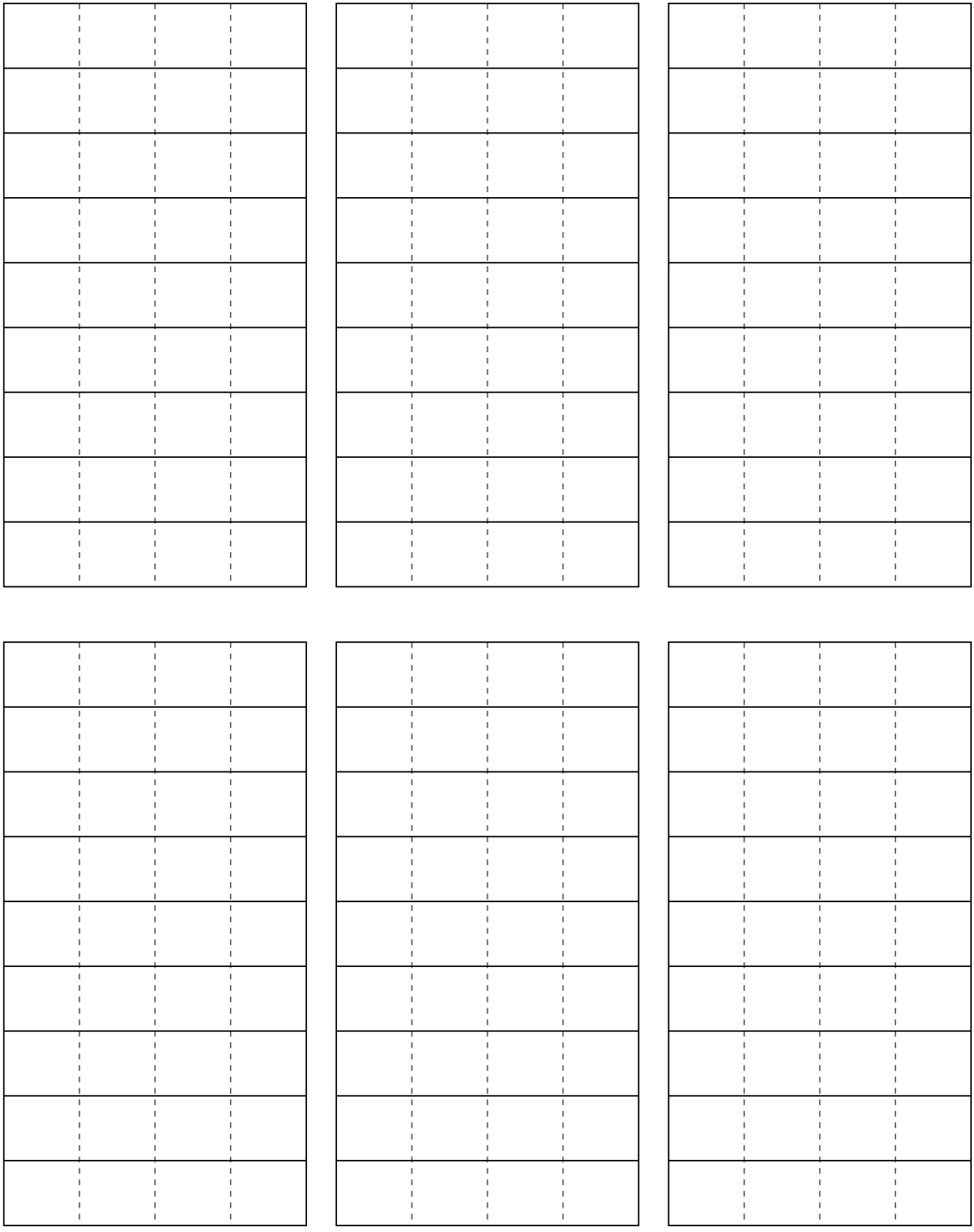
TD de simulation mémoire

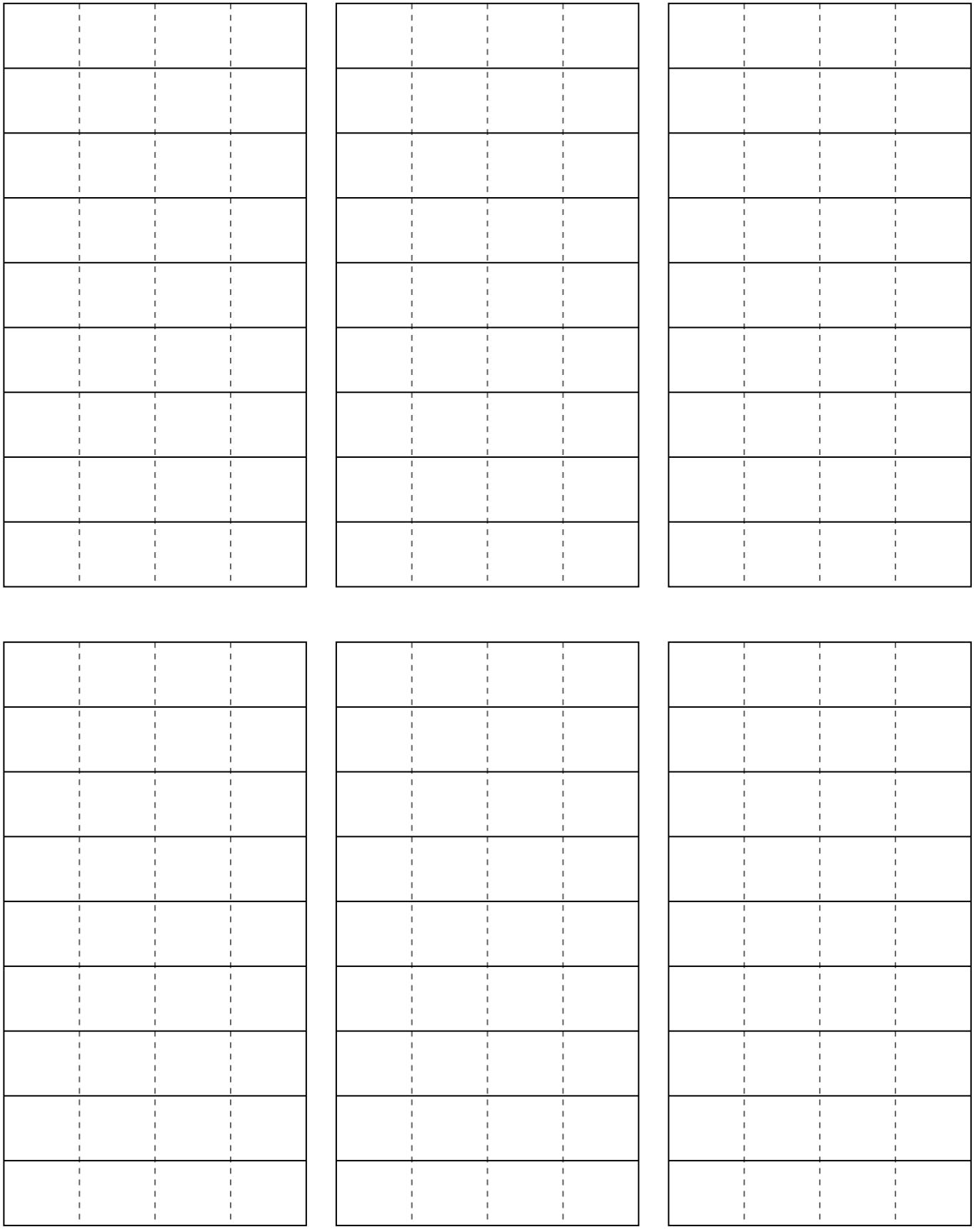
Dans l'ensemble des exercices suivants, on supposera:

- l'adresse de début de la pile à 200
- l'adresse de début du tas à 500
- l'adresse de début de la zone statique à 300









La pile: déclaration de variables (1)

0

Code à simuler :

```
void main(void)
{
    char c1 = 'A';
    char c2 = 'B';
    char c3 = 'C';
    char d = c3 - c1;
    c2 += d;
}
```

Notes:

- les variables dans la pile sont alignées sur 4 octets (l'adresse de chaque variable de pile est un multiple de 4).
- le code ASCII fixe la valeur correspondante au caractère 'A' à 65, 'B' à 66, 'C' à 67 ...

La pile: déclaration de variables (2)

1

Code à simuler :

```
void main(void)
{
    int i = 10;
    int j = i % 3;
    if (j == 0)
        puts("i est multiple de 3\n");
    else
        puts("i n'est pas multiple de 3\n");
}
```

Notes:

- ne simuler pas les appels de fonctions
- les chaînes de caractères seront supposées être avec les variables statiques

La pile: les pointeurs

2

Code à simuler :

```
void main(void)
{
    int i = 0;
    int *p = &i;
    *p = 10;
    i += 1;
}
```

Notes:

- pour toutes les variables (y compris les pointeurs), “&variable” fait référence à l’adresse de cette variable, “variable” fait référence au contenu de la variable.
- Pour les pointeurs, “*pointeur” fait référence au contenu de la case dont l’adresse est dans “pointeur”.

Questions:

- Quelle est la valeur de i en fin de programme?
- Quelle est la valeur de p en fin de programme?
- Quelle est la valeur de *p en fin de programme?

La pile: les tableaux (1)

3

Code à simuler :

```
void main(void)
{
    int i[] = {3, 2, 1, 0};
    int j = sizeof(i);
    j /= sizeof(i[0]);
    i[j - 1] = 2;
}
```

Notes:

- pour toutes les variables (y compris les pointeurs), “&variable” fait référence à l’adresse de cette variable, “variable” fait référence au contenu de la variable.
- Pour les pointeurs, “*pointeur” fait référence au contenu de la case dont l’adresse est dans “pointeur”.

Questions:

- Quelle est la valeur de i en fin de programme?
- Quelle est la valeur de p en fin de programme?
- Quelle est la valeur de *p en fin de programme?

La pile: les tableaux (2)

4

Code à simuler :

```
void main(void)
{
    int i[4] = {3, 2, 1, 0};
    int j;
    for (j = 0; j < 4; ++j)
    {
        i[j] += 1;
    }
}
```

Notes:

- "sizeof" fournit la taille en octets occupé par un type ou une variable.
- Dans le cas d'un tableau de pile, sizeof renvoie le nombre total d'octets utilisé par le tableau
- Dans le cas d'un tableau de pile, "tableau" vaut l' adresse de début du tableau. "tableau[i]" est équivalent à *(tableau + i). C'est à dire la case à l'adresse tableau + i*sizeof(type).

Questions:

- Quelle est la valeur de j en fin de programme?
- Quelles sont les valeurs du tableau en fin de programme?

La pile: les tableaux (3)

5

Code à simuler :

```
void main(void)
{
    char i[6] = {'f', 'c', 'j', 'j', 'm', '\0'};
    char *p = i;
    while (*p != '\0')
    {
        *p += 2;
        p += 1;
    }
}
```

Notes:

- Le codage de César consiste à crypter un message en effectuant une rotation circulaire des lettres de l' alphabet. Il était utilisé par Jules César pour ses correspondances secrètes (d'où le nom!).

Questions:

- Quelle est la valeur de p en fin de programme?
- Quelles sont les valeurs du tableau en fin de programme?

La pile: les structures (1)

Code à simuler :

```
struct Point
{
    int x, y;
};
int main(void)
{
    struct Point p = {1, 2};
    struct Point q = p;
    q.x = 0;
    q.y = 0;
}
```

Notes:

- la copie de structure se fait champs par champs.

Questions:

- Quelle est la valeur de p en fin de programme?
- Quelle est la valeur de q en fin de programme?

La pile: les structures (2)

Code à simuler :

```
struct Point
{
    int x, y;
};
int main(void)
{
    struct Point p = {1, 2};
    struct Point *q = &p;
    q->x = 0;
    q->y = 0;
}
```

Notes:

- L'opérateur "`q->x`" est équivalent à "(*q).x"

Questions:

- Quelle est la valeur de p en fin de programme?
- Quelle est la valeur de q en fin de programme?

La pile: listes chainées

8

Code à simuler :

```
struct Chainon
{
    char v;
    struct Chainon *suivant;
};
void main(void)
{
    struct Chainon chainons[] = {{0, NULL}, {1, NULL}, {2, NULL}};
    chainons[0].suivant = chainons + 2;
    chainons[2].suivant = chainons + 1;
    struct Chainon *p = chainons;
    while (p != NULL)
    {
        printf("%d\n", p->v);
        p = p->suivant;
    }
}
```

Notes:

- NULL est une constante définie dans stdlib.h qui représente une adresse invalide.

Questions:

- Quelle est l'affichage du programme?
- Quelle est la valeur de p en fin de programme?
- Que ce passe-t-il si on remplace "chainons+1" par "chainons+0"?

Appel de fonction: fonction sans paramètre

9

Code à simuler :

```
int zero(void)
{
    int i = 0;
    return i;
}
int main(void)
{
    int i = 10;
    i = zero();
}
```

Notes:

- Si une fonction a un type de retour différent de "void" alors il faut réserver l'espace pour cette valeur de retour dans la pile avant l'appel à la fonction.

Questions:

Appel de fonction: fonction avec paramètre

10

Code à simuler :

```
void plus1(int i)
{
    i = i + 1;
}
int main(void)
{
    int i = 10;
    plus1(i);
}
```

Notes:

- Le passage de paramètre à une fonction se fait toujours par copie de valeurs.

Questions:

- Quelle est la valeur de i en fin de programme?

Appel de fonction: fonction avec paramètre

11

Code à simuler :

```
int plus1(int i)
{
    return i + 1;
}
int main(void)
{
    int i = 10;
    i = plus1(i);
}
```

Notes:

- Le passage de paramètre à une fonction se fait toujours par copie de valeurs.
- N'oubliez pas de réserver l'espace nécessaire à la valeur de retour.

Questions:

- Quelle est la valeur de i en fin de programme?

Appel de fonction: fonction avec paramètre pointeur

12

Code à simuler :

```
void plus1(int *i)
{
    *i = *i + 1;
}
int main(void)
{
    int i = 10;
    plus1(&i);
}
```

Notes:

- Le passage de paramètre à une fonction se fait toujours par copie de valeurs

Questions:

- Quelle est la valeur de i en fin de programme?

Appel de fonction: fonction et valeur de retour

13

Code à simuler :

```
int *dix(void)
{
    int i = 10;
    return &i;
}
int main(void)
{
    int i = 100;
    int *p = dix();
    i = *p;
}
```

Notes:

- Une fois la valeur de retour copiée, on dépile les variables locales. Il faut alors considérer leur mémoire comme invalide.

Questions:

- Quelle est la valeur de i en fin de programme?
-

Appel de fonction: fonction et valeur de retour

14

Code à simuler :

```
int *dix(void)
{
    int i = 10;
    return &i;
}
int zero(void)
{
    int i = 0;
    return i;
}
int main(void)
{
    int i = 20;
    int *p = dix();
    zero();
    i = *p;
}
```

Notes:

- Une fois la valeur de retour copiée, on dépile les variables locales. Il faut alors considérer leur mémoire comme invalide.

Questions:

- Quelle est la valeur de i en fin de programme?

Appel de fonction: fonction et valeur de retour

15

Code à simuler :

```
int *indice(int *t, int i)
{
    return t + i;
}
int main(void)
{
    int t[] = {0, 1, 2};
    int *p = indice(t, 1);
    *p += 1;
}
```

Notes:

- Une fois la valeur de retour copiée, on dépile les variables locales. Il faut alors considérer leur mémoire comme invalide.

Questions:

- Quelle est la valeur de t en fin de programme?
- Quelle est la valeur de p en fin de programme?

Appel de fonction: fonction et valeur de retour

16

Code à simuler :

```
char *indice(char *t, int i)
{
    return t + i;
}
int main(void)
{
    char t[] = "helko";
    *indice(t, 3) += 1;
}
```

Notes:

- Une fois la valeur de retour copiée, on dépile les variables locales. Il faut alors considérer leur mémoire comme invalide.

Questions:

- Quelle est la valeur de t en fin de programme?

Appel de fonction: cohérence / header

17

Code à simuler :

```
int *f(char *t, int i);
int main(void)
{
    char t[] = "helko";
    *f(t, 3) += 1;
}
// dans un autre fichier :
int *f(int i, char *k)
{
    return k + i;
}
```

Notes:

- Cet exemple montre l'importance des fichiers d'entête pour garantir la cohérence entre le site d'appel et l'implémentation.

Questions:

Appel de fonction: récursivité

18

Code à simuler :

```
int mult(int a, int b)
{
    if (b == 1)
        return a;
    return a + mult(a, b - 1);
}
int main(void)
{
    int i = 10, j = 3;
    mult(i, j);
}
```

Notes:

- Une fonction récursive se caractèrise par le fait qu'elle se rappelle elle-même sur un problème plus petit.
- Pour qu'elle soit valide, une fonction récursive doit toujours avoir une condition d'arrêt.

Questions:

- Combien de fois la fonction mult est-elle appelée?
- Que se passe-t-il si j vaut -1?
- En tenant compte de l' adresse de début de pile, quelle est la plus grande valeur possible pour b?

Appel de fonction: récursivité

19

Code à simuler :

```
void mult(int a, int b, int *result)
{
    if (b == 1)
        *result = a;
    else
    {
        mult(a, b - 1, result);
        *result += a;
    }
}
int main(void)
{
    int i = 10, j = 3, r;
    mult(i, j, &r);
}
```

Notes:

- Lorsque le traitement est fait après l'appel récursif on parle de post-traitement

Questions:

- Combien de fois la fonction mult est-elle appelée?
- Que vaut r à la fin du progamme?
- En tenant compte de l'adresse de début de pile, quelle est la plus grande valeur possible pour b?

Appel de fonction: récursivité

20

Code à simuler :

```
void mult(int a, int b, int *result)
{
    if (b == 1)
        *result = a;
    else
    {
        *result += a;
        mult(a, b - 1, result);
    }
}
int main(void)
{
    int i = 10, j = 3, r;
    mult(i, j, &r);
}
```

Notes:

- Lorsque le traitement est fait avant l'appel récursif on parle de pré-traitement

Questions:

- Combien de fois la fonction mult est-elle appelée?
- Que vaut r à la fin du progamme?
- En tenant compte de l'adresse de début de pile, quelle est la plus grande valeur possible pour b?

Allocation dynamique: malloc

21

Code à simuler :

```
int main(void)
{
    char *p;
    p = malloc(sizeof(char) * 6);
    p[0] = 'h';
    p[1] = 'e';
    p[2] = 'l';
    p[3] = 'l';
    p[4] = 'o';
    p[5] = '\0';
    p = "world";
}
```

Notes:

- La fonction malloc renvoie l'adresse de début d' une zone réservée (vis à vis d'autre malloc) dont la taille est passée en argument.
- On dit qu'il y a une fuite mémoire si toute la mémoire obtenue par malloc n'a pas été rendu à la fin du programme à l'aide de la fonction free.

Questions:

- Quel est le contenu de la zone mémoire allouée en fin d' execution?
- Quel est le contenu de p en fin de programme?
- Y-a-t-il une fuite mémoire?

Allocation dynamique: malloc et free

22

Code à simuler :

```
int main(void)
{
    int *p;
    p = malloc(sizeof(int) * 2);
    p[0] = 'A';
    p[1] = 'B';
    free(p);
}
```

Notes:

- La fonction malloc renvoie l'adresse de début d' une zone réservée (vis à vis d'autre malloc) dont la taille est passée en argument.
- On dit qu'il y a une fuite mémoire si toute la mémoire obtenue par malloc n'a pas été rendu à la fin du programme à l'aide de la fonction free.

Questions:

- Quel est le contenu de la zone mémoire allouée en fin d' execution?
- Quel est le contenu de p en fin de programme?
- Y-a-t-il une fuite mémoire?

Allocation dynamique: malloc et free (2)

23

Code à simuler :

```
int *newArray(int size)
{
    return malloc(sizeof(int) * size);
}
int main(void)
{
    int *p = newArray(2);
    free(p);
}
```

Notes:

- La fonction malloc renvoie l'adresse de début d' une zone réservée (vis à vis d'autre malloc) dont la taille est passée en argument.
- On dit qu'il y a une fuite mémoire si toute la mémoire obtenue par malloc n'a pas été rendu à la fin du programme à l'aide de la fonction free.

Questions:

- Quel est le contenu de la zone mémoire allouée en fin d' execution?
- Quel est le contenu de p en fin de programme?
- Y-a-t-il une fuite mémoire?

Allocation dynamique: matrice 1

24

Code à simuler :

```
int *newArray(int size)
{
    return malloc(sizeof(int) * size);
}
int main(void)
{
    int *p[2];
    int i, j;
    p[0] = newArray(2);
    p[1] = newArray(2);
    for (i = 0; i < 2; ++i)
        for (j = 0; j < 2; ++j)
            p[i][j] = i + j;
    free(p[0]);
    free(p[1]);
}
```

Notes:

Questions:

- Quel est le contenu de la zone mémoire allouée en fin d' execution?
- Quel est le contenu de p en fin de programme?
- Y-a-t-il une fuite mémoire?

Allocation dynamique: matrice 2

25

Code à simuler :

```
int *newArray(int size)
{
    return malloc(sizeof(int) * size);
}
int main(void)
{
    int **p = malloc(sizeof(int *) * 2);
    int i, j;
    p[0] = newArray(2);
    p[1] = newArray(2);
    for (i = 0; i < 2; ++i)
        for (j = 0; j < 2; ++j)
            p[i][j] = i + j;
    free(p[0]);
    free(p[1]);
}
```

Notes:

Questions:

- Quel est le contenu de la zone mémoire allouée en fin d' execution?
- Quel est le contenu de p en fin de programme?
- Y-a-t-il une fuite mémoire?

Divers: bug

26

Code à simuler :

```
int main(void)
{
    char tab[4];
    char c;
    for (c = 0; c < 5; ++c)
        tab[c] = 0;
}
```

Notes:
Questions:

- Trouver le bug!

Algorithmique: strlen

27

Code à simuler :

```
int strlen(char *s)
{
    int l = 0;
    while (s[l] != '\0')
        l++;
    return l;
}
int main(void)
{
    char s[] = "hello";
    strlen(s);
}
```

Notes:
Questions:

- Quelle est la complexité de strlen ?

Algorithmique: strcmp

28

Code à simuler :

```
int strcmp(char *a, char *b)
{
    int i = 0;
    while (a[i] != '\0' && b[i] != '\0')
    {
        if (a[i] < b[i])
            return -1;
        if (a[i] > b[i])
            return 1;
        i++;
    }
    if (a[i] == '\0')
    {
        if (b[i] == '\0')
            return 0;
        else
            return -1;
    }
    return 1;
}
int main(void)
{
    char s1[] = "abba";
    char s2[] = "ac";
    char b = (s1 == s2);
    char c = strcmp(s1, s2);
}
```

Questions:

- Quelle est la complexité de strcmp?

Algorithmique: tri

29

Code à simuler :

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
void sort(int *t, int n)
{
    int i, j;
    if (n <= 1)
        return;
    i = 0;
    for (j = 1; j < n; ++j)
        if (t[j] < t[i])
            i = j;
    if (i > 0)
        swap(t, t + i);
    sort(t + 1, n - 1);
}
int main(void)
{
    int *t = malloc(sizeof(int) * 5);
    int i;
    for (i = 0; i < 5; ++i)
        t[i] = 5 - i;
    sort(t, 5);
}
```

Questions:

- Quelle est la complexité de l' algorithme?
- Combien de fois la fonction swap est appelée? Quel est le pire des cas (celui qui engendre le plus d'appels à swap)?

Algorithmique: tri (2) — 30

Code à simuler :

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition(int *t, int s, int pivot)
{
    int i = 0, j = 0;
    swap(t + pivot, t + s - 1);
    for (i = 0; i < s - 1; ++i)
        if (t[i] <= t[s - 1])
        {
            swap(t + i, t + j);
            j += 1;
        }
    swap(t + j, t + size - 1);
    return j;
}
```

```
void sort(int *t, int start, int end)
{
    int pivot;
    if (start < end)
    {
        pivot = start + random() % (end - start);
        pivot = partition(t, end, pivot);
        sort(t, start, pivot - 1);
        sort(t, pivot + 1, end);
    }
}
int main(void)
{
    int *t = malloc(sizeof(int) * 5);
    int i;
    for (i = 0; i < 5; ++i)
        t[i] = 5 - i;
    sort(t, 0, 5);
}
```

Questions:

- Quelle est la complexité de l' algorithme?
- Combien de fois la fonction swap est appelée? Quel est le pire des cas (celui qui engendre le plus d'appels à swap)?