

# Révisions

Vincent Lepetit

`vincent.lepetit@epfl.ch`

## Qu'affiche le code suivant ? (1)

```
void f(int * m, int n) {  
    if (*m < n)  
        *m = n;  
    n = 0;  
}
```

```
int * g(int * m, int * n) {  
    if (*m < *n)  
        return n;  
    else  
        return m;  
}
```

```
int a = 1, b = 2;  
b = *(&a);  
a = *(&b);  
cout << "A) " << a << ", " << b << endl;
```

```
a = 1; b = 2;  
f(&a, a);  
cout << "B) " << a << ", " << b << endl;
```

```
a = 1; b = 2;  
f(&a, b);  
cout << "C) " << a << ", " << b << endl;
```

```
a = 1; b = 2;  
int * p = g(&a, &b);  
int * q = g(&b, &a);
```

```
cout << "D) " << *p << ", " << *q << endl;
```

# Qu'affiche le code suivant ? (2)

```
struct Toto
{
    int * p;
    int a;
};

void ft(Toto * t, int * b, int * c) {
    t->p = b;
    t->a = *c;
}

Toto T;
int a = 0, b = 10;
ft(&T, &a, &b);
cout << "A) " << *(T.p) << "; " << T.a << endl;

T.a = 5;
T.p = &(T.a);
T.a = 6;
cout << "B) " << *(T.p) << "; " << T.a << endl;
```

## Qu'affiche le code suivant ? (3)

```
struct Toto
{
    int * p;
    int a;
};

Toto * gt(Toto * t, int n) {
    if (n == 0)
        return t;
    else {
        Toto * u = new Toto;
        u->p = new int[n];
        u->a = n;
        for(int i = 0; i < n; i++)
            u->p[i] = (i + 1) * (i + 1);
        return u;
    }
}

...
Toto T;
Toto * P;
int a = 0, b = 10;

T.p = &b;
T.a = b;
P = gt(&T, a);
cout << "C) " << *(P->p) << "; " << P->a << endl;

P = gt(&T, b);
cout << "D) " << *(P->p) << "; " << P->a << endl;
```

# Qu'affiche le code suivant ? (4)

```
int T[10];

for(int i = 0; i < 10; i++)
    T[i] = i * i;

int * p = T, * q = &(T[2]);

cout << "A) " << *p << ", " << p[0] << ", " << p[1] << endl;

p++;

cout << "B) " << *p << ", " << p[0] << ", " << p[1] << endl;

cout << "C) " << *q << ", " << q[0] << ", " << q[1] << endl;

p = q + (q - p);

cout << "D) " << *p << ", " << p[0] << ", " << p[1] << endl;
```

# Structures

Soit les structures suivantes:

```
struct Etudiant
{
    char nom[100], prenom[100];
    char section[5];
    float note;
};
```

```
struct Liste_Etudiants
{
    Etudiant * etudiants;
    int nb_etudiants;
    float moyenne;
};
```

**Écrivez les fonctions suivantes:**

- `Liste_Etudiants * alloue(int n)` qui crée une structure `Liste_Etudiants` pour stocker `n` structures `Etudiant`.
- `void calcule_moyenne(Liste_Etudiants * liste)` qui calcule le champ `moyenne` de `liste` à partir des notes des étudiants.

# Tableaux à 3 dimensions

On souhaite représenter une animation, les images étant faites à partir de caractères sous la forme d'un tableau à 3 dimensions de caractères:

```
const int N = 20;  
const int M = 15;  
const int NT = 100;
```

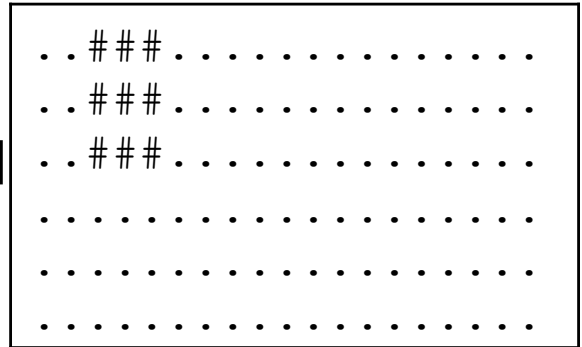
un tableau déclaré par:

```
char anim[NT][N][M];
```

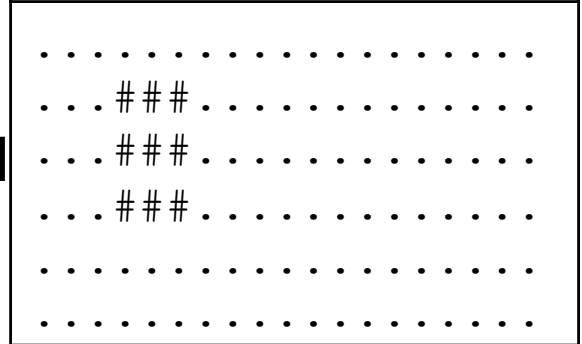
permettra de contenir NT images de N colonnes et M lignes.

Le caractère `anim[t][i][j]` sera le caractère de l'image au temps t, placé sur la i<sup>ème</sup> colonne et j<sup>ème</sup> ligne.

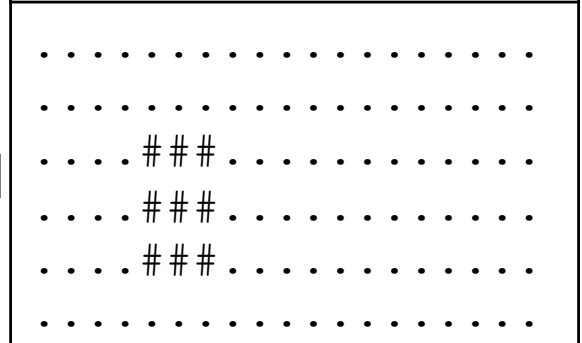
**anim[0]**



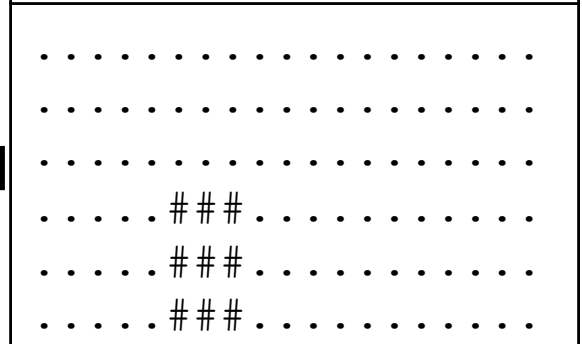
**anim[1]**



**anim[2]**



**anim[3]**



# Particules

Le but de cet exercice est de représenter l'évolution de particules dans un champ de pesanteur constant, le tout restreint à un espace 2D.

On utilisera la structure suivante pour représenter une particule:

```
struct Particule
{
    float x, y;
    float vx, vy;
};
```

où  $x$  et  $y$  représentent les coordonnées de la particule, et  $v_x$  et  $v_y$  son vecteur vitesse.

# Écrire les fonctions suivantes

```
struct Particule
{
    float x, y;
    float vx, vy;
};
```

$x$  et  $y$  représentent les coordonnées de la particule, et  $v_x$  et  $v_y$  son vecteur vitesse.

- `void init(Particule * p, int w, int h)`

qui initialise  $p$  aux coordonnées  $(w / 2, h)$ , sa vitesse sera initialisée aléatoirement:  $v_x$  sera compris entre -1 et 1 (en flottant), et  $v_y$  entre -5 et -15 (en flottant également). [4 lignes]

- `bool dehors(Particule * p, int w, int h)`

qui renvoie vraie si  $p$  se trouve en dehors du rectangle défini par  $(0, 0) \times (w, h)$ . [1 ligne]

- `void avance(Particule * p, float gx, float gy, float dt)`

qui modifie la vitesse et la position de la particule placée dans le champ de pesanteur défini par  $g_x$  et  $g_y$ , pendant l'intervalle de temps  $dt$  supposé petit.

On aura donc:

nouvelle vitesse  $\leftarrow$  ancienne vitesse +  $g * dt$

nouvelle position  $\leftarrow$  ancienne position + nouvelle vitesse \*  $dt$